

# **Developing interactive systems**

a perspective on supporting ill-structured work



# **Developing interactive systems**

a perspective on supporting ill-structured work

## **Proefschrift**

ter verkrijging van de graad van doctor  
aan de Technische Universiteit Delft,  
op gezag van de Rector Magnificus prof. ir. K.F. Wakker,  
voorzitter van het College voor Promoties,  
in het openbaar te verdedigen op maandag 19 november 2001 om 16.00 uur  
door Johannes Jacobus DE GRAAFF  
informatica ingenieur  
geboren te 's-Gravenzande

Dit proefschrift is goedgekeurd door de promotor:  
Prof. dr. H.G. Sol, Technische Universiteit Delft

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. H.G. Sol,	Technische Universiteit Delft, promotor
Prof. dr. J. Aasman,	Technische Universiteit Delft
Prof. dr. J.H.T.H. Andriessen,	Technische Universiteit Delft
Prof. dr. ir. J.L.G. Dietz,	Technische Universiteit Delft
dr. ir. C.A.P.G. van der Mast,	Technische Universiteit Delft
Prof. dr. W.G. Vree,	Technische Universiteit Delft
Prof. dr. R.W. Wagenaar,	Technische Universiteit Delft

*Published and distributed by:* DUP Science

DUP Science is an imprint of  
Delft University Press  
P.O. Box 98  
2600 MG Delft  
The Netherlands  
Telephone: +31 15 278 5121  
Telefax: +31 15 278 1661  
Email: DUP@Library.TUdelft.NL

ISBN: 90-407-2248-X

Keywords: interactive systems, design, ill-structured work

Copyright ©2001 by Hans de Graaff (hans@degraaff.org)

All rights reserved. No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission of the publisher: Delft University Press.

Printed in The Netherlands

---

## Contents

---

<b>Preface</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 The interaction dilemma . . . . .	2
1.1.1 Examples of the interaction dilemma . . . . .	3
1.1.2 Evidence for the downside of the dilemma . . . . .	5
1.2 History of the interaction dilemma . . . . .	7
1.2.1 User-centered design: a possible solution? . . . . .	8
1.2.2 Origins of HCI . . . . .	8
1.2.3 Evolution of HCI . . . . .	9
1.2.4 Consequences of the HCI evolution . . . . .	11
1.3 Research questions . . . . .	13
1.4 Research approach . . . . .	15
1.4.1 Research philosophy . . . . .	15
1.4.2 Research strategy . . . . .	17
1.4.3 Research instrument . . . . .	19
1.5 Thesis outline . . . . .	19
<b>2 DIANA– a case study</b>	<b>21</b>
2.1 Case study description . . . . .	21
2.1.1 TNO Building and Construction Research . . . . .	22

## CONTENTS

---

2.1.2	Finite element analysis . . . . .	22
2.1.3	The I-DIANA project . . . . .	25
2.1.4	Relation between the case study and the I-DIANA project . . . . .	26
2.2	I-DIANA Project approach . . . . .	27
2.2.1	I-DIANA constraints . . . . .	27
2.2.2	Analysis approach . . . . .	27
2.2.3	Design approach . . . . .	29
2.3	Analysis of current and future use of DIANA . . . . .	31
2.3.1	The Questionnaire: a first profile of DIANA users . . . . .	31
2.3.2	The interviews: fleshing out DIANA's usage . . . . .	35
2.3.3	Task analysis: a formal analysis of DIANA's usage . . . . .	36
2.4	Design of a graphical user interface for DIANA . . . . .	39
2.4.1	Approach for creating a conceptual design . . . . .	39
2.4.2	Prototyping for technical feasibility . . . . .	42
2.4.3	Conceptual design: 1st generation . . . . .	43
2.4.4	Conceptual design: 2nd generation . . . . .	44
2.4.5	Conceptual design: 2nd generation, revised . . . . .	46
2.5	Reflections . . . . .	50
2.5.1	Reflections on the analysis process . . . . .	50
2.5.2	Reflections on the design phase . . . . .	52
2.5.3	The case study and the research questions . . . . .	55
<b>3</b>	<b>Human Computer Interaction</b> . . . . .	<b>57</b>
3.1	Terminology . . . . .	58
3.1.1	The development process . . . . .	58
3.1.2	Interactive systems . . . . .	59
3.1.3	Tasks . . . . .	61
3.1.4	Ill-structured work . . . . .	62
3.2	Support for analysis . . . . .	64
3.2.1	Task Analysis methods . . . . .	65
3.2.2	Traditional task modeling . . . . .	65
3.2.3	Adding knowledge to task structures . . . . .	67
3.2.4	Task modeling for ill-structured work . . . . .	69
3.2.5	Other developments in interactive systems analysis . . . . .	70
3.2.6	Discussion . . . . .	73
3.3	Support for conceptual design . . . . .	73
3.3.1	Examples of design tools in general . . . . .	74
3.3.2	Examples of design tools for interactive systems . . . . .	75
3.3.3	Representation of interaction as a design tool . . . . .	80

3.3.4	Real world requirements for design support . . . . .	83
3.3.5	Discussion . . . . .	85
3.4	Multi-disciplinary design teams . . . . .	86
3.4.1	Disciplines and their cultures . . . . .	86
3.4.2	The constituents of a design team . . . . .	87
3.4.3	Discussion . . . . .	88
3.5	Conclusions and requirements . . . . .	89
3.5.1	The underlying paradigm . . . . .	90
3.5.2	Open issues . . . . .	90
3.5.3	Requirements and suggestions . . . . .	91
<b>4</b>	<b>A Workspace-Oriented Design Representation</b>	<b>95</b>
4.1	WONDER's central concept: the workspace . . . . .	95
4.1.1	The watchmaker's workplace . . . . .	96
4.1.2	Analysis of the watchmaker's workplace . . . . .	100
4.1.3	The workspace as the basis of a design representation . . . . .	100
4.2	Way of thinking . . . . .	101
4.2.1	Foundations of the workspace concept . . . . .	101
4.2.2	The elements of WONDER. . . . .	103
4.3	Way of modeling . . . . .	105
4.3.1	Types of models . . . . .	105
4.3.2	Media for models . . . . .	106
4.3.3	WONDER elements . . . . .	107
4.4	Way of working . . . . .	113
4.4.1	Finding workspaces . . . . .	115
4.4.2	Assessing workspaces . . . . .	115
4.4.3	Refining workspaces . . . . .	117
4.5	Way of controlling . . . . .	119
4.5.1	The design team . . . . .	120
4.6	Way of supporting . . . . .	123
4.7	The design process following WONDER . . . . .	125
4.8	Reflections on WONDER . . . . .	126
4.8.1	A look back at the requirements . . . . .	126
4.8.2	Assumptions on the use of WONDER . . . . .	128
4.8.3	Observations about WONDER . . . . .	130

## CONTENTS

---

<b>5</b>	<b>Computer support for workspace design</b>	<b>131</b>
5.1	Introduction . . . . .	131
5.1.1	Main areas of computer support for WONDER . . . . .	132
5.1.2	Requirements . . . . .	133
5.2	Computer support for WONDER . . . . .	134
5.2.1	Selection of tools and platforms . . . . .	134
5.2.2	Design . . . . .	135
5.2.3	Examples of presentation and editing . . . . .	136
5.2.4	Examples of overviews . . . . .	138
5.2.5	Search facilities . . . . .	144
5.3	Evaluation and recommendations . . . . .	145
5.3.1	Evaluation of use . . . . .	145
5.3.2	Recommendations . . . . .	146
<b>6</b>	<b>Use of WONDER in a design process</b>	<b>149</b>
6.1	SHIPSHAPE case description . . . . .	149
6.1.1	Shipyard planning in general . . . . .	149
6.1.2	SHIPSHAPE issues . . . . .	150
6.1.3	A good case for applying WONDER . . . . .	151
6.2	SHIPSHAPE's design in WONDER . . . . .	151
6.2.1	Describing a WONDER design . . . . .	152
6.2.2	SHIPSHAPE workspaces . . . . .	153
6.2.3	SHIPSHAPE materials . . . . .	155
6.2.4	Joint evolution of <i>workspace</i> and <i>material</i> . . . . .	156
6.2.5	The WONDER representations . . . . .	159
6.3	WONDER in use . . . . .	160
6.3.1	Finding workspaces . . . . .	160
6.3.2	Evolution . . . . .	163
6.3.3	Assessing workspaces . . . . .	164
6.3.4	Refining workspaces . . . . .	167
6.3.5	Design team . . . . .	168
6.3.6	Comparing use of WONDER to the task structures . . . . .	170
6.4	Testing the SHIPSHAPE design . . . . .	175
6.4.1	Test approach . . . . .	175
6.4.2	Test results . . . . .	182
6.5	Testing the assumptions . . . . .	189
6.5.1	Assumptions about the way of thinking . . . . .	190
6.5.2	Assumptions about the way of modeling . . . . .	192
6.5.3	Assumptions about the way of working . . . . .	194



6.5.4	Assumptions about the way of controlling . . . . .	195
6.5.5	Testing the way of supporting . . . . .	196
6.5.6	Other assumptions . . . . .	196
6.5.7	Additional observations . . . . .	197
<b>7</b>	<b>Conclusions</b>	<b>201</b>
7.1	The thesis in a nutshell . . . . .	201
7.1.1	WONDER and its roots . . . . .	202
7.1.2	Assumptions . . . . .	202
7.1.3	Reflecting the assumptions onto the research questions .	203
7.2	Reflection on the research . . . . .	205
7.2.1	Consequences of the research . . . . .	205
7.2.2	A broader look . . . . .	206
7.2.3	WONDER and the underlying paradigm . . . . .	207
7.3	Future research . . . . .	208
<b>A</b>	<b>Structure and operation of DIANA</b>	<b>211</b>
A.1	Overview . . . . .	211
A.2	Working with DIANA . . . . .	213
<b>B</b>	<b>PSM notation</b>	<b>219</b>
<b>C</b>	<b>WONDER representations for SHIPSHAPE</b>	<b>221</b>
C.1	SHIPSHAPE example workspace . . . . .	221
C.2	SHIPSHAPE example material . . . . .	223
	<b>Bibliography</b>	<b>225</b>
	<b>Summary</b>	<b>243</b>
	<b>Samenvatting</b>	<b>247</b>
	<b>Curriculum Vitae</b>	<b>251</b>

## *CONTENTS*

---

---

## List of Figures

---

1.1	Research activities for an inductive-hypothetic research strategy	18
2.1	Erasmus bridge, Rotterdam . . . . .	23
2.2	Example mesh of model with loads and constraints . . . . .	24
2.3	Task structure: construction of storage racks . . . . .	37
2.4	Decision structure: validate model . . . . .	39
2.5	Screenshot: MENTAT II mesh definition . . . . .	40
2.6	Overview of the I-DEAS user interface . . . . .	41
2.7	Early prototype screen shot of entering an input file . . . . .	48
3.1	Star life cycle [HH89] . . . . .	59
3.2	ADEPT models and processes [WJK <sup>+</sup> 93] . . . . .	68
3.3	Seaton and Stewart's task hierarchy . . . . .	76
3.4	UIDE database structure [FKKM91] . . . . .	79
3.5	UAN example—select file [HH93, p. 167] . . . . .	83
4.1	Example workplace of a watchmaker [Dan81] . . . . .	97
4.2	Watchmaker workspace—finishing bench . . . . .	98
4.3	Watchmaker workspace—lathes . . . . .	99
4.4	Watchmaker workspace—calibrated clock . . . . .	99
4.5	PSM diagram of workspaces, materials, and actions . . . . .	112
4.6	Task structure—creating a workspace model . . . . .	114

## LIST OF FIGURES

---

4.7	Task structure—Assessing workspaces . . . . .	116
4.8	Refinement step 1: creating workspace descriptions . . . . .	118
4.9	Refinement step 2: cleaning up a workspace . . . . .	119
5.1	Makefile (excerpt) . . . . .	136
5.2	Text based workspace description (excerpt) . . . . .	137
5.3	HTML presentation of workspace description . . . . .	139
5.4	List of all available descriptions (excerpt) . . . . .	140
5.5	Table of workspace relations (excerpt) . . . . .	141
5.6	Workspace hierarchy (excerpt) . . . . .	142
5.7	Workspaces per material (excerpt) . . . . .	143
5.8	Workspaces per material (excerpt) . . . . .	144
5.9	Search facility including search results . . . . .	145
5.10	Example of integrated editing system . . . . .	147
6.1	SHIPSHAPE design timeline . . . . .	153
6.2	High-level hierarchy of workspaces for SHIPSHAPE . . . . .	154
6.3	Example action ‘create’ . . . . .	160
6.4	Part of an early design diagram . . . . .	162
6.5	Design diagram with annotations . . . . .	163
6.6	Screenshot from presentation . . . . .	176
6.7	Capacity view (screenshot) . . . . .	179
6.8	Activity view (screenshot) . . . . .	180
6.9	Visual addition to WONDER representation (detail) . . . . .	193
A.1	DIANA’s user environment . . . . .	212
A.2	DIANA program structure . . . . .	214
A.3	Syntax specification for load calculation . . . . .	215
A.4	Example specification for load calculation . . . . .	216
A.5	Syntax specification for load specification . . . . .	216
A.6	Example of load specification . . . . .	217
B.1	PSM Graphical notation . . . . .	220

---

## List of Tables

---

4.1	Workspace representation definition . . . . .	109
4.2	Material representation definition . . . . .	110
4.3	Action representation definition . . . . .	111
4.4	Structuredness of elements . . . . .	113
4.5	Roles in the design team . . . . .	121
4.6	Requirements and suggestions implemented in WONDER . . . .	128
4.7	Assumptions on the use of WONDER . . . . .	129
6.1	SHIPSHAPE workspaces . . . . .	155
6.2	SHIPSHAPE materials . . . . .	156
6.3	SHIPSHAPE materials and workspaces . . . . .	157
6.4	Revisions of Maintain scope and Scope . . . . .	158
6.5	Contents of analysis documents . . . . .	161
6.6	Categorization of changes to design representations . . . . .	165
6.7	Role assignment in SHIPSHAPE design team . . . . .	169
6.8	Assessment changes to workspaces . . . . .	171
6.9	Occurrence of different clusters of changes . . . . .	172
6.10	Types of workspaces . . . . .	173
6.11	Contents of SHIPSHAPE presentation . . . . .	177
6.12	Support tools and their ratings on a scale of 1–5, with 5 being best	183
6.13	Results for “Does the workspace allow you to reach the stated goal?” . . . . .	184

*LIST OF TABLES*

---

6.14 Results for “How do the workspaces compare to the current tools?” . . . . .	185
6.15 How the currently used tool influences the comparison question	186
6.16 Missing items per workspace . . . . .	187
6.17 Quick comments per workspace . . . . .	188
6.18 Assumptions about WONDER and their results . . . . .	198

---

## Preface

---

It is said that writing a doctoral thesis is a learning experience. Looking back at the past years I can only concur. I've learned much more about design than I can present within the confines of this thesis. Design is an inspirational topic. Creating something original is simply a wonderful thing to do, no matter how small or large it is. Thinking about that act in a more abstract and philosophical way has certainly captured my attention. I hope that is reflected in this thesis.

Inspiration for my thinking during my research has come from more places than those mentioned in the bibliography. Books by Henri Petroski, for instance, discussing design from an inspiring engineering perspective [Pet85, Pet94]. Kevin Mullet's book on the design of visual interfaces, showing actual examples of the principals at work [MS95]. Some of the books Rodney Fuller tends to point out to me, such as a beautifully photographed book on packing gifts in traditional Japanese manner [Oka75]. It's a shame I can't bring all of this to your attention.

Many people have influenced this work in one way or the other. Most directly Henk Sol and Charles van der Mast. I want to thank Henk for his continuing support and patience, even when my procrastination must have driven him mad sometimes; and also for providing high-level but accurate comments on my drafts, while leaving the details to me. Charles, for introducing me to the field of HCI, for supporting me throughout my research, and for many invaluable revisions of and discussions about this thesis.

Ajanthe Dahanayake proved to be a great person to share a room with at

## LIST OF TABLES

---

the university. Her different perspective on things was always enjoyable and inspiring, and secretly I always was jealous with the ease with which she seems to research things. Equally enjoyable were the interactions with the other AIOs. We had many fun lunches, you were great company throughout the years at the university.

The people in the MMUIS group provided a great way to learn about research in the other Dutch universities and about the different ways in which a topic can be approached and researched. I think we shared enough common interests to be attracted to each other while also being different enough to keep it interesting. This also made our meetings so enjoyable.

I have to thank the people at KPN Research, my employer for the last few years, with their patience. It took a bit longer than planned before I could finish this work. In particular, Heidi Vogel for reading through some of the early manuscripts, and Angelien Sanderman for nagging me in many ways to get this finished. I don't feel sorry for all the fun projects I have done there in the last few years, even though they kept me from finishing this thesis quickly. And ladies, now it *is* time to party.

The fact that this thesis is written in decent American English is due to Miranda Aldham-Breary. She kindly showed me that my English writing wasn't as good as I thought it might be, but she also taught me to simply avoid making the same mistakes over and over again. Visiting her was always a joy, if only for her many cats. She also introduced me to my two current roommates Holly and Niels.

Finally, I want to end this preface by thanking my parents and brother for their support, for providing a wonderful family environment, and for giving me the freedom to find out things for myself.

*Hans de Graaff*



# CHAPTER 1

---

## Introduction

---

We live in an age of information. Whether this age will be seen as a new kind of industrial revolution is for future historians to decide, but it can not be denied that computers and their wide range of possible applications are rapidly changing our lives. Ever more data is being stored in computers, being combined and cross-checked; computers are used to control and monitor more and more parts of our lives. Computers also keep getting smaller and smaller, and they have started to turn up in disguise, i.e. in embedded systems such as microwaves, cars, and mobile phones.

The pervasiveness of and dependence on information systems in our society places a huge responsibility on the shoulders of the people who design, build and market such systems. They have to make sure people can actually deal with all this information, with all these systems, and that the promises of computing technology can be delivered. Unfortunately, this does not always happen. Careful reading of the news, ones own experiences, and stories heard from friends and colleagues can all add up to the conclusion that computers often make life harder instead of easier. As an issue of the Communications of the ACM put it succinctly: “so much research, yet so few good products” [cac96].

Interactive systems, i.e. systems such as ticket vending machines, spread-

sheets, and planning systems, which people use directly to get something done, pose additional problems over more traditional systems such as transaction systems, because they try to support more ambitious goals, and therefore offer more and more diverse functionality. With transaction systems the transactions determine how and in what order the functionality is accessed, and which functions are dealt with by the system, and which by the person operating it, a process known as function allocation. With interactive systems there is usually no such intrinsic order. Instead, all the functionality of the system always needs to be made available to the user. Determining a proper function allocation, and mapping these functions to specific interface elements in such a way that the user's goals still can easily be reached is a major challenge in designing interactive systems. Until now, in many such systems, this challenge has not been met.

We create and use interactive systems because we believe they enable us to do things we would not otherwise be able to do, or because they enhance our productivity. After the considerable success of transaction systems, e.g., salary administrations, it was believed that similar results could also be achieved for more complex work which involved decision making and problem solving. Researchers in the field of Artificial Intelligence (AI) have been trying to automate the decision making and problem solving processes in a manner comparable to that available for transaction systems. Although progress has been made, this has only been successful for very specific, often constrained, application domains. Whether decision making and problem solving can be fully automated is a long-standing philosophical debate with strong opponents [HD81] and proponents [Pen90], and as yet no clear outcome.

For now the best we can do is to provide computing support for problem solving and decision making, but even this has not met with as much success as support for transaction processing. Several studies claim that efficiency and effectiveness are not increased as much as they should for systems supporting decision making and problem solving [Lan95, Nor93]. In this thesis the issue of efficiency and effectiveness is explored, and a better approach for designing interactive systems is given. Hopefully use of this approach will allow our productivity to improve.

## 1.1 The interaction dilemma

Interactive systems present us with a dilemma. On the one hand we want or need those interactive systems to get complicated jobs done, but on the other

hand the use of these systems frustrates us, lowers our efficiency, and gets in the way of the job to be done. In this section examples of both sides of the dilemma are presented, followed by an explanation of which applications of information technology are most likely to suffer from this dilemma.

### 1.1.1 Examples of the interaction dilemma

Examples for the upside of the dilemma, that interactive systems can help us get complicated jobs done, can easily be found. Creating this thesis, for instance, is much easier with the support of a computer and word processor than it would be with pen and paper or even a typewriter. Computers allow me to confer with colleagues in different countries about the thesis through email or video conferencing. The chip-card with which I pay also ‘knows’ how much credit I have to make telephone calls, and could provide brief medical information if needed. Interactive systems also support surgeons examining heart conditions [dB95], and therapists helping people suffering from acrophobia [HKM<sup>+</sup>95].

The downside of the dilemma is much less recognized. Interactive systems often get in the way of getting the job done, for instance by preventing the work being done in a particular order, or by simply not supporting all the aspects of our work. For example, I have been making changes to the draft text of this thesis using pen and paper first, as opposed to using my word processor, because the word processor does not facilitate the kind of notes and annotations needed to work through a draft. Also, the positive aspects of the chip-card are offset by the fact that transactions will be much harder to keep private. Two good sources for an overview of such problems are the RISKS forum [Neu95, Ris] and several of the books by Norman [Nor88, Nor93, for example]. The following examples illustrate this downside of the dilemma.

Several years ago I was teaching adults word processing using WordPerfect. Most people in the class had never worked with computers before, and were not used to the way in which a computer deals with files. Much to my surprise many of them had trouble saving files after making changes to it. The problem was in the wording of the accompanying question. When a file is saved again after making changes, WordPerfect will ask the user to replace the file: Replace (Y/N): FILENAME.WP. This posed a dilemma for the people on the course: answering ‘yes’ seemed to imply that their *current* text would be replaced (but with what?), while ‘no’ seemed like the right answer, but did not actually save the file, presenting them with the same question later on.

Norman describes problems with a telephone system, and in particular its

‘hold’ function. He compares an old telephone, with a hold button and associated light, with a new telephone which requires a code of ‘#’ and some numbers to put a conversation on hold, and a similar sequence of keys to get the conversation back. As a result many people stopped using the hold function of the telephone altogether [Nor88].

These two problems, and many similar ones, are caused by some mistake in the design or realization of the system. The user of such a system is stopped either because of confusion or because it becomes impossible to continue, or the user simply avoids using the feature altogether, no matter the convenience it could provide. In the worst case the users will blame themselves for not understanding, and feel bad about this [Nor88]. Such problems are usually caused by some oversight of the designer, or by unexpected use of the system. Often they are easier to fix in hindsight than to avoid them in the first place. Fixing them is often as easy as changing the wording of a message or adding an additional button to the telephone.

These problems have not gone away in recent years, although their nature is changing. The accessibility issues as evidenced by the previous examples are often not an issue now, but the fix that has been applied often consists of keeping the user within a narrow, well-defined and ‘friendly’ process. Users are fine as long as they don’t venture off of the beaten track. For instance, my new video automatically adjusts the time to the teletext time broadcast on the first channel. While this avoids a number of problems in setting up the time correctly, it does pose a problem for me, because I have a channel in a different timezone from my own on the first channel, and there is no way to disable the automatic update.

On a similar note the strong coupling between Microsoft’s Exchange and Schedule+ applications brings many benefits, but it also has the unfortunate side-effect of locking the user out of the schedule when a mail message is being composed from within another application. In summary, often the fix is not to create a usable complex system, but rather to oversimplify it, and then lock the user into that solution.

This touches on one of the hallmarks of truly interactive systems: the fact that they should provide the user with a large amount of freedom to carry out work. Unlike transaction systems, there is no fixed, pre-defined path. This freedom of choice of action provides much of the power of interactive systems, along with other attributes such as responsiveness and visualization. The flexibility allows interactive systems to support ill-structured work, i.e. work for which no program has been described [Bot89]. In other words, user interfaces often are non-deterministic [Dix90].

This ability to support ill-structured work comes at a price: usability problems are less easily noticed. The lack of a program for, or description of, the ill-structured parts of the work makes it impossible to check the paths through the software, as can be done for transaction systems. The obvious errors will still surface, but problems with the functionality of the system will often go unnoticed. Examples of such problems are that functionality is located in odd or unexpected places (“I never would have thought I could also do feathering. Why isn’t it in the column menu where I would expect it!”), related functionality is not located closely together, that it is not clear which of all the possible options apply in a given context, or that the user is locked into a single path through the application.

Such usability problems often go by unnoticed. People can and will adjust to these situations, avoiding trouble areas, inventing kludges to get the job done, or simply not use the functionality at all. They often do not even consider the problems to be problems, instead thinking they do not really understand how it works or that their job is too complicated or different for the system they are using.

Even though the users do not generally notice these usability problems, this does not mean that these problems cannot have a large impact on their work. Trying to assert this turns out to be difficult, because asking the users will not work as they are not directly aware of the problems.

### 1.1.2 Evidence for the downside of the dilemma

Circumstantial evidence provided in this section suggests that the problems with interactive systems as sketched in the previous section do exist. The evidence is given in two parts. First, a macro-economic approach is used to look at the impact of information technology on productivity, and second, the results of usability studies of individual systems are discussed.

**Productivity** The positive side of the interaction dilemma suggests that we should see an increase in productivity as a result of the use of interactive systems, while the negative side suggests a decrease.

An investigation by Landauer suggests that the increase of productivity has dropped as a result of the introduction of information technology in the workplace, in particular when interactive systems have been introduced [Lan95]. His study shows a noticeable drop in the rate of productivity growth after 1973, and calculates that, on average, IT investments yield 13.3% less than

other investments. Landauer postulates that this loss is encountered because interactive systems often are not really relevant to productivity (e.g., generating nice-looking reports, or surfing the web), and because they often do not support those tasks which really influence productivity. This latter argument is supported by Taylor, who claims office automation often fails because of “an erroneous conception of what an office is, what people do in it, and what the practice of management amounts to in the present age” [Tay93]. He argues that office automation in its current state is flawed, mostly because it simply does not address the necessities of the office, the organization, and the office worker, a point that is also stressed by Hammer [Ham90]. In a study of computer aided design (CAD) tools, Liker has found that support for creative engineering is even more important to get right than the more functional, analytical engineering tasks [LFNZ92].

Others take issue with the conclusions of Landauer’s study, better known as the ‘productivity paradox’. Brynjolfsson, for instance, argues that more accurate measuring and better calculations do not show this drop [Bry93]. He also argues that, instead of just looking at a single number for productivity, additional metrics need to be taken into account, such as user satisfaction, e.g. when a reports looks better. In other words, the drop in productivity might be made up for in other metrics such as consumer value, they might just fall outside the productivity scope [HB95].

Whether there really is a productivity paradox is debatable and depends on the calculations used and the assumptions made. At the very least we can say that the rate of productivity growth has not increased, as the positive side of the interaction dilemma argues.

**Usability** For the second part of the evidence we look at individual systems. Nielsen cites several studies into the effects of usability engineering [Nie93]. In each of these cases the cost savings resulting from fixing usability problems found in the products were significant, in particular when compared to the cost of carrying out the usability work. The evidence here is that each of the systems under study had usability problems, similar to the ones outlined in the previous section, leading to a loss of productivity.

The users of the systems described in these studies were lucky. The problems with their systems were fixed, although sometimes after being introduced in the workplace. How many systems have not had such usability checks? Many of the problems found during such usability checks can be fixed fairly easily, for instance by changing the wording of a message. Some of the prob-

lems, however, can be much larger than that, and tied to the whole architecture or metaphor of the system. In these cases only a major redesign can really solve the problems. For these problems prevention is much better than trying to fix them afterwards.

**In summary** The interaction dilemma, then, is that we may try to create interactive systems to get more and more complicated jobs done, but often these systems will not help us as much as we would like, sometimes to the point of being detrimental. This seems to be true in particular for interactive systems which support a much more flexible approach to work than transaction systems, as their projected benefits are usually larger and more visible than their real drawbacks and problems.

## 1.2 History of the interaction dilemma

The examples and circumstantial evidence of the downside of the interaction dilemma suggest that many interactive systems do not provide proper support for the job they are meant to support. Both the arguments for the productivity paradox and the usability problems found suggest that the design process of interactive systems should be more sensitive to the work that needs to be supported, and to the people who need to operate these systems.

The design process can be influenced by a better understanding of organizations, and the way in which these organizations can be supported by information technology in general. This influence is provided by the field of Information Systems (IS). While IS can improve the way in which interactive systems are incorporated in an organization, it does not address explicitly the people working with these systems. This aspect is covered by the fields of Computer Supported Cooperative Work (CSCW), which addresses the support of people working together in both a social and a technical sense, and Human-Computer Interaction (HCI), which is concerned with the interaction of a person with a computer system.

To make the research in this thesis more manageable I have decided to concentrate on supporting a single person using a single interactive system while assuming the cooperation of this person with other people does not need active support, and that the organization and the work of the person remain the same. In other words, I look at this problem from the perspective of HCI.

### 1.2.1 User-centered design: a possible solution?

The problems surrounding the use of interactive systems have been noted for quite a while in the field of HCI. In 1986 Norman and Draper edited a book about the importance of user-centered design [ND86]. In their Human Machine Interaction project at the University of California in San Diego they came to see the importance of a user-centered approach. The book is, as the authors acknowledge, a book of questions. All authors recognize the importance of user-centered design, and they can explain why such a perspective is important, but specific instructions on how to achieve user-centered design are lacking.

Their efforts have been repeated during the last ten years. Each time many examples of problems with interactive systems are cited, and user-centered design, or something similar, is identified as the solution [Gou88, Lan95, e.g.]. In that sense, Landauer's book is just a continuation of the issues raised ten years earlier by Norman et al..

This observation begs the question: if the problems, and the general solution have been known for some time, then why have we not seen any obvious progress during the last ten years? This question can be answered by a brief overview of the origins and evolution of the field of HCI. This overview is partly based on articles on the history of HCI and usability as described by Butler [But96] and Myers [Mye96].

### 1.2.2 Origins of HCI

The origins of HCI can be found in three fields: human factors, psychology, and computer science [MG95]. The field of human factors started early this century, studying the interactions between people and machines. During World War II more and more complicated machinery was being built, which boosted the human factors field. In recent years human factors has evolved to also include computers and interactive systems, because many of the principles and guidelines from interaction with mechanical machines can also be applied to interaction with computers.

Psychology, cognitive psychology in particular, takes another view on HCI, trying to explain and predict how people perform with interactive systems. An early example of this is 'Fitts Law', describing the relation between speed and accuracy in human movement, making it a prime candidate for mouse-based graphical systems [Fit54]. Additional research continued to look at more detailed aspects of interactive systems and the psychological aspects of using



them. A seminal work in this area is the Goals, Operators, Methods, and Selection rules (GOMS) model, which claims to predict human performance for different variants of an interactive system [CMN83]. More work is needed in this area to not just understand what is going on in human-machine interaction, but also to actively guide the design process [Kir95].

Computer science has been a dominant field in HCI through the development of new software techniques and new possibilities for interaction. The development of graphical interfaces opened up a whole new realm of possibilities. Technological improvements have continued since then, currently resulting in the ubiquitous direct manipulation interfaces with mouse and bit-mapped display. Computer scientists have been pushing technology at a high pace, making it hard for the two other original fields to keep up with new possibilities and applications.

The three fields, together with some other fields such as graphical design and social sciences, have been merging slowly. A single unified HCI field has only recently begun to emerge [Sea96].

### 1.2.3 Evolution of HCI

Theory on design recognizes three main stages in development: analysis, design, and realization [Jon92]. The evolution of HCI has followed these three stages in an interesting way, starting with realization, then moving to analysis, and only recently shifting to design. The shift from realization to analysis to design can also be observed in the development of User Interface Management Systems (UIMSs) [HH89]. Hartson and Hix describe several generations of UIMSs, and even though their classification is not based on time, the advancements in UIMSs show a similar evolution towards supporting the design, starting with support for realization.

**Realization** The initial focus of the emerging field of HCI was on realizing interactive systems. This focus was primarily driven through technology push, instigated in the computer science field, because its workers possessed and created the technology to actually build such systems. This resulted in a strong engineering focus on interactive systems, with much attention for new possibilities and techniques associated with the new technology. Typical conference paper topics from that time include direct manipulation, use of windows, and graphical interfaces in general, the mouse, and the application of these concepts to several application domains such as word processing and drawing

[BC85, Jan83, e.g.].

Once the basic components of a graphical system were understood, focus shifted to supporting the realization of such interfaces. It was clear that building graphical interactive systems was no small task, and supporting tools and methodologies were needed. Examples of such realization oriented research are: the creation of toolkits [LVC89, Ope91, Sun89]; dialogue descriptions [Gre86]; interface builders [ea83, DH86]; and User Interface Management Systems (UIMSs) [FKKM91, vdMV92, MGD<sup>+</sup>90, Ols92].

All this research was focused on aspects of the realization of the interactive system [Thi90]. Even the UIMSs have a strong focus on the technical aspects of the system, even though they generally try to support some analysis and design activities. For instance, the User Interface Development Environment (UIDE) [FKKM91] contains an elaborate data model for modeling the technical part of the interface, and it contains a large number of modules to use this data to create different user interface elements such as context-sensitive help [dG92]. However, none of these modules make it easy to actually design an interface, nor do any of the modules contain any inherent knowledge about user-centered design. Even worse, the built in constraints can easily work against user centered design, even with a competent designer trying to create such a design.

**Analysis** After the technical difficulties of the current user interface paradigm were solved, or at least under control, attention in the HCI community started to slowly shift away from realization. People realized that building systems was not enough, but that they would also need to know *what* to build. This led to attention for analysis of the system to be designed and built. People started to describe techniques and methods to do this analysis in such a way that it would lead to usable interactive systems. The Human Factors field contributed the use of task analysis [Dia89, Joh92, KA92]. Task analysis had already been used for a long time by human factors people to describe tasks in operator oriented environments (such as power plants or satellite guidance and tracking) [Mit87, for example]. Unfortunately, task analysis is less helpful in analyzing ill-structured work.

Recently, the focus on analysis has been taken one step further. In addition to analyzing the tasks, all of the work context is taken into account. Examples of such techniques go by different names: contextual inquiry [WHK90], contextual design [GK91], activity theory [Kap92], and situated action [Suc87]. The main theme with all these approaches is that work is not just about some ab-

stract tasks, but rather about the whole of the work environment. This implies that the designer needs to take much more into account than just the tasks. The way in which the work is organized, the relative importance of each task and its relationship with other tasks, and the influences of other artifacts all need to be taken into account when designing an interactive system, according to those theories.

**Design** A thorough analysis of the work situation is useful, but it can not negate the fact that we still need to design a new interactive system. Design is an important part of the development process, because it is used to try to organize the chaos found during analysis into a meaningful order which can be implemented. Once a proper design is available it can be realized with tools and interface builders. Analysis is much needed during development, but cannot be used alone to guide the creation of an interactive system. Analysis only yields a flood of information, which in turn is canalized by design. Despite its central role in the development process, design has not been very well developed in HCI. The primary cause for this lack of attention appears to be a lack of involvement of design disciplines, such as industrial design, early in the evolution of HCI.

The need for design has been stressed enough, for instance in the pleas for user centered design encountered earlier, in analogy with longer standing fields such as architecture, and in discussions about the analogies with arts such as theater [Lau93]. Only recently more specific and widely applicable approaches to design have been published. A good example is ‘Bringing design to software’, which brings forward issues identical to the ones in this thesis [Win96]. Conference proceedings also show a marked rise in the number of articles on design through the years, and those conferences are integrating real-world examples of design and the underlying design process, the ‘design briefings’, in their conference programs [ADO94, KMM95]. Older examples of specific design techniques are Design Space Analysis (DSA) [MYBM91], and, to a lesser extent, critiquing systems [FLMM90].

#### 1.2.4 Consequences of the HCI evolution

The evolution of HCI has had its consequences, three of which are described in this section.

**Interface vs. interaction** One consequence is the confusion between ‘interface’ and ‘interaction’. Many people still believe that HCI concerns itself exclusively with the interface of an application, i.e., the windows, buttons, and graphics used to present the application. This view is a consequence of the early focus on realization, because this often means implementing the interface components of an interactive system, and because much of the research during the early HCI period focused exclusively on interface techniques and tools to help build interfaces.

Instead HCI is concerned with the whole of the application, not just the outside of it, simply because the whole of the application influences the use and usability, not just the interface at the outside of the application. All of the interaction needs to be designed properly; the windows on the screen are simply the last step in this process [VSG91, Har96].

This implies that the principles of HCI, such as user centered design, should be applied throughout the whole design cycle, from the earliest start of the design process. In this thesis I consider HCI to include all the aspects related to the design of interactive systems, and not just the interface of such a system.

**Disciplines in HCI** Given the wide range of aspects related to HCI, many different disciplines are needed to create good interactive systems, but a consequence of the HCI evolution is that the presence of different disciplines is skewed. There certainly are plenty of computer science people involved, but other disciplines such as graphical design and drama are not represented very well. Yet such disciplines could make meaningful contributions to the field of HCI [Lau93, for example]. A suitable analogy is film technology, which did not become a success until its use moved from the engineers to art-oriented people [Hec91].

Design is another minority discipline. Within HCI there has not been much emphasis on design, unlike in industrial design or architecture, until lately. One consequence of this is that only a few design methods are available. Many publications have presented some kind of design approach, but almost always these approaches are tied intimately to the particular problem they try to solve. This conclusion also answers the question why not much progress has been made in user centered design in the past years.

It is not just a matter of adding a bit more or less of one discipline or another. All of these disciplines need to work together to create a good HCI experience. With each of the disciplines having different values and interest, this is not something that comes easy or automatically. Explicitly dealing with multi-

disciplinary teams is crucial for good HCI.

**Integration of the development phases** The three main phases of the development of interactive systems, analysis, design, and realization, are described separately in the paragraphs above, not just because each of them was given more attention in turn as time progressed, but also because they are usually separated during development. Even though there is a logical, yet iterative, flow from analysis to design to realization, the developments in the three areas hardly ever allow an easy crossing from one to the other. As Fischer puts it:

“The conceptual structures underlying complex software systems are too complicated to be specified accurately in advance and too complex to be built flawlessly [...]. Specification and implementation have to coevolve [...] requiring a tighter integration of the frequently separated stages of software development: analysis, design, specification, and implementation. Thus, evolution occurs as feedback from partial solutions improves the developers’ understanding of the problem.” [FRW<sup>+</sup>95]

As Fischer indicates, the connections between the three stages of design have not received much attention. Most research is concerned with a single aspect of the development of interactive systems, and does not try to bridge the gaps to other stages. This has resulted in a large number of independent solutions to particular aspects instead of more broadly oriented approaches.

### 1.3 Research questions

The following objective for this research was formulated based on the observations outlined in the previous two sections:

to explore how explicit attention to work context early in the design process can be used to improve the usability of interactive systems facilitating ill-structured work.

This objective is based on the following observations:

- the productivity paradox seems to apply in particular to systems facilitating ill-structured work;

- problems with interactive systems facilitating ill-structured work are particularly hard to diagnose and fix;
- design is the least supported part of the development process;
- most currently available support for design is aimed at the later stages of design, i.e., interface design, instead of the initial stages of design, i.e., design of the structure and interaction of the system.

To meet the research objective, ill-structured work and its context needs to get a central place in design. Only then can usable interactive systems be designed, because it is the design activity which determines structure and interaction, and it is this structure and interaction which in turn determines the usability of the interactive system in relation to the work. The question is how this can be accomplished. What kind of design activities are needed to meet the research objective?

**Research question 1** *How can we formulate the design activity for interactive systems which facilitate ill-structured work?*

The conclusions from the evolution of HCI presented above already indicate that these design activities should not initially include design of the actual user interface. Structure and interaction of the system are, at least initially, more important. To emphasize this the second research question is formulated to find a suitable description which does not include user interface components:

**Research question 2** *How can ill-structured work be described explicitly during design without reverting to interface components?*

Finally, it is important to ensure that the results from the first two research questions provide a usable and workable solution fitting the research objective. Usable, in that the design activities and design representation can be used by a design team; workable, in that the activities and representation fit well into the development process; and fitting, in that the activities and representation yield the desired results:

**Research question 3** *How can it be ensured that the formulated design activities and representation of ill-structured work provide a usable, workable, and fitting solution towards the design of interactive systems facilitating ill-structured work?*

Underlying the research objective and the research questions is a paradigm that is not explicitly tested in this thesis. The research objective and the research questions have been created with this paradigm in mind, though, and this paradigm also strongly influences the remainder of the thesis, in particular the case analysis and theory formulation. In Section 7.2.3 the paradigm will be revisited in light of the research presented in this thesis. The paradigm is:

*An interactive system should be designed as a whole; the leading perspective during design should be that of its users.*

This paradigm states that it is the future user of a system who needs to be satisfied to the full, and that the design therefore has to start there. Issues related to the realization of the system can be valid constraints but should not determine the direction of the design. Too much is designed from a technology perspective as it is [Tha01]. In addition, it makes clear that this design should be seen as one piece of work. It can not be taken apart initially. This is in contrast with the idea that the user interface can be designed as some sort of add-on separately from the remainder of the system as mentioned on page 11.

## 1.4 Research approach

A research philosophy, an accompanying research strategy, and research instruments to carry out the research are described in this section.

### 1.4.1 Research philosophy

For this research two major research philosophies are distinguished, in analogy with research into organizations [vM94, dV95, for example] and in social research [Hug90].

**Positivism** Positivists believe in the objective nature of research. They assume all things can be approached objectively, and hence try to describe the phenomena being researched in an objective manner. To accomplish this, positivists require experiments to be repeatable, and stress isolation of the independent phenomena within the research object. This attitude is reflected in their choice of research instruments such as experiments, surveys, and field experiments.

**Interpretivism** Interpretivists believe in the subjective interpretation of the researcher. Based on the idea that the researcher always has some influence on the research object, they mostly rule out objective research. Instead they propose conducting the research in a natural environment, and interpret the results. Again, this attitude towards research is reflected in the research instruments, such as action research, field studies, and reviews.

Most of the people who write about research approaches in HCI do so from a strong positivist perspective. Kirakowski and Corbett, for instance, have written a book about research methodology in HCI [KC90]. In their book they focus solely on doing experiments and user studies, on making aspects of the user interface or the interaction measurable, and on doing this in a statistically sound way. In similar methodology oriented articles a strong psychology background seeps through [Lan88, McG95, for example].

In contrast, not much has been written about an interpretivist approach to doing HCI research. No articles or books available at the start of this research on research methodology for HCI mentioned an interpretivist approach to HCI research. A likely explanation is the strong influence of the positivist research tradition of psychology, human factors, and to a lesser extent computer science.

During the course of this research support for an interpretative approach has been voiced within the HCI community. Carroll and Kellogg argue that much of the theory behind HCI can be found by studying and observing the artifacts which are designed and used [CK89]. Additional arguments can be found in the social sciences where a similar choice between positivist and interpretative approaches is leading increasingly to interpretative approaches [Hug90].

An interpretative approach seems to be particularly suited for research surrounding the design stage of the development of interactive systems. Quantifying phenomena during the design stage is almost impossible, because most of these phenomena can not be quantified, e.g. ease of use of a design, or esthetics. Creating experiments which are repeatable can be done [NF, for example], but it is very hard to define and measure individual variables, and it remains to be seen what can be generalized from these experiments. Even repeatability is often a problem, since human designers are directly involved with the experiments. Finally, with the small amount of theory available in this area of HCI, not much can be tested right now, and instead it seems theory has to be developed based on practice in the field. These arguments lead me to choose an interpretivist approach to the research presented in this thesis.



### 1.4.2 Research strategy

Several research strategies are available to carry out interpretivist research. The selection of a proper strategy is guided by the nature of the problem, and by the status of theory development in the research field. The status of theory development in the field of HCI, in particular in the design of interactive systems, is still very much in a very early stage of theory development (See Section 1.2).

The research presented in this thesis represents an ill-structured problem [Sol82]. This essentially means that the problem of improving the design of interactive systems supporting creative work can not be solved in a purely deductive manner. In addition, to my knowledge no readily available theory can be used as a guide to solve this problem. This implies that the research problem can only be solved by inductively studying the design process, and developing a supporting theory based on these observations.

Both De Vreede and Van Meel show that the most appropriate research strategy for this type of approach is an adjusted version of the inductive hypothetical research strategy [vM94, dV95]. According to Sol, this approach has the following advantages [Sol82]:

- it stresses the inductive specification, testing and expanding of theory;
- it offers possibilities for an interdisciplinary approach;
- it enables the generation of various alternatives for the solution of the problem;
- it emphasizes learning by considering analysis and synthesis as interdependent activities.

These advantages are directly applicable to the research problem presented in this thesis. The only way to formulate some theory, given the lack of available theory in the research field, is to observe what currently happens in practice, reflect on this using available literature, and use this combined knowledge to form a theory. This can be facilitated by an inductive approach. Developing interactive systems is inherently an interdisciplinary approach. A useful theory can not be created in one step because of the complex nature of the development of interactive systems. Rather, several alternatives need to be explored, and a proper way to improve design needs to be found through iteration. Finally, we can learn iteratively from applying the theory and using the feedback to improve the theory.

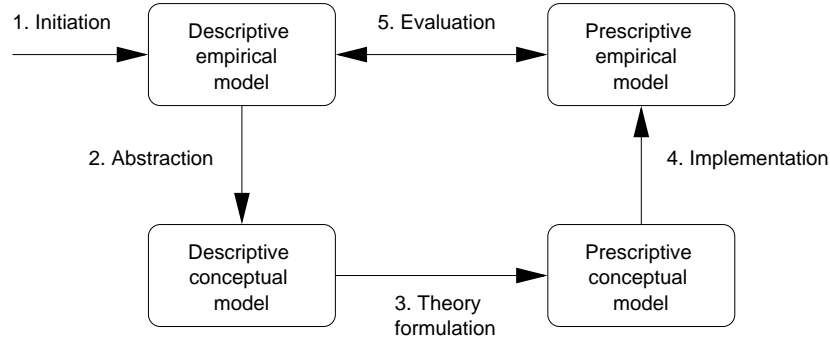


Figure 1.1: Research activities for an inductive-hypothetic research strategy

The inductive hypothetic research strategy consists of five activities and four models, as shown in Figure 1.1 and described below [dV95, Sol82].

1. A number of initial theories are identified, regardless their completion or verification. These theories are used to investigate empirical situations, leading to one or more empirical models.
2. Empirical situations are abstracted into a descriptive empirical model. The emphasis during abstraction is on all essential aspects, but more importantly the perceived problems.
3. A prescriptive conceptual model is derived from the descriptive conceptual model. This prescriptive model takes the form of a theory, and it should be capable of solving the problems observed during the first step.
4. The prescriptive conceptual model is then implemented in one or more prescriptive empirical models, to test and validate the prescriptive conceptual model.
5. The results of the prescriptive empirical situation are evaluated. Additional improvements or changes to the prescriptive conceptual model may be identified.

### 1.4.3 Research instrument

Research instruments are needed to carry out the steps of the research strategy. Action research is the instrument that fits this research best.

**Action research** Action research is similar in some respects to case study research. Case study research aims to provide the actual examination of a contemporary phenomenon in its real life context [Yin89, Che81]; its focus is on the observation. Action research aims to have a dual outcome [APS90, ALMN99]. Using action research results in *action*, changes made to a system, and *research*, an understanding of the changes made. This dual character affords the simultaneous application of a framework, and an understanding of the way in which the framework supports the design process [Dic93].

This is very similar to what Collins described as ‘participant comprehension’ [Col84]. With this term Collins means that the researcher should actively engage in the topic under research, so that all of the forces and actions can be experienced first-hand. This can lead to a much better understanding of the topic from the inside out, while with observation only the effects of this can be seen, but not the motivations underlying them.

**Why action research is suited for this thesis** The research questions posed in Section 1.3 all deal with the act of designing. This is an inherently hard activity to observe, because so much goes on inside the designers head. Verbalizing all the different trade-offs, constraints, and alternative possibilities is hard to do. Experiencing them first-hand and then reflecting on them will provide much better results.

## 1.5 Thesis outline

This section contains an introduction of the remainder of this thesis, and explains how the research activities introduced in Figure 1.1 and Section 1.4 can be found in this thesis.

First, Chapter 2 describes the design process of an interactive system to accommodate complex numerical analysis of civil engineering constructions: DIANA. This experience is summarized in Section 2.5, and this summary can be seen as the *descriptive empirical model* as it describes the design experience gained.

Next, in Chapter 3, this one experience is abstracted. This is done by examining relevant literature, and reflecting this back onto the conclusions from the DIANA-case. In addition, some of the issues introduced in Section 1.2 are explored further. As a result, Section 3.5 draws conclusions and lists requirements for a design tool. This section can be seen as the *descriptive conceptual model*, because it brings together real-world experience and existing theory to formulate what problems remain which must be addressed to answer the research questions.

Chapter 4 contains a formulation of the theory called WONDER which tries to address the conclusions and requirements presented in the previous chapter. Thus it can be seen as the *prescriptive conceptual model*; it prescribes how the design process should be carried out, and what design tools need to be used. To facilitate evaluation a number of assumptions are presented in Section 4.8.2. Chapter 5 describes a research instrument created to allow WONDER to be used in a real-world situation.

Chapter 6 describes the use of WONDER in the design of a shipyard planning system called SHIPSHAPE. The use of WONDER on an actual problem can be seen as the *prescriptive empirical model*, in that the prescriptive conceptual model is now tried in an empirical situation. This can then be evaluated, leading in turn to another *descriptive empirical model* which can be used as input for another research cycle. The evaluation is described in Section 6.5. Starting points for other research cycles are described in Chapter 7, in addition to a complete overview of the research presented in this thesis.

## CHAPTER 2

---

### DIANA— a case study

---

In this case study the design of an interactive system in a real world situation is investigated. The goals of this case study were to apply current techniques and methods for analysis and design of interactive systems, to observe what kind of problems surface while doing this, and to analyze the causes and generality of these problems. As such, this case study provided an approach to understanding the practical side of designing interactive systems. The chapter has a strong focus on the process carried out, and much less on the product being designed. Further background information on DIANA and its architecture can be found in Appendix A.

### 2.1 Case study description

The case study was carried out at TNO Building and Construction Research as part of the I-DIANA project. The goal of this project is to design a graphical user interface for DIANA, a large software system for finite element analysis. At the start of the project DIANA existed as a batch-oriented system.

### 2.1.1 TNO Building and Construction Research

TNO<sup>1</sup>, i.e. the Netherlands Organization for Applied Scientific Research, is a large Dutch research and technology organization. It is a commercial organization which tries to apply technological knowledge with the aim of strengthening the innovative power of industry and government. Approximately 25% of turnover is currently generated internationally. [TNO94, TNO00].

TNO consists of fourteen institutes. One of these institutes is TNO Building and Construction Research. This institute has several departments, including the department of engineering mechanics and information technology. One of the main activities of this department is the development and enhancement of the DIANA finite element analysis (FEA) system [Dia00].

DIANA is a general purpose system for analyzing the behavior of various types of structures and materials. Use of DIANA allows detailed calculations to be made of the behavior of complex structures under both static and dynamic conditions. The importance of having reliable simulation models has increased significantly with the need to use materials in a cost-effective manner and the growth in the complexity of buildings and structures. Development of DIANA started as early as 1972 as a vehicle for research in numerical algorithms applied to finite element analysis. From the mid-eighties onwards it has also been applied to real-world engineering problems. Today, these engineering problems constitute the majority of DIANA's use. A recent example of its use was for Rotterdam's Erasmus bridge, where DIANA was used to calculate the structure itself as well as specific smaller details [Dia92].

### 2.1.2 Finite element analysis

DIANA's application domain is very broad, consisting of all areas where finite element analysis can be used. New algorithms and numerical methods are continually added through strategic alliances with universities and other computational mechanics research centers. Such a diverse application domain implies a broad spectrum of users. Indeed, users are found in such diverse areas as universities, construction and engineering companies, a nuclear power company, biology, and municipal organizations.

Finite element analysis can be applied to a wide variety of problems, but the underlying principles remain the same. With finite element analysis, a skeletal structure is assumed to be made up of a collection of two and three

---

<sup>1</sup>TNO is the abbreviation for 'Toegepast Natuurwetenschappelijk Onderzoek'



Figure 2.1: Erasmus bridge, Rotterdam

dimensional elements, connected together at discrete joints. In this way a continuum with infinite degrees of freedom can be made discrete, and turned into an equivalent system with finite degrees of freedom [CY79]. Finite Element Methods (FEMs) can then be used to carry out numerical calculations on this equivalent, simplified, system to study the actual structure.

Working with FEMs requires a number of steps to be followed. Here only a brief overview is given. A more detailed example is given in Section 2.3.3. More specific information on how these steps are done for DIANA can be found in Appendix A. First, a model of the object under study, describing the geometry, needs to be created. This can be done in a variety of ways, e.g. by hand, using modelling software, or by importing the model from another application such as a CAD system. During the second step, this model is meshed, i.e., it is divided into small elements. The computation of the actual mesh is done by DIANA, but the user needs to provide DIANA with information on how to divide the model into a mesh. Again, this can be done by hand or by using modelling software. The amount of elements depends mainly on the numerical algorithms used, and on the required accuracy of the results. A finer mesh will yield better results, but will also cost more computing time. For larger models the difference in computing time can be significant. Computations taking up to three weeks, using computers which were state-of-the-art in 1992,

are not exceptional. As the machines get more powerfull, the models get more complex, often keeping computational time constant.

In the third step additional characteristics of the model are defined. This is done by hand when using DIANA. The characteristics are added to the file describing the model and mesh. Materials, including several of their characteristics, are assigned to the elements of the mesh. Additional constraints such as loads are placed on the model, and boundary conditions are added. An example mesh with boundary conditions and loads is shown in Figure 2.2. The figure shows a square sheet of material, anchored at the left side, e.g. cemented into a wall, and loaded at the right side, e.g. by placing weights at the far right end of the sheet. The mesh is created by dividing the model of the sheet into sixteen squares.

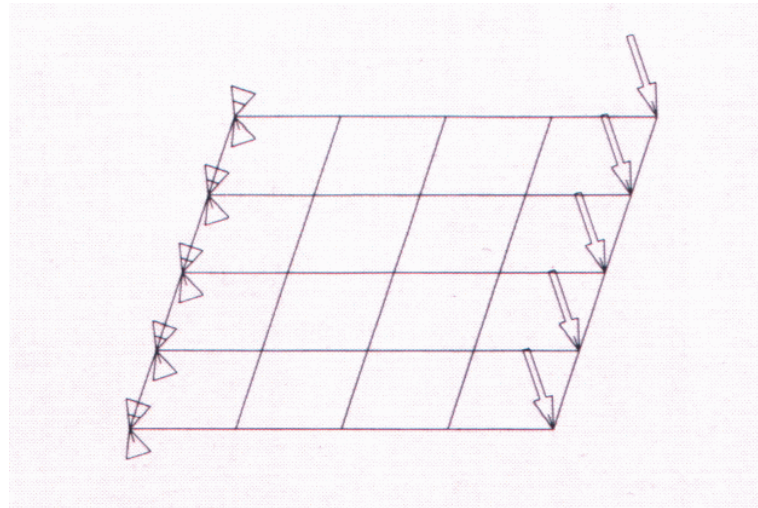


Figure 2.2: Example mesh of model with loads and constraints

Then, one or more calculations are carried out. These calculations are specified by hand. Often several calculations are defined at the same time, so that the often lengthy calculations can run without further interruptions. For instance, first the loads are calculated, yielding a distribution of loads through the material. This in turn could affect small cracks defined in the model, and



a separate calculation can be carried out to calculate the effects of the load on the cracks.

Finally, during the last step, the results of the calculations can be visualized and analyzed in a number of ways. This is normally done by manually operating a separate application which can visualize the results from the calculations in several ways. This analysis can trigger an iterative process where one or more steps are refined further.

### 2.1.3 The I-DIANA project

The I-DIANA project was a large ongoing project at TNO Building and Construction Research. The project goal was to create a graphical and interactive user interface for DIANA. At the start of the project DIANA only had a batch-oriented interface, and it was decided that a graphical and interactive interface should be developed in addition to the batch mechanism.

**Project motivation** The main motivation for starting I-DIANA was market pressure. DIANA is not the only software system for complex FEA calculations, and several of DIANA's competitors already have graphical user interfaces. While DIANA still held a competitive edge with superior algorithms and calculations, it was slowly losing market share.

TNO also wanted to increase its market share by selling DIANA to new user populations, and the people in these new markets expected graphical user interfaces. In DIANA's traditional market, mostly research institutes and universities, users had been more interested in the numerical results than in its interface. People in other markets, e.g. small engineering firms or civil engineering departments of larger companies, are less interested in all the details of the specific algorithms implementing their calculations. They simply want to be able to do their job without any hassle, and without learning a complex and arcane syntax.

Finally, some current users of DIANA, triggered by the advent of other graphical interfaces for their desktop, were starting to ask for a better, more user-friendly, user interface for DIANA.

**Project environment** The I-DIANA project was only one of several projects aimed at further developing DIANA although all other projects were directed at development of new or improved numerical analysis and algorithms.

The project team for I-DIANA consisted of people already working on DIANA for TNO, with the exception of the author. As each of these people had particular skills with respect to DIANA e.g. writing manuals, system engineering, numerical algorithms, each of them participated in several projects at any one time, because their expertise would be needed in each of these projects. As a result, there was continuous tension and competition for resources between different projects; and priorities between projects would sometimes be different for individual team members. In total, six people were involved in the I-DIANA project, but each of them was also involved in one or more other projects.

#### **2.1.4 Relation between the case study and the I-DIANA project**

Only part of the I-DIANA project was investigated for the case study. When the case study started, the I-DIANA project had already begun. A core project team had been formed and a first inventory of possibilities, goals, and problems had been carried out.

Only part of the I-DIANA project was examined for two reasons. First, it was estimated that the project would take a significant amount of time, at least several years. The time available for the case study, however, was limited. Second, not all parts of the project were deemed equally interesting with respect to the problems noted in the previous chapter: creating the design of the interactive system was the most interesting part of the project for this research. Analysis was also included in the case study, both because it precedes design, and because the results of the analysis process provide the input for the design process.

Realization of the design, apart from some small prototypes as part of the design process, testing, usability studies and implementation of the new software in a user environment were specifically omitted.

The author participated actively in this case study as the user interface expert, carrying out user analysis, and leading the design of the user interface. A participative role was chosen to gain a better understanding of carrying out the design process for a complex interactive system. Observing an experienced designer during a similar design process would not have yielded similar experience.

## 2.2 I-DIANA Project approach

The I-DIANA project approach followed standard development steps: analysis, design, realization, testing, and implementation. While these steps follow from each other, they are not strictly separated and iteration between the different steps is encouraged.

To make the design process more manageable, and to check continually the software constraints of the project, one small, but representative, part of DIANA was iteratively designed and implemented as a prototype. This design was later extrapolated to all other parts of DIANA.

### 2.2.1 I-DIANA constraints

The major constraint for the I-DIANA project was a lack of manpower, which inhibited a total rewrite of the software. This constraint influenced the approach taken as well as the outcome of the project. Several requirements resulted from this constraint:

- maintain the current batch interface in the new system
- keep as much existing code as possible, major rewrites were not possible
- strive for gradual implementation so that some results can be delivered quickly

As a consequence of these requirements, the design phase of the project had to be closely coupled with part of the realization phase because a new user interface had to work with the existing system. It also had consequences for the analysis phase, for instance because no requirements were required to define the underlying functionality, which was already present in DIANA.

Additional constraints were mostly related to less important details such as the platforms on which the product should be available, and the development methods and tools used for realization of the software and prototypes.

### 2.2.2 Analysis approach

Several textbooks on HCI give overviews of an approach for the analysis phase [DFAB93, HH93, PRS<sup>+</sup>94]. All of them mention the use of task analysis during this phase. In addition to this, each of the books takes a somewhat different stance on analysis. Dix et al. emphasize usability and models of the user, but

provide no additional guidance for carrying out analysis. Preece et al. detail several methods for user-centered design, and talk about requirements gathering at some length. Hix and Hartson list a number of different ‘early analysis activities’, and show their relationships and dependencies. They also stress the importance of ‘contextual inquiry’, and of having a domain expert on the development team.

In summary, the textbooks provided much generic advice, e.g., involve users in the design process; get to know the users, but little in the way of concrete methods to work with, other than task analysis methods. Based on this advice the project team decided on three major goals for the analysis phase:

1. to involve users in the design process
2. to get to know the users
3. to apply task analysis

**Involving users in design** Almost all members of the development team had experience in using FEA. They were not, however, typical users, as they had more experience with developing numerical algorithms, or applying DIANA towards very specific development oriented problems.

No ‘real’ users were included in the development team. Finding representative users of DIANA is hard because DIANA is used in so many different application domains: no single user or small set of users can be representative. The only common factor between all domains is creating the input files, and running DIANA on them; and all the members of the development team did have experience with this.

We did involve users in the design by keeping them informed through user group meetings. At these meetings, we also solicited feedback on our design proposals.

**Getting to know the users** Getting to know DIANA’s users is a complicated job, for two reasons. First, DIANA is used internationally, which means that going out and talking to users is not possible in all cases. We were only able to contact users in the Netherlands directly. For this reason we started the analysis by sending out a questionnaire to DIANA users, hoping that this would at least provide us with some user profiles, a general sense of how DIANA is used, and expectations of and suggestions for the I-DIANA project.

A questionnaire can not be regarded as a substitute for contextual inquiry. We decided to use interviews with selected users to get information about work context. Observation would not be a practical approach, because, in general, real-world problems with DIANA take a long time to complete, up to several weeks. In addition, we would only observe the user *using* DIANA and possibly miss opportunities to get a feel for the real underlying problems and issues.

The interview design put a strong emphasis on the daily work of the users, and the role of DIANA in their work. A major part of the interviews was spent on talking through the life cycle of one of the current projects of the user. In a sense this provided us with a condensed version of an observation.

**Task analysis** Since all textbooks recommended task analysis as a more formal method for gathering information on the work of the user, this possibility was also investigated. No specific method came highly recommended, so the task analysis method described by Bots [Bot89], which was already familiar to some members of the team, was selected.

Task analysis will help to provide an overview of the tasks of the users in an organized and structured way. Depending on the tasks and the system to be designed, task analysis can either provide direct input towards the system architecture, e.g. by suggesting menu layout or order of input screens. Otherwise, task analysis will at least provide significant insight in what the users try to accomplish with the system.

The main questions with respect to task analysis in this case were, first, to what level of detail the tasks could be broken down, and second, whether tasks could be considered across the sometimes very different domains of DIANA's users.

### 2.2.3 Design approach

Standard textbooks on HCI do not give much support for a design approach [DFAB93, HH93, PRS<sup>+</sup>94]. They move quickly on to more technical approaches to design, such as representing interaction of the user with the interface and use of metrics and guidelines. Real attention to the design process and to different approaches to designing interactive systems is not easily found, or only superficially described. Several techniques that are described include conceptual design, scenario based design, design rationales, and prototyping. Most of these techniques will be discussed in more detail in Chapter 3.

**Conceptual design.** Conceptual design is not as much a particular technique but more a very general approach. The idea is first to create a design in which the actions and objects needed in the systems are identified and placed in a logical framework so that they will support the user. While this is an important step, no specific methods are identified. Examples in the textbooks remain vague and informal. It is clear that this is an important step to make, as it influences the overall impression of the system.

**Scenario-based design.** Using scenarios to influence design decisions is gaining momentum. The idea is to create more detailed parts of the interface of a system based on possible scenarios which describe part of the work of the user. These ideas will be constrained and shaped by the conceptual design made previously. The main advantage of using scenarios is that it quickly becomes clear whether a particular approach in the system can support the work of the user in a sensible way. By thinking about what the users need to do with the system, it may become easier to influence the design. Again, no formal techniques exist for scenario-based design.

**Design rationales.** The use of design rationales allows the designer to expand the design space, to explore other alternatives to particular design problems. Essentially, the designer lists design problems, possible options to solve those problems, and the criteria which the solution should satisfy. The use of design rationales supports a structured way to explore design alternatives.

**Prototyping.** Use of prototyping can help to assess a given design, both in terms of technical feasibility, and in terms of usability. It is often used to support testing scenarios, or to run general usability tests on the user interface in an early stage.

**Approach for I-DIANA** Given the lack of formal methods and techniques which could be applied, a formal approach for design in the I-DIANA project was not written down, however both conceptual design and prototyping were seen as important steps in the design process.

A conceptual design was deemed very important, because such a large system, dealing with so much information, would have to be presented in a uniform way. The added complications of a very diverse user population made it clear that good concepts would be very important. Several other finite element analysis systems were studied to see what kind of concepts were used to present the common objects and actions.

Prototyping was deemed important not only to get user feedback, but also to assess the feasibility of the technical issues of the design. Adding an interactive user interface while also keeping the old system operational provided a major technical challenge. Creating prototypes of proposed technical solutions could help solve this problem.

The size of the project made it impossible to prototype all of it. Therefore only one DIANA module was selected for prototyping. This selection was based on current experience of the development team, and because the module provided a good representation of the different ways input and algorithms are supported in DIANA.

The design process for I-DIANA had to be iterative, because there were still many blank spots in the information collected during analysis when the first design activities started, and because of the many technical constraints, that required the exploration of possible solutions.

## 2.3 Analysis of current and future use of DIANA

During the analysis phase of the project, three major activities were undertaken: sending a questionnaire, conducting a number of interviews, and analyzing tasks found during those interviews. In this section the results of these activities are discussed briefly with an emphasis on the process of doing analysis instead of the products of the analysis. The results of the questionnaire and interviews are described in more detail in TNO reports [dG93, dG94].

### 2.3.1 The Questionnaire: a first profile of DIANA users

A questionnaire is not a very good way to get to know the user, as we came to understand when we were drafting it. Multiple choice answers are often preferred, because they are much easier to answer. Open questions have the potential to be answered in a more complete, or unexpected, way, but often this does not happen either, as it takes more of the respondent's time and effort. Furthermore, analyzing ones own work is very hard to do, because it requires a fresh and neutral look at all activities associated with the work. Therefore a questionnaire is not the best means to find out anything meaningful about the work of the users. For this reason the questionnaire was targeted mainly as an inventory of DIANA use. The diversity of DIANA users was already known, but the extent and variation of this diversity was not known and the unraveling of this diversity was the main target of the questionnaire.

Additionally problems with the current way of working with DIANA were also solicited, as were expectations and wishes regarding an interactive graphical interface. Not much feedback was expected on these latter items, they were mostly added for completeness, and to allow people to express their thoughts about these matters.

**Respondents** The questionnaire was sent out to all current users of DIANA through the user contact group, for a total of sixty organizations. We received 15 responses.

What was most standing out was the variance between the respondents. Some had up to 9 years of experience with DIANA, others less than a year. Some used DIANA daily, others at most once a month. The type and size of calculations carried out with DIANA also varied quite a bit.

**DIANA user population** The answers to the questionnaire provided a good overview of the different types of organizations and application domains in which DIANA is used. Four main categories of DIANA use can be distinguished:

**Commercial production computations.** Here DIANA is used for production-oriented computations, e.g., whether a bridge is strong enough to sustain traffic and sundry weather conditions, or whether a building will collapse given the materials used. Computations are often small and detail oriented, and similar in structure. DIANA is used occasionally, and access to pre and post processors is limited. Users have experience with FEA.

**Commercial research.** Here DIANA is used for very specific and complex problems, often requiring specific domain expertise, e.g., new construction methods or materials, or new types of structures. Both models and computations are very large and one of a kind, and use of DIANA can be extensive. Users have a good understanding of FEA, and additional pre and post processes are available.

**Educational organizations.** Several universities use DIANA to teach the practical aspects of finite element analysis. Their use of DIANA is hard to characterize. The instructors have considerable experience with DIANA, while students do not. Models and computations are usually small, but they vary quite a bit between users.



**Research into numerical algorithms.** The DIANA framework is also used to work on new algorithms and numerical methods. Models and computations are small, and specifically created to test the algorithms. Users have considerable experience with DIANA and FEA, and other software is rarely used.

These four categories present only a very broad subdivision of DIANA's users and they do not take into account the diverse application domains described earlier.

**Comments on current use of DIANA** Against expectations many respondents gave good feedback on the open questions regarding current use of DIANA. Only experienced users had few problems with DIANA, but they often admitted to having problems once they had to use a module of DIANA which they normally did not use.

Most problems, however, were encountered by novice users. Some of these problems were related to internal DIANA issues. The fact that these issues surfaced through the user interface is due to the structure of DIANA, which is explained in more detail in Appendix A. Two main themes ran through the complaints:

**Making mistakes in input is expensive.** Most of the complaints were related to the handling of mistakes in the input files. The batch-oriented nature of DIANA makes it impossible to detect errors in the input until that part of the input is actually processed. This could take several hours if the error resides in parameters for a computation; DIANA stops immediately after finding an error. Fixing several small syntactical errors in an input file quickly becomes tedious and time consuming.

**Too many options to choose from.** DIANA provides a flexible environment for finite element analysis, but as a consequence, there are many options to choose from at any one point. Obviously the users do not complain about having these options available, but each user only needs a fraction of these options for their application domain. In addition, the consequences of each option are often not very well explained. This can in part be blamed on the manuals, but even the manuals cannot cope with all the possible combinations of options. Currently users deal with this situation by re-using previous input files, and by trial and error. The latter in particular, can be very time consuming and inefficient.

**Questions regarding a new graphical interactive interface** The answers to these questions were the least interesting. Most people responded positively towards a graphical user interface for DIANA, expecting it would make working with DIANA more pleasant, more direct, and more interactive, and therefore easier for novice users, and that it would allow easier analysis of computational results.

Some concerns were also voiced, for example about a possible rise in the cost of using DIANA, both for the software and because of the additional hardware requirements. Some people concluded that the batch-oriented interface might disappear, and protested against it either because they had custom tools built to generate or modify input files, or because they were used to the current batch-input system. Finally, some people expressed doubts whether a graphical user interface would be more efficient, in particular for expert users who were used to the cryptic but quick keystrokes.

The questionnaire also asked which elements of graphical user interfaces should be included in DIANA, but no meaningful suggestions were made. The suggestions all listed existing user interface elements, and were often in contradiction with each other, e.g., one person might like to see many separate windows, while another wanted just one window to work with.

**In summary** The purpose of the questionnaire was to get an overview of the user population of DIANA, and to be able to categorize them. With an approximately 25% response rate and good representation across application domains, the answers yielded enough information to be able to do this.

Although little response was expected to the questions regarding current use, users felt strongly about this issue and provided many problems with and observations on current use. This provided the development team with a strong indication of the problems in the current batch-oriented implementation of DIANA.

Finally, the questions regarding the new graphical user interface did not provide useful answers. Partly, this is because the questionnaire did not include any suggestions or hints on how the development team envisioned the user interface, and partly because users found it hard to visualize a user interface which would enable them to do their work, as some of them indicated in their responses.

### 2.3.2 The interviews: fleshing out DIANA's usage

The initial purpose of the interviews was to get a good sense of the work being done with DIANA, and insight in how this work could be supported through a graphical user interface. Users from each of the four main categories were selected, based on their responses to the questionnaire and their geographical accessibility. A total of six interviews were carried out.

**Structure of the interview** The interviews were conducted according to a fixed structure in three sections to ensure that the information would be useful. [tH88] The first section of the interview asked follow-up questions about the questionnaire, either to clarify previous responses or to solicit additional comments.

In the second, largest, section of the interview, the user was invited to discuss a recent project done using DIANA. Users were asked to bring relevant material when they came to TNO to be interviewed. When they were interviewed at their own workplace there would be enough information on recent projects around. The life span of the project was then discussed, including those parts not currently supported by DIANA. A discussion of a recent project carried out by the user provided a natural way to discuss all aspects of the use of DIANA, from preparing input files to analysis of the computational results; and this part of the interviews provided us with the work context of DIANA users, in particular because the discussions were augmented with real artifact-sartifact.

Some paper prototypes of a potential interface for DIANA were shown during the third section of the interview. The prototypes were very rough, with many details still missing. They resulted from the initial conceptual design described in Section 2.4. Again, the user's project was used as a vehicle to identify in which way DIANA's interface might support parts of the process. Where needed additional details were filled in by hand to facilitate discussion. With these prototypes a step by step walkthrough was not possible because of the level of complexity involved, making it hard to simulate the interface by hand.

**In summary** The main goal of the interviews, to establish the work context of users in the different user categories, was met. The main lesson learned about the work context of DIANA was that there were large similarities between users of the different categories. These similarities were caused by the

underlying similarities of the finite element methods, which prescribe a strict order of working through a problem.

The problems with current use as noted above were further re-enforced by the interviews. In fact, it was striking to see how many users re-used their previous work in one way or another, often because this would, in the end, be much quicker than to create input from scratch with all its associated problems like not knowing the correct options or arguments for an option.

In addition the interviews resulted in detailed descriptions of the tasks users carry out with DIANA. Although not initially planned that way, this part of the interviews provided valuable input for the task analysis described in the next section.

### 2.3.3 Task analysis: a formal analysis of DIANA's usage

All three textbooks used as references contained sections on task analysis, and suggested its use either during analysis, or as a means to structure information during design. A task analysis was made of the work discussed in several of the interviews. To describe and analyze the work presented in the interviews, 'task structures' and 'decision structures' [Bot89] were used. In a task structure tasks are represented by boxes, and decisions by circles. In a decision structure, decisions are again represented by circles, and information is represented by boxes.

**Example task analysis** Consider the task structure presented in Figure 2.3, which describes the construction of storage racks. The objective of this task is to design such a construction as cheaply as possible, while still taking several constraints such as strength into account. The price of the construction can be varied by the material and profiles being used; details of the construction such as reinforcements can also be varied.

The task structure shows the different tasks and branching points in this job. The whole task is highly iterative. Several loops are used to try and improve the different attributes of the problem. For instance, the small loop for visualization is used to adjust the visualization parameters until the current solution is properly visualized.

Several of these tasks could be detailed further, for instance 'make computations'. Detailing these tasks, however, is not very interesting because the details are obvious; further detailing them with smaller tasks will not add significant insight into the work being done there. It would be much more interesting to

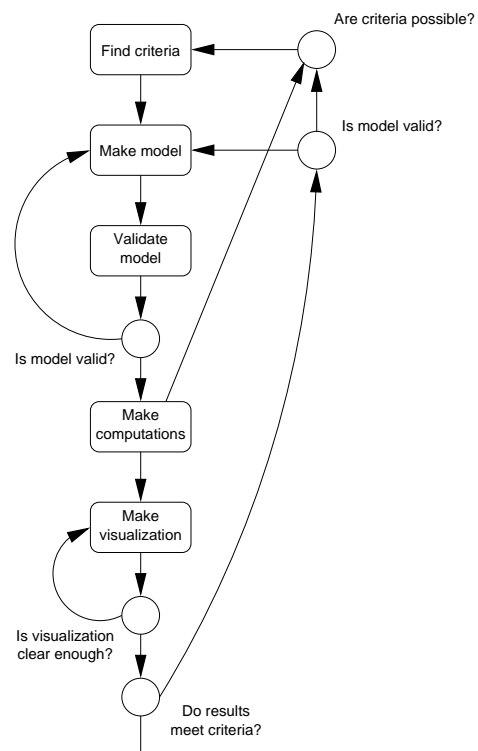


Figure 2.3: Task structure: construction of storage racks

further detail a task such as ‘validate model’. The constituent parts of this task are not straightforward, and detailing the smaller tasks might help to better understand how a user carries out this task. Unfortunately, further detailing of this task proved to be impossible to do. The matter is that there is not one single approach or recipe for validating the model. During the interviews we noted that different users would often use different approaches, and, more importantly, these approaches would depend on the problems under study, and also vary with a single user.

The individual tasks could have been recorded and presented as a task structure in each case. This would not have provided a starting point for design, though, because it would have resulted in a number of individual *descriptive* models instead of a single *prescriptive* model that would be suited for design. A prescriptive representation could be created, for instance by taking the most common details, or by rationalizing the best approach. Users have more experience of validating models, though, and it is very likely that this single task structure would inhibit their work instead of augmenting it.

Bots proposes using ‘decision structures’ for such tasks which can not be detailed further. Decision structures contain the individual decisions which need to be made as part of the task, and the information needed for and resulting from each decision. An example of the decision structure ‘validate model’ is shown in Figure 2.4.

**Analysis of the structures** An analysis of the task and decision structures shows that iteration is an important aspect of the problem solving process associated with making FEA calculations. A supporting tool should allow for ‘what if’ analysis, and provide rapid feedback on changes.

**Use of Task Analysis** All three textbooks recommended task analysis in one form or another. In this case task analysis proved to be useful to gain insight in the work of DIANA’s users in a structured way. The technique’s main value turned out to be its descriptive power. In contrast with what the textbooks seem to imply, not all activities could be modelled in a meaningful and detailed enough way with task analysis. It did yield a good visual description of the iterative and exploratory nature of the tasks, which can serve as a starting point for the conceptual design which is described in the next section.

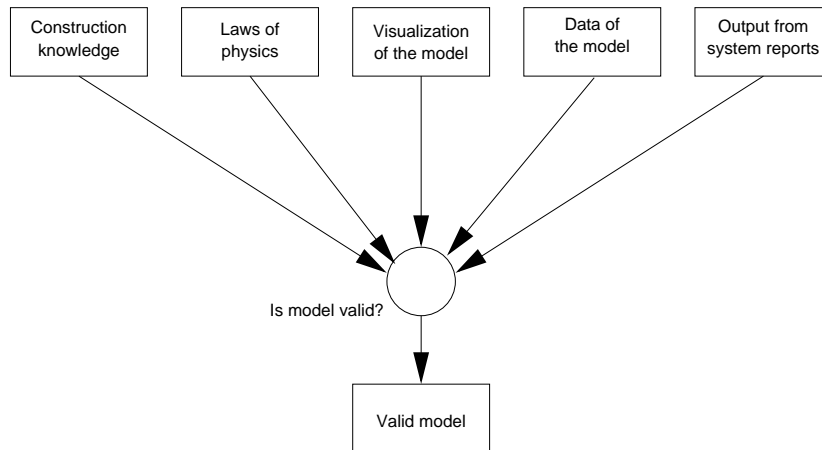


Figure 2.4: Decision structure: validate model

## 2.4 Design of a graphical user interface for DIANA

The *conceptual* design work done for the I-DIANA project is described in this section. Other design work was done for the project, such as more detailed screen layouts and additional work on menu structure, etc. This design work falls outside the scope of the case study because they are primarily aimed at implementing the system, and is therefore omitted.

### 2.4.1 Approach for creating a conceptual design

The approach for creating a conceptual design for I-DIANA started with an analysis of several DIANA competitors that already had a graphical user interface. The idea was not so much to mimic their approach to the interface, but rather to observe how different FEM concepts had been translated into interface concepts, and how well this had actually worked out.

**Conceptual design of competing products** Two similar products were available for study. Example screens of these products, MARC/MENTAT II and

I-DEAS, are shown in Figure 2.5 and Figure 2.6, respectively. Both were briefly studied.

The main concept of the interface was identical for both systems. In each interface the representation of the model takes a central place in the user interface. This representation is used to define the model, create the mesh and to add boundary conditions, constraints, and other loads. The representation always reflects the current state of the input, and changes over time with changes in input. The model representation is displayed in a large window, which is the focal point of the interface. Calculations are also set up and executed through this interface, with a graphical rendering of the results being displayed in the same representation area.

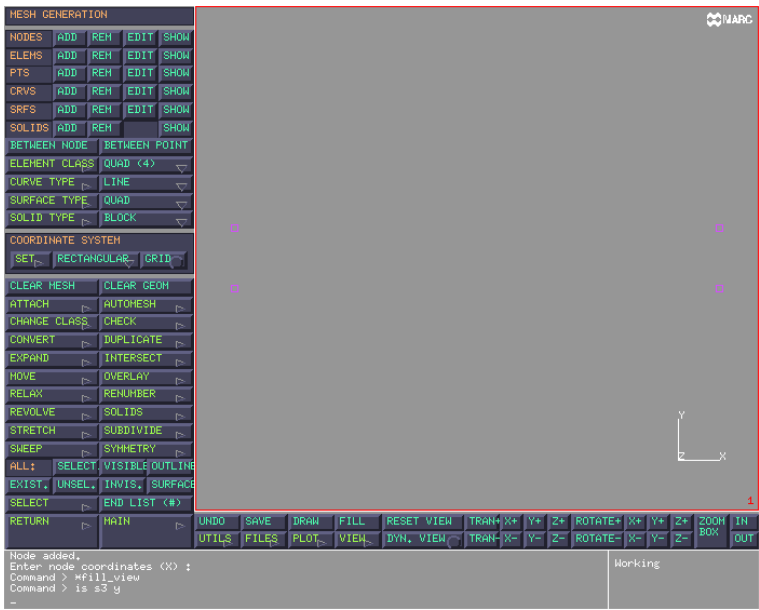


Figure 2.5: Screenshot: MENTAT II mesh definition

Additional input, control, and output tools are displayed in separate windows, one for each set of functions. These windows can be located around or over the main window, depending on use and personal preference. Access to



these windows is provided through a large menu structure, or, when the functionality depends on functionality already present, through other windows.

Consider, as an example, Figure 2.5 which shows the initial steps of a mesh definition using MENTAT II [MAR96a]. Clearly the model provides the main focus point in this interface. Instead of actual physical windows, MENTAT II uses fixed areas of its window to provide access to the different functions available in the system. It is also noteworthy that the command line interface is visible at the bottom of the screen, this provides a means to enter commands using the keyboard as well in MENTAT's command language.

The I-DEAS interface is displayed in Figure 2.6. While this interface looks different, it is actually strikingly similar to the MENTAT II interface. The only difference worth noting is that I-DEAS uses separate windows, for the four main functional areas of the interface, allowing the user some more flexibility in arranging these areas. I-DEAS also clearly provides a strong focus on the model, which will be shown in the Graphics Window.

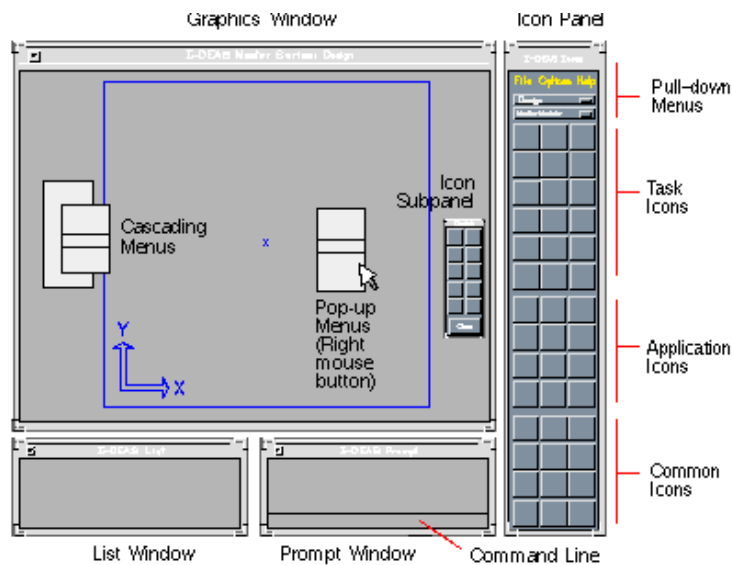


Figure 2.6: Overview of the I-DEAS user interface

**Lessons learned for DIANA’s conceptual design** A major lesson learned by looking at the competitors’ products was that a visual representation of the model is very important. Both competitors had put great emphasis on presenting a visual representation of the current model. This emphasis was in accordance with the results obtained from the interviews (Section 2.3.2) and the task analysis (Section 2.3.3) carried out during the analysis phase.

In terms of organization, both systems use a similar approach. Options are presented through a menu structure. Once the options get detailed enough, values can be entered. The menu structure is organized into several different functional groupings, for instance, all options and functions regarding creating a mesh are grouped together. This makes it easy to remain in this ‘mode’ of functions while working on the mesh, and then to switch to another group of functions for another task.

### 2.4.2 Prototyping for technical feasibility

The technical heritage attached to the DIANA project implied that prototyping must be done right from the start, not to validate the current design ideas, but rather to assess the technical feasibility of the design implementation.

The architecture of the current DIANA code was not suited to use in an interactive system. Since rewriting it was not an option, only small improvements could be made. The problem with the existing architecture is its strong dependence on structured batch input. This architecture is assumed throughout all code, e.g., input is not checked until a relevant module needs to look at it. As a consequence, each module knows its options, dependencies, and valid commands, but this information is not available at a higher level of the software. In fact, a module only knows which options are valid at a given position in the input file when it has successfully parsed all previous input. Appendix A describes this architecture in more detail.

To construct the information for model building and computation specification in an interactive way, this information needs to be available much earlier. One would like to have all dependencies, constraints, etc., beforehand, so that they can be used to create a proper presentation of available options to the user. Given that it was not possible to do a major rewrite of the DIANA code, some elaborate ‘tricks’ were needed to distill this information from the current code so that it could be presented to the user in an interactive way. Testing these ‘tricks’ required early and extensive technical prototyping. This early prototyping had its consequences on the design process in that it consumed considerable time, and limited design options.

### 2.4.3 Conceptual design: 1st generation

The first generation of the conceptual design for I-DIANA was initiated before it became clear that the current software could not be changed significantly. The focus of the design was similar to those of the competitors shown in Figure 2.5 and Figure 2.6. It provided an interface with a focus on the most tangible part of finite element analysis: the model to be analyzed. It consisted of a single large model view, and several buttons and menu bars to allow access to all functionality of the system. This type of design makes it possible to address two of the three important issues found during analysis: better support for input and better error handling.

Better, interactive, input support is almost automatically supported by this type of interface. Direct manipulation of the model, as was implemented by the two competing systems, can be easily supported because it is central to the interface. The visual representation of the model also makes supporting input easier. For instance, nodes can be placed and connected visually. Error handling is improved in a similar manner: when errors occur during user input, they can be noted immediately, and, if applicable, their location in the model can be shown visually.

**Design problems for this concept** Two main problems are found when this concept needs to be translated into an actual interface design.

First, it is hard to design the representation so that it can be used easily. The model representation is often three dimensional, but needs to be displayed in two dimensions. This in itself does not pose a big problem, but when additional information about the model is added the display quickly becomes cluttered. So many attributes and properties can possibly be displayed on the model that the display can become cluttered and useless. Designing which data needs to be displayed, and which different types of data need to be superimposed, is hard because of the amount of options and their interdependencies.

Second, organizing all the functionality of DIANA in this interface is also hard. Analysis made it clear that there are many different groups of users. Each of these groups uses a different small subset of DIANA's functionality; because each group uses a different but overlapping subset of functionality it is very hard to group functionality in such a way that it can easily be used by each group. DIANA does provide its own grouping, but this grouping is rooted in the types of algorithms and problems being addressed, not in the way people work with DIANA.

**Evaluation of the 1st conceptual design** Before trying to solve these design problems, the first conceptual design was first evaluated by the development team. There was much discussion about this approach, and in general there was a consensus that this would be a very good approach towards a new user interface for DIANA. The experience with competitive systems in particular supported this opinion.

Doubts were voiced by the software engineers. They pointed out that this would require a major rewrite of almost all the existing code of DIANA. Obviously the current algorithms and structure could be re-used, but even just rewriting all existing code would require a tremendous effort. The effort might better be spent on developing new DIANA modules. They also warned that this would likely introduce new bugs into existing, stable, and tested code. The software engineers argued that a rewrite would be necessary because the current code did not support such an interactive approach; and the continuous visualization, including input through the visual representation of the model, was conceptually very different from the current approach with input files containing tables.

The tight budget for development of I-DIANA, both in terms of manpower and money, made the project leader decide not to pursue this particular design approach, but rather try to find an approach that would not involve such a major rewrite. His decision was supported by several arguments. First, it was not clear whether such a large project, including a complete rewrite, could be supported financially, and second, such a major rewrite would take too long. An initial rough estimate was 2–3 years before a shippable version could be delivered, while the people backing the project financially wanted to see at least some progress or results at a much earlier stage. Finally, such a major rewrite would also mean moving away from the current batch-oriented philosophy, possibly losing this feature altogether and there was some concern that this would alienate existing users.

#### 2.4.4 Conceptual design: 2nd generation

The decision not to pursue the first design approach added additional constraints to the I-DIANA project. Most importantly, the new interface would have to be compatible with the current implementation of DIANA and keep the current structure intact. The biggest consequence for the conceptual design was that the interactive visualization could not be realized. It was simply too different from the current structure of DIANA.

This led to a revision of the goal for I-DIANA:

*Redesign the current 'control' layer to be interactive, and extend its functionality where possible.*

The 'control' layer controls all communication between the user and the DIANA modules. It currently does this by parsing the batch files and passing on the relevant parts to the modules. In the new design the 'control' layer would be extended to be interactive. The planned improvements are summarized in the following four goals:

- realize interactive creation and editing of input files, including interactive support for syntactic and semantic errors;
- allow visualization simultaneous with input, without explicitly creating a suitable input file;
- support for arranging the calculations to be made and their parameters;
- manage and track ongoing calculations, e.g., by showing timing estimates and current status.

While this approach poses an important improvement over the current situation, it does not address all the issues raised during analysis. In particular, interaction would be much less well supported when compared to the previous design approach, primarily because it lacks a visual, interactive, representation of the FEA model.

**Creating input files through the user interface** Creating the input files was seen as the hardest problem of this conceptual design, both from a technical and a design viewpoint.

The design problem with creating an input file is that there are so many possible options available at any time, and that these options have complex interdependencies. Solutions need to be found to present the options and their relationships, and for navigating through the options. A prerequisite for making this possible is that the allowed options and relationships are known in advance. The following list of requirements should ensure a good design for the creation of input files:

- entering options and values should be flexible;
- it should be easy to go back or undo options;

- error messages should appear at the moment the errors are made;
- all options should be listed in a hierarchy so that the user knows what options are available and what the result of choosing one will be.

Unfortunately technical barriers prevented us from fully satisfying these requirements. To understand why, some knowledge of DIANA internals is needed. The architecture of DIANA consists of a number of independent modules, each of which is subdivided into individual segments. Responsibilities, e.g., parsing of the input file, or checking values for options, are pushed down as much as possible onto the segments. This keeps the system flexible, and allows individual segments to be added or changed easily, but the implication is that arguments are not checked until the segment they belong to is actually run. In part, this can not be avoided because some options or values depend on results obtained from earlier calculations, however most options, and in particular the syntax, could be checked earlier. The reason for not doing this now is that in a batch environment it is easier to make a single pass through all the input instead of two separate ones.

This architecture makes it all but impossible to know all options and relationships in advance. The information is encoded directly into the individual segments, and can not be parsed from it. In fact, as DIANA's technical writer admitted, even the manuals do not contain *all* the detailed information on the relationships between options.

### 2.4.5 Conceptual design: 2nd generation, revised

The second conceptual design was revised to accommodate the technical limitations. Unfortunately, this could only be done by sacrificing some of the requirements for supporting input. In particular, going back and undoing an option, and getting a hierarchy of possible options could not be satisfied within the technical constraints. Although this design would still support users better than the batch input system, it also did not address several of the items found during analysis.

**Possibilities within technical constraints** The final design that complied to the technical constraints was essentially an interactive interpreter for DIANA. This interpreter interacts with the user, asking for options and parameters. It then presents the input to the relevant segments of DIANA, and parses the results of the segments. The interpreter can then present new options as indi-

cated by the segments, or display error messages if there were any, along with a possibility to correct the problem.

While this method was desirable from a technical viewpoint, it was not very good in terms of user-centered design. Interaction suffered because only a single path can easily be followed. Backing up and trying something else, a useful strategy in what-if explorations, is no longer possible.

Making corrections is also a problem, due to the way DIANA stores and processes state information on an input file. Making syntactic changes is no problem, for instance changing a number because it is integer while a real number was expected; but if an error is caused by erroneous input earlier in the input file, then it becomes much harder to fix, because none of the segments expect the user to take back arguments.

**General user-oriented problems with DIANA** While creating the conceptual designs, the development team came to realize that many of the problems of current DIANA users were not so much related to a particular (graphical) interface for DIANA, but that they were caused more often by structural problems with DIANA.

Error messages, for instance, were often cryptic. They were usually meaningful only to the author of the segment which generated the message, but users who knew nothing about this specific implementation were mystified. Error messages were often phrased in internal DIANA terms, or simply did not give enough information to correct the problem. Some users told us in interviews about their interpretations of such cryptic messages. Users also complained about the way errors were reported, often hidden between all the numerical output generated by other segments. As a result it can sometimes be very hard to determine why a particular calculation has terminated prematurely.

Another important point was the organization of input handling. DIANA's structure has grown through history, and may make sense from some viewpoints, but not from others. A critical look towards organization and grouping of modules and segments might help to make DIANA easier to understand in more situations; also, more lenient or flexible option parsing might help to avoid many of the errors made during input. For instance, if an option is expected to be a real number, and an integer is encountered, this could be flagged as a warning, but the input system could still convert the integer and continue.

**Prototype for the 2nd conceptual design** A screen shot of an early prototype is shown in Figure 2.7. In this situation the user is providing information on loads to be added to the model. In DIANA related loads are grouped together in cases. When entering new loads into the model, three options are normally available: a new load case can be added, a new influence, i.e., a specific type of load, can be added, or a combination of existing load cases can be added. Since the user in the prototype just started adding load cases, no combination can be added yet, because this requires two cases to exist.

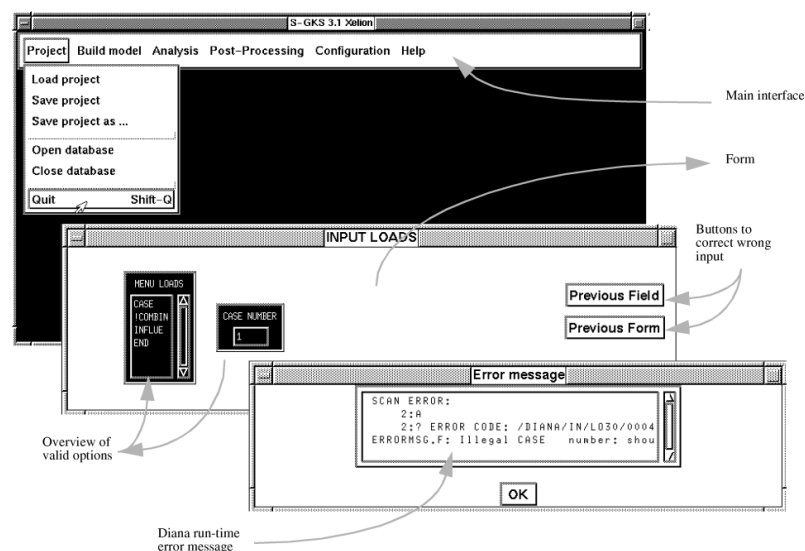


Figure 2.7: Early prototype screen shot of entering an input file

The screen shot shows the main window of the prototype with a menu bar controlling all interaction with DIANA. Adding loads is part of the Build model menu. The input loads window allows the user to add loads to the model. When this window first appears, it contains the menu loads menu, which allows the user to indicate which type of load needs to be added. Since no loads have been entered yet, no combinations can be made. This is indicated by the ! sign in front of the COMBIN option.

In the screen shot the user has selected the CASE option to enter a new load case. DIANA has responded to this request by adding a new widget, case



number, to the `input loads` window. This widget allows the user to enter the number of the load case to be added. For purposes of demonstration the user has entered an A as a case number instead of a valid number. This triggers an error in the input, which is signalled by showing an error window explaining the problem. After acknowledging the error by clicking the `Ok` button, the user can either correct the problem by entering a correct case number, or use the `previous field` button to revert to the `menu loads` menu.

**Task grouping through the menu bar** The main steps of finite element analysis are reflected in the menu bar. `Project`, `Configuration`, and `Help` are functions which are not specifically related to FEA, but which support the work and add additional value to the user interface. The menu items `Build model`, `Analysis`, and `Post-processing` resemble the three steps of the FEA process. `Build model` is equivalent to the creation of input files. `Analysis` is equivalent to the creation of command files, and running them through DIANA. `Post-processing` has no direct equivalent in DIANA, because it used to be done through input and command files, and a separate visualization program.

**Use of preferences and configurations** Creating small and domain specific interfaces, the last of three important issues found during analysis, was partly addressed through the configuration mechanism, located under the `Configuration` menu. With this configuration system, users of DIANA are able to select the modules and materials which they normally use. During model building and analysis selection I-DIANA uses this information to only show those options which are valid or useful according to the current configuration. This cuts down significantly on the amount of options and possible arguments. It is important to note that this configuration is done in terms of the user, e.g., selecting types of calculations, or types of materials, and that users can create their own configuration.

**Evaluation of conceptual design** Evaluation of this conceptual design indicates several shortcomings. Most importantly, interaction is not very well supported. There is some interaction in the sense that input is evaluated immediately, but these changes are not reflected immediately in the model representation, so it is not possible to have an interactive dialogue with the model data directly.

Error and input handling are much improved over the batch-oriented implementation. Errors are shown as soon as they can be found, they are clearly

presented, and their texts are being adjusted to yield more meaningful information. Input is also much improved, because all available options are now presented, as the type of arguments is clearly reflected by the interface, and because an overview of completed input is shown. One remaining problem with input is that only the next option or argument is shown at any time. This makes it hard to estimate what will happen after this argument has been entered.

Finally, small and domain specific interfaces are partly implemented with the configuration mechanism. This mechanism can be accessed through the Configuration menu item. This approach helps to keep down the amount of options and arguments and therefore makes the interface more manageable. It does not address the underlying issue, though, which is that users want to have a subset of functionality which resembles their particular domain, using appropriate wording for it, and to have pre-defined short cuts particular to this domain.

Still, this design approach addresses a fair number of complaints about the current batch-oriented implementation. It is expected that this new interface will make it easier for novices to work with the DIANA environment.

## 2.5 Reflections

The goals for the case study described in this chapter were:

- to apply current techniques and methods for analysis and design of interactive systems;
- to observe what kind of problems surface while doing this;
- to analyze the causes and generality of these problems.

A description of the analysis and design process was given in Sections 2.3 and 2.4 respectively. The two sections below contain reflections on both processes.

### 2.5.1 Reflections on the analysis process

Three different instruments were used during the analysis process to collect information: a questionnaire, interviews and task analysis.

**Questionnaire** The questionnaire fulfilled its goal quite well. As expected it provided a broad overview of DIANA's users, their application domains and their problems with DIANA. It did not provide much *detail* on any of these areas. The questionnaire was particularly successful in this case because DIANA users are such a heterogeneous and wide spread group. Obtaining a small sample of representative users is difficult without first knowing what types of users are present in what quantities.

**Interviews** After analyzing the results from the questionnaire a representative set of users could be determined for use during the interviews. The interviews proved to be very successful for getting a good overview of the different ways in which people work with DIANA, of their problems and their solutions. The interviews worked particularly well because they were grounded in the context of the current projects of the users interviewed. Allowing users to talk about their own projects in their own work environment allowed us to gain considerable insight in the way DIANA is used by its current users and how they incorporate DIANA in their work.

**Task analysis** The task analysis was done according to Bots' method [Bot89]. The author was already familiar with this particular type of task analysis, and knew about its strengths and weaknesses. Given the amount of problem solving involved in carrying out FEA, this task analysis method seems suited, as it is designed specifically for these kinds of problems.

None of the textbooks gave specific recommendations for one method or another; some form of HTA (Hierarchical Task Analysis) is generally used to explain the concept, and in fact most task analysis methods use some form of HTA. Bots' method also supports this, but adds to it by providing a means to chart those tasks which can not easily be unraveled. In short, Bots' method seems to be better suited for charting FEA work than other task analysis methods, because it acknowledges that some tasks cannot be detailed further, while still providing support for these tasks.

The interviews were the primary source of information for modeling the tasks, and the tasks were modeled on a basis of the notes made during the interviews. This process was facilitated by the structure of the interviews. During each interview a current or recently finished project was discussed. In each case the full lifespan of the project was discussed in chronological order, and the interesting or complex points were further detailed. This provided enough information on the global task hierarchy, and additional detail for those tasks

that were considered to be complex or interesting. In addition to the interviews, task analysis was also aided by the DIANA experience within the design team.

It can be argued that this approach was not exhaustive enough, for instance because the project was only discussed, instead of observed, or because only a single project was discussed in each interview. These issues do not seem to have caused problems. Most interviewees use DIANA as part of their professional work, and consequently they keep extensive records and notes of a project. Using these records and notes provided a level of detail at least as good as observation would have given.

Although each interviewee mostly discussed a single project, references to other projects were also made or asked for. The interviewees would often volunteer how similar problems were solved in other projects, especially when the current project used a non-standard solution.

It could be argued that in this case task analysis was not a success because the resulting task descriptions are not detailed enough. This is not quite true: task analysis did work, but only for the higher level tasks. These do not really provide much information, however, apart from some structure for the system as a whole. These high level tasks ended pretty much at the level of those described in Figure 2.3. Getting more detailed task descriptions was not possible, as is explained in the section accompanying the figure. Fortunately, Bots' decision structures did give some means to describe these tasks further.

With the use of a questionnaire, interviews and to some extent task analysis, not much information was missing to start the design process, based on the recommendations made by the HCI textbooks used. Some additional information might be needed later, e.g., actual artifacts used during working with DIANA, or additional observations of people working on FEA problems, but these would only add more detail. We felt no significant information was lacking when we finished the analysis process. Obviously additional information could always be collected during the design process once the need for this information had been established.

### **2.5.2 Reflections on the design phase**

The HCI textbooks used recommend four instruments for supporting the design process: use of design rationales, conceptual design, scenario-based design, and prototyping.

**Design rationales** Design rationales were not used. While this is a useful technique for exploring the different alternatives and consequences of particular decisions, it does seem to be geared more towards those choices where one of a set of options can be selected. Such design decisions usually start to occur later in the design process, for instance when particular interface decisions need to be made. It appears to be less useful early in the design process, when the choices are not yet known, let alone the options.

**Conceptual design** Conceptual design was the first instrument to be used to support design. As mentioned earlier, conceptual design is not a technique which can be used, but rather a more abstract approach towards design, before looking at the specifications of the user interface and its appearance. The key notion of conceptual design is to describe the concepts, objects, and functions of a system. It is hoped that these can be described without adding user interface details, and that this can provide ample possibilities for discussion and evaluation of the design.

During the case study two different conceptual designs were made, and they were described earlier in this chapter in Sections 2.4.3 and 2.4.4. Both of the descriptions are textual and often somewhat implicit. They were described in this way because there appears to be no representation for a conceptual design. None of the HCI textbooks give suggestions for such a representation, nor do any other publications we are aware of.

During the case study a number of communication problems surfaced in the design team. Since no particular representation was used, the conceptual designs were communicated through textual descriptions and verbal presentations, and both of these forms were open to different interpretations by different members of the design team. Recognizing and resolving these differences in communication took a lot of effort which could have better been used elsewhere.

In addition to the differences in interpretation, the abstract nature of the textual description of the conceptual design also caused trouble, because some of the design team members could not easily deal with this abstract description. A common solution to this type of problem is to use screenshots, sketches, or something similarly tangible. With DIANA's conceptual design this proved to be hard to do because many of the design issues were more abstract than could be represented by sketches; also, by using sketches for these issues, some implicit decisions would have to be made about the visual interface. This clashes with the goal of conceptual design in that it is meant to focus on the design

at a higher level than the widgets and windows of the interface. Finally, using sketches usually solicited discussion about the sketches or the associated interface issue, not about the real underlying issues.

Finally, using a textual representation for the conceptual design resulted in one more problem: dealing with the complexity of the design. The amount of information, issues, and relationships between them was very large; keeping them all in good order, in particular with respect to their relationships, proved to be very difficult.

**Use of scenarios** The use of scenarios during design was greatly simplified by having the results of the interviews available. Each interview provided a detailed walk through of a project. Each of these walk throughs could be used as a scenario during design. Scenarios were used primarily to evaluate specific designs, i.e. when a specific design idea emerged from the design team, it was often tested by running it through a few of the scenarios collected with the interviews.

**Use of prototyping** Prototyping was used primarily to evaluate the *technical* consequences of different design approaches. It was not used much for user testing, because no user testable prototypes could be created within the time frame of the case study.

**Overall evaluation** What was noteworthy about the design process for I-DIANA was that the whole design problem was so different from the examples given in the three HCI textbooks used as references.

One important difference was that the I-DIANA project team tried to renovate existing software, while the textbooks all assume that a system will be developed from scratch. With the large amount of currently installed systems this issue is likely to become more important over time.

With such a renovation project it is important to realize the distinction between the user interface and the interactive system itself. A bad or unusable system can only be improved marginally by a good user interface. Often it is much more important to improve the interactive system itself. This was also the case with this case study. For instance, improving error checking by making checks earlier and by improving the error messages improved the quality of the system without any changes to the user interface.

### 2.5.3 The case study and the research questions

It is important to retain sight of the relationship between the case study and the research questions posed in Section 1.3 for the remainder of this thesis.

#### Question 1

*How can we formulate the design activity for interactive systems which facilitate ill-structured work?*

This research question asks which activities are needed for design, with a particular focus on those activities related to ill-structured work. In this case study several relevant observations have been made.

The interviews proved to be a useful activity, although strictly speaking this would be an analysis activity. The interviews were particularly useful to uncover work context, which in turn gave a great deal of insight into the ill-structured parts of the work, for example by showing the forces outside the system involved with a decision.

Task analysis did prove to be useful too by providing structure at the highest level of the design. Once ill-structured parts of the work needed to be described, task analysis was not able to deal with this. The specific task analysis method used did at least have some representation for ill-structured tasks in the form of decision structures. Based on the HCI textbooks this is not likely to be available in many of the other task analysis methods.

The HCI textbooks did not offer much concrete support for creating the conceptual design. This is in line with the progression of the field as described in Section 1.2.3. Clearly some more guidance at this point in the design process could prove to be beneficial to the overall results.

#### Question 2

*How can ill-structured work be described explicitly during design without reverting to interface components?*

The case study has not provided much insight into this question, although some experience was gained with a few ways of representing design information.

Most of the design information in the case study was purely textual in nature, sometimes accompanied by sketches or mock-up screenshots. While the textual representation provides a wide range of possibilities, it is very much

open for interpretation. In fact, the design team did have communication troubles on a number of occasions because of different interpretation of the same design document.

The graphics that were included mostly were showing screens in the proposed interface. This often caused the focus of the discussion to shift to the details of that screen and of the sketch itself, instead of focusing the discussion on the big picture of the whole design and the role of the sketch in that.

Finally, the case study made clear that the information which needs to be addressed during design has a high level of complexity, especially in real-world problems.

### Question 3

*How can it be ensured that the formulated design activities and representation of ill-structured work provide a usable, workable, and fitting solution towards the design of interactive systems facilitating ill-structured work?*

This case study has made the importance and relevance of this question clear. The communication problems within the design team show that a usable, workable, and fitting solution for the design process does not happen automatically. One of the reasons is that not everyone in the design team relates to a textual description of the conceptual design in the same way, as described in Section 2.5.2. The two major issues found during the case study were the fact that the representation of the conceptual design was completely free-form, without any particular pre-defined structure, and the fact that only text was used. During the formulation of the design activities and representations in the remainder of the thesis care needs to be taken in this respect.



## CHAPTER 3

---

### Human Computer Interaction

---

A case study was presented in the previous chapter, describing the analysis and design of a usable complex interactive system aimed at supporting ill-structured work. The conclusions drawn from this case study were that current theory as described in HCI textbooks does not provide sufficient guidance for this process. In particular, support for creating a conceptual design and working with it is lacking [WZ96]; also, task analysis does not yield as much benefit as the textbooks suggest.

These issues are further explored in this chapter. General textbooks such as those used in the case study [DFAB93, HH93, PRS<sup>+</sup>94] are limited by their nature. They try to give a balanced and stable overview of the whole field of HCI, and cannot be expected to give much detail on specific issues. This chapter contains an exploration of additional information on task analysis and conceptual design, and whether this will provide further insights and methods not mentioned in the textbooks. This chapter also provides some definitions of several of the terms used in this thesis.

## 3.1 Terminology

The terms and definitions used in conjunction with the design of interactive systems are often confusing because they mean different things to different people. In this section a number of these terms are defined according to their use in this thesis.

### 3.1.1 The development process

In this thesis the development process is taken to be the whole process of developing an interactive system, starting with the first ideas about developing it, and ending when the system is in use. As such, the development process spawns a wide variety of activities. The research questions posed in this thesis are concerned with only part of the whole development process.

The analysis, design, and realization activities are the parts of the development process mostly referred to in design literature [Jon92, RE91, for example]. Several additional activities can be identified, such as evaluation [But96], and implementation, i.e. the actual introduction of the system in the workplace. All these activities combined make up the development process. It is important to stress that no specific order of activities is implied. Some precedence relations exist between these activities, for instance, the design process can not be started before at least some analysis has been done. The iterative nature of the development process, however, precludes any ordering between the different processes, and they should all be considered to be interconnected [Law90]. The boundaries between the different activities within the development process can be very hard to determine. Lawson argues that the products of these activities are what actually matters, not the classification of each activity during the development process [Law90]. In this thesis only analysis and design activities are considered.

*Analysis* aims to understand the problem by collecting information. Collecting information, both about the current situation and about the requirements for and possibilities of the system to be developed, provides the designer with a larger design space. More available information allows for more informed design decisions, and it allows for different design solutions to be considered.

*Design* is the primary focus of this thesis. Its objective is to restructure the information found during analysis in such a way that the problems which initiate a development process can be addressed. Design activities can be subdivided into smaller tasks, in particular when the design of systems is considered. As was already noted in the I-DIANA case study, it is useful to start

the design activity by making a *conceptual* design. In this thesis this part of the design activity is also called the *early design* activity. During this design phase a number of alternatives are often briefly developed, after which the most promising alternative is further detailed, providing the basis for further design activities.

This description is only one way to look at the development process. For instance, Hartson and Hix propose a star life cycle for the development of interactive systems [HH89]. This life cycle, shown in Figure 3.1, provides an interconnected and highly iterative approach to development. The extent of the central activity, evaluation, can vary significantly from essentially none to extensive study. What remains constant is the focus on iteration, and on a gradual shaping of the design through a succession of activities.

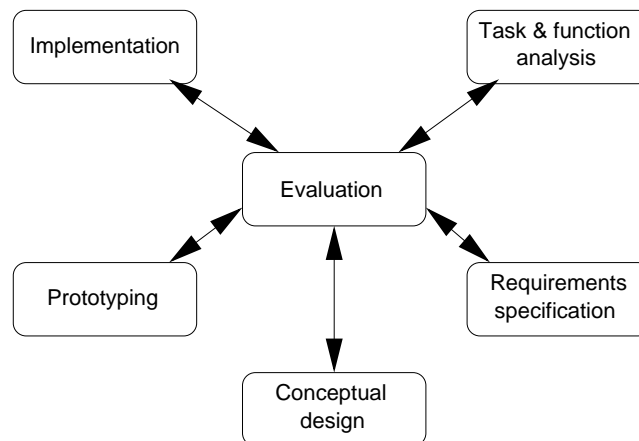


Figure 3.1: Star life cycle [HH89]

### 3.1.2 Interactive systems

Given the background and topic of this thesis some people might be confused by the use of the term ‘interactive system’. Instead, they would argue, ‘user interface’ should have been used. This argumentation is based on the fact that the user interface is the part of the system with which the user directly interacts. It can be regarded as an interface in the traditional, software engineering, sense: as a well-defined place where interaction between two system compo-

nents occurs. It also pays homage to those times when the technical challenges of graphical interfaces were the paramount concern of HCI. HCI practitioners are currently finding out that the user interface is only one of a whole set of issues with which they should be concerned. Therefore design of *interactive systems* provides a much more inclusive term.

John Seely Brown, director of Xerox PARC, said in an interview that

"design is moving beyond objects to supporting relationships, to work practice, to social intercourse. [...] I think more often of a whole set of offerings around something, not of the product itself." [Rhe95b].

In the remainder of the interview Brown keeps stressing, at several points, that products can not be separated from the environment they are placed in. This emphasis on the work *environment* is also witnessed by several of the new approaches for analysis such as those described in Section 3.2.5.

In the light of this growing attention to interactive systems as part of their environment, it is clear that interaction designers should be concerned with more than just the user interface. Buxton explains this by noting three levels of interaction: physical, cognitive, and social interaction [Bux94].

- Physical interaction is related to our sense and control, and relates mostly to what is often regarded as the user interface: that part of an interactive system which we can see and manipulate.
- Cognitive interaction relates to our thinking and problem solving. Buxton argues that this level of interaction is least developed, even though it is the most important level for dealing with all data the information revolution is producing. Some work done in this area includes Norman's affordances [Nor93] and research done by Gibson [Gib79].
- Social interaction provides a social context for interaction. This social interaction is often lacking, although CSCW systems and a growing use of computers for communication are changing this.

The user interface only deals with one of these levels of interaction: the physical layer. Buxton shows that both the cognitive and social interaction are important as well. Both go beyond the user interface: functionality and structure of the application may impede or stimulate understanding of the data, the functionality may encode policy which has social influences in the work environment.

Note that this does not mean that the user interface can be seen separately from the interactive system at all. For instance, from the point of view of software engineering it is preferable to separate the design and realization of the user interface from the remainder of the application [vdMV92, for example]. This separation allows division of labor and modularization of the application, both deemed good things from a software engineering point of view. This leads to additional support for just considering the user interface, and nothing else, though. Care should be taken to consider the implications on all levels of interaction first, and only then look at the specifics of the user interface.

Newman and Lamming repeat many of the same points when discussing interactive systems in their introduction. They add that the crucial property of an interactive system is that it provides support for human activity [NL95]. The design of the whole interactive system, from the point of view of supporting its users with their work, is looked at in this thesis.

### 3.1.3 Tasks

This thesis contains many references to tasks and to related concepts such as process, activity, and action. Unfortunately no single definition of these terms is in general use. Every author or system uses the terms in slightly different ways. The terms are used in this thesis in a consistent way, which is explained below.

The differences between the different terms are a function of complexity or length of the task. For instance, the term ‘action’ indicates something very small and simple, while the term ‘process’ has a much lengthier association. Even so, the meaning of a single term can vary a lot. Compare for instance the use of the term ‘task’ by GOMS and by Bots’ method later in this chapter. In GOMS tasks are defined to be very small and well-defined, while Bots defines tasks at a much higher level of abstraction. In this thesis only three terms will be used to describe parts of the work of a user. These terms are ‘objective’, ‘task’ and ‘action’.

**Objective** The term ‘objective’ describe the objectives of the work of the user. Objectives provide the highest level of abstraction: their use allows a coarse subdivision of the work. A related term found in other literature is ‘goal’. Examples of objectives are ‘revise chapter 3’ and ‘add markup for indices’.

**Task** The term ‘task’ describes a self-contained piece of work. Normally several tasks are needed to be able to reach an objective. A task differs from an action in that a task usually includes some kind of problem solving or decision making, while actions are always straightforward. A task differs from an objective in that a task is a description of what needs to be done, where an objective describes the result which needs to be attained. Examples of tasks are ‘check English grammar of current sentence’ and ‘mark up important words in current sentence’.

**Action** The term ‘action’ describes concrete interactions of the user with the interface. An action is a well-defined sequence of steps the user carries out, with no room for decisions. This makes an action the lowest level description of a piece of work used in this thesis. In a different context the actual interactions the user has, e.g. press mouse button, move mouse, and release button, might also be used. Actions are the building blocks of the interactive system: their presence determines what can be done in the system. Examples of actions are ‘delete word’ and ‘add author index marker at current location’.

### 3.1.4 Ill-structured work

The notion of ill-structured work is introduced in Chapter 1. This term is derived from Sol, who discusses ill-structured work in the context of problem solving [Sol87a]. The distinction made by Sol between structured and ill-structured work [Sol87b] is related to the distinction between Type I and Type II work made by Panko and Sprague [PRS84]. Type I work is characterized by a large volume of relative low cost transactions, well defined procedures, structured data, and an emphasis on efficiency. Type II work consists of fewer but more costly transactions for which no definite procedure exists, transactions that are performed using less structured and often ambiguous data, with an emphasis on effectiveness. Landauer uses the distinction between Type I and Type II work as a basis for his argumentation. He claims that large successes have been made in automating Type I work, and that the drop in productivity increase is due to the difficulty of automating Type II work in a similar way [Lan95] (See also Section 1.1.2).

**Model-oriented vs. procedure-oriented definitions** Bots discusses the difference between structured, i.e. Type I, and ill-structured, i.e. Type II, tasks in detail [Bot89]. In an attempt to better understand the ‘structuredness’ of

a problem, Bots explores two complementary types of definitions: procedure oriented and model oriented.

Procedure oriented definitions focus on the novelty of the task, i.e., whether it has been encountered before, in combination with a lack of predetermined and explicit steps to carry out the task. These definitions are aimed at encounters with *novel* ill-structured problems. Once these problems are encountered and it has become clear how they can be solved, the procedural definitions of structuredness indicate that these problems are now considered to be structured. This outlook is less useful for working with interactive systems, because often solving ill-structured problems is an integral part of the work carried out using the system. These ill-structured problems will be encountered by the users time and again while remaining ill-structured.

Model oriented definitions focus on how much is known about the problem, for instance whether it can be expressed in terms of objectives, variables and relations. These definitions are a better match for interactive systems, because they make an effort to describe what the task is about, given that it is unclear how the task can be carried out best. The complexity in interactive systems is often caused by the relations between affecting variables and outcomes being unclear and because it is not clear what the outcome-affecting variables are.

**Basic components of problems** Young recognizes three basic components of problems, and argues that the structuredness of a problem depends on how much is known of each of these components [You84]. These basic components are: objectives, outcome-affecting variables, and relations between affecting variables and outcomes.

The *objectives* in the context of an interactive system are clear only up to a point. The interactive system is designed with particular objectives in mind, so these objectives are clear. When the level of details descends to working on particular tasks, however, the objectives may well be unknown up front. Consider for instance the task 'Construction of storage racks' described in the I-DIANA case study on page 37. In this task the overall objective is clear, i.e. to create a construction for storage racks within the given constraints of price, maximum load, etc. The specific objectives for a problem depend on the exact constraints of that problem. This means that the exact objectives of an ill-structured task may not be known in advance, even though the overall objectives for the system are known.

The *outcome affecting variables* are in effect the data objects which are ma-

nipulated in the interactive system. All of these data objects need to be known in the context of the interactive system, because they need to be explicitly included in the system's design. If a particular variable is not included in the system, then it can not affect the outcome supported by the system, even if perhaps it should.

The *relationships* between the outcome and the affecting variables is always known internally in the system, because all of these relationships need to be programmed explicitly into the system. This does not imply that the people using the system will infer all these relationships, however, for instance because some of them can be hidden from direct observation, or because the relationships can cause chaotic behavior induced by their interdependencies. This means that from the standpoint of the user the relationship between the outcome and the affecting variables can be quite unclear.

In summary, the structuredness of a task in the context of this thesis depends on how much is known about the objectives, and whether this depends on a specific task, and on the complexity and clarity of the relationships between the outcome and the affecting variables. If not enough structure is available then no programme can be devised for the task, and it is considered to be ill-structured.

## 3.2 Support for analysis

The three textbooks used during the I-DIANA case study all recommended task analysis. Task analysis is further explored beyond what is offered in the textbooks in this section. The purpose of this exploration is to find out whether task analysis can be used to give explicit attention to ill-structured work during analysis. For a more exhaustive overview of task analysis methods refer to Diaper [Dia89] and Kirwan [KA92].

After discussing task analysis in general, several new approaches towards analysis for interactive systems design are explored briefly to see whether any of them can aid in focusing attention on ill-structured work during analysis and early design. One reason for looking at methods other than task analysis is that task analysis alone will not be rich enough to capture all aspects of an interactive system. Task analysis is mostly an algorithmic description of events happening. It can be argued, however, that an interaction paradigm is more rich than an algorithmic paradigm [Weg97], which implies that task analysis may be a useful approximation, but can never fully describe what is going on.



### 3.2.1 Task Analysis methods

The term 'task analysis' implies that some active type of analysis is conducted on the tasks. With most methods this is only implicitly the case: these methods consider construction of the task model to be the analysis. Methods that do actively support analysis on a previously constructed task model are almost always based on cognitive psychology theory and principles. The purpose of such analysis methods is to predict performance for a number of tasks, given an existing interface to evaluate. A prime example of such methods is GOMS (Goals, Operators, Methods and Selections) [CMN83]. Examples of methods based on similar principles are CLG (Command Language Grammar) [Mor81] and TAG (Task Action Grammar) [PG86, SG90].

These analysis methods are based on cognitive psychology theory. In this theory people are seen as information processors, but it can be argued that there is more to people than just entities shuffling information. In addition, Greif has not been able to verify that GOMS models give reliable predictions of actual performance in his experiments [Gre91]. Regardless of the arguments against these methods, using this type of task analysis will not be useful to reach the goal of this thesis. These methods can only be used to analyze existing systems for the purpose of evaluation, and it is unlikely that these methods are able to support the design process other than through evaluation of specific interface alternatives, which is not done until after the early design has been created.

### 3.2.2 Traditional task modeling

A better name for most 'task analysis' methods would be 'task modeling' methods, because this describes their purpose and what they facilitate much better. Almost always with such methods it is only an attempt to try and create a description with a task model. Explicit techniques for some kind of analysis beyond creating the description are very few.

Traditional task modeling methods usually consist of a variation on Hierarchical Task Analysis (HTA), where a hierarchy of tasks is created, with reaching the top level goal as the main task to be accomplished. The leaf tasks contain the actions to be carried out [PBACMF88]. Sometimes several levels of tasks are described separately [Bas93], but this does not change the basic principle.

This method of task description originates from the first applications of task analysis in the late forties and early fifties as part of the then evolving human factors movement [But96]. HTA was very well suited to the type of work for

which models needed to be created in that period, as the work was mostly operator-based, i.e. an operator operates a machine according to a fixed set of tasks and possible operations. Tasks were clear and unambiguous, and the leaf actions consisted of simple operations on a machine, such as pulling a lever or monitoring a gauge. A more modern but otherwise similar example can be found in the Operator Function Model (OFM), which is used to describe satellite monitoring jobs [Mit87].

**Application of HTA to interactive systems design** HTA and its derivatives provide a simple yet powerful means to describe the goals, tasks, and actions associated with operator work, but when they are used for the design of interactive systems additional requirements emerge. With the design of interactive systems the purpose of the designer is not to describe the tasks in a task model, but rather to use the task models to facilitate analysis. This means that task modeling has changed from an end into a means. This is also the way in which task modeling is used in the DIANA case. In addition to this changed use of task modeling, two problems with HTA make these methods less suited for use during the design of interactive systems.

The first problem has to do with context. HTA only describes tasks. Many different ways have been found to describe tasks and their relations, but the context of these tasks is almost always ignored. This was not a major problem using the traditional application of HTA, because operator tasks are always combined in leaf tasks, describing specific machine operations, and the decisions and possible courses of action are described up front using explicit rules. Finally, in the past operating the equipment was the goal of the work, so all context was contained within the equipment or the rules applying to it. In contrast, operating an interactive system is rarely a stand alone goal, more often it is one of several means to reach another goal that is external to the interactive system. This causes the context to become, at least partly, invisible to the system: external documents, phone calls, people coming in, weather status, all of them and more can have an effect on the tasks being carried out and on the selected strategy, but this context is not explicitly represented in HTA.

The second problem has already been encountered and described in the case study. HTA cannot deal well with ill-structured work. The reason for this is obvious: ill-structured work is ill-structured precisely because no program exists to describe it. If such a program did exist, the work would not be ill-structured. HTA provides such a program, and hence cannot be used to describe ill-structured work, even though it is suited to identify the tasks up to

the point where they become ill-structured.

Both of these problems make HTA not very suited for supporting the analysis and design of interactive systems.

Another way of looking at this is to realize that the formal tasks and activities are not always the same as the actual activities [Sac95]. Sachs looks at an organization through different lenses, for instance an 'organizational, explicit' view and an 'activity-oriented, tacit' view. Both of these are valuable and necessary viewpoints, but methods such as task analysis often favor the organizational view over the activity-based view. In contrast, the activity-based view often is much more relevant to ill-structured work than is the organizational view.

### 3.2.3 Adding knowledge to task structures

The first problem noted in the previous section, i.e., not providing contextual information, has been addressed by TKS (Task Knowledge Structures) [Joh92]. Use of TKS not only facilitates modeling of the tasks, but also modeling of the knowledge people possess *about* these tasks. This includes knowledge of goals, procedures, actions and objects. In addition to providing a method for analysis and modeling of this information, TKS can also be used to identify how typical each task is for the domain, and its importance for the overall goals.

TKS is commonly used as the foundation for an analysis method called KAT (Knowledge Analysis of Tasks). This method uses TKS to store the knowledge of people about their tasks. With KAT a number of techniques are employed, e.g., observational techniques, concurrent and retrospective protocols, and frequency counts, to collect the data for the TKS model. KAT then provides a structured method for identifying TKS components, and for identifying the representative, central, and generic properties of tasks.

While TKS and KAT address the first problem found with HTA methods, they do not explicitly address how to deal with ill-structured work. This is illustrated in the case study of a direct manipulation CAD system for use in the jewelry business [Joh92, 187–192]. The task model for this system contains a number of tasks which most likely contain ill-structured work, for example 'decide overall shape' and 'decide position'. None of these tasks have been further detailed in the task model. The case study description does not make clear in what way support for these tasks is reflected in the CAD system. In fact, it is likely that the user needs to make these decisions based on knowledge and esthetics, but whether the system can provide assistance with this, and if so, how, remains unclear.

A follow up system to KAT called ADEPT goes one step further by providing an integrated design environment [MPWJ92, WJK<sup>+</sup>93, WJ95]. The goal of ADEPT is to bridge the gap between psychologically motivated modeling, and implementation oriented modeling. An attempt is made with ADEPT to provide this bridge by integrating a number of different models within a single design environment (see Figure 3.2). These models are based on a task model and a user model, which are combined in several models, finally yielding the ‘concrete interface model’, which describes the resulting interface.

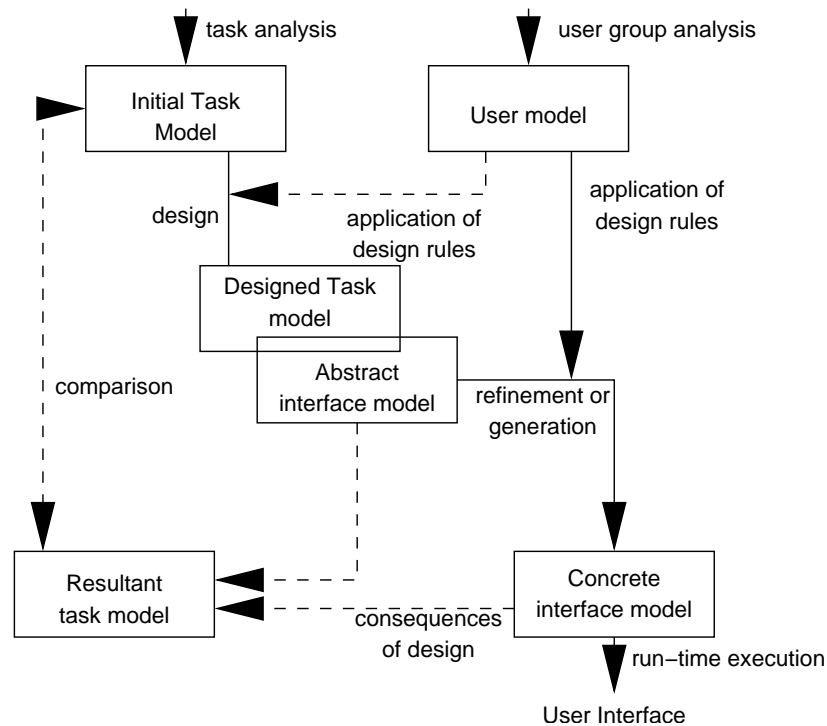


Figure 3.2: ADEPT models and processes [WJK<sup>+</sup>93]

The approach taken in the ADEPT method has both its good and bad sides. On the positive side it includes an explicit redesign of the task model composed during analysis into a ‘designed task model’. This allows the designer to explore new solutions, instead of just improving an existing solution. The use of

these models is also good in that it requires the designer to be explicit. This facilitates communication and discussion about the design within the design team. The ADEPT models require a lot of explicit detail right from the start, however, and this can easily turn out to be a drawback; commonly there is little detail known when the design process is started. Finally, it is not clear how contextual information plays a role in ADEPT, and in fact the authors point out that this should be an area of further study.

### **3.2.4 Task modeling for ill-structured work**

The task modeling method described by Bots does have a specific representation for ill-structured work [Bot89]. Bots uses a simple HTA model called a 'task structure' to represent the tasks in this task model. A little bomb inside a task box indicates that this task can be exploded, i.e., further detailed. For structured work, the exploding is done using additional task structures, thus creating a normal HTA hierarchy.

Some of the tasks need to be exploded further, but because they represent ill-structured work, this can not be done with additional task structures. Bots then introduces another representation specifically for these ill-structured tasks: decision structures. These structures get their name from the fact that in Bots' domain area, decision support systems, ill-structured tasks often involve problem solving and decision making tasks. A decision structure contains one or more decisions, and information elements related to these decisions. These information elements can be used as input or output of decisions, or both. A decision structure is an example of a model oriented description of an ill-structured task. It defines what the problem is, but not how to work through it.

Bots is specifically interested in supporting problem solving within decision support systems. While his approach may not be directly applicable to the design of interactive systems, it does show that, in the context of task analysis, dealing explicitly with ill-structured work is possible. Bots' method was used in the case study described in Chapter 2.

Finally, Bots does not provide a means to add additional knowledge about the context of tasks to the task model. The only contextual information which can be added in his models is what kind of information is associated with particular decisions in the decision structures. No other contextual information, e.g. goals, underlying assumptions, frequency of tasks, can be added to either the task or decision structures.

### 3.2.5 Other developments in interactive systems analysis

In addition to task analysis there are several other approaches towards analysis of interactive systems. These approaches have in common that they emphasize the strong relationship between work and its environment. Kyng argues that new representations are needed to take the environment into account [Kyn95]. Unfortunately he does not provide such a representation himself, and it can be argued that this is not easily possible either [Ban95]. Draper provides an argument for sometimes using artifacts instead of analyzing tasks, but remains too brief to build upon [Dra93]. Bannon and Bødker also argue that artifacts are important, and should be studied while in use: “artifacts are crystallized knowledge” [BB91]. Fischer introduces explicit representation of context and intent, but does so for a domain-oriented design environment [FNO95].

The three approaches discussed in the remainder of the section also do not provide ready for use representations. What they do offer are approaches in which the context of the work gets explicit recognition. As such they might prove to be useful with regard to the central research questions in this thesis.

**Ethnography** Ethnography in and by itself is not a new approach. Developed within anthropology, it refers both to the process of conducting field studies and to the written text produced as a result of these studies. The application of ethnography to HCI did not start until the eighties. It began primarily as a means to focus on the social and cooperative aspects of computing, and to counter the laboratory experiments on which much theory was based until then [Blo95].

Indeed the primary asset of ethnography for HCI is its broader focus on the socially situated nature of HCI. This social context of the work people do can be particularly valuable when designing CSCW systems. One of the major drawbacks of ethnographic studies is the long period of time they tend to take. It has been proposed to make much shorter observations according to the same principles to get at least the most important social aspects into focus [HKRA95].

A second drawback of using ethnography for HCI is that the standard reporting method for ethnography consists of written text. This representation is often not suited for the dynamics of the workplace, in particular when design of interactive systems is considered. In effect, ethnography is aimed primarily at analysis, although work is underway to bridge the gap between ethnography and design [BD96].

Ethnography provides a very broad perspective on work, in particular from a social perspective, and hence will mostly guide the design of social interac-

tion. It is argued that the experience gained from an ethnographic study can not easily be made explicit in a condensed form. While ethnographic studies will add insight, they do not provide specific ways to describe contextual knowledge or ill-structured work.

**Contextual Inquiry** Contextual inquiry is a method for collecting information during analysis in a structured way [HB96]. It was initially developed by Digital Equipment Corporation's usability team as a means to come to a better understanding of how users work. The primary means of collecting information in contextual inquiry is the interview. These interviews are structured according to an apprenticeship relationship between the designer and the user. One of the premises of contextual inquiry is that understanding work at the day to day level of detail can only be achieved through observation. Even the users themselves can not properly describe what they normally do, because the details of their work become second nature to them, and thus 'invisible' because they are not consciously carried out.

The apprenticeship model of the interviews uncovers many of these details. It allows the designer to observe the user and to engage in discussion about observations and interpretations. Retrospective accounts are also used to uncover information which cannot be uncovered during two or three hours of observation. With a retrospective account the user leads the designer step by step through some event, using artifacts as a means to guide the explanation. This technique is identical to the way interviews were conducted in the I-DIANA case study (Section 2.3.2).

Contextual inquiry also provides a way to interpret and model the data collected during the interviews. Contextual inquiry uses five diagrams to model work:

**Context model** The context model shows the environment of the user in terms of organizational culture, policy, and procedures. It serves as a check for the acceptance of particular design solutions.

**Physical model** The physical model shows the physical layout of the work. It is used to model movement between spaces and different kinds of communication.

**Flow model** Flow models show people's responsibilities, communication, and coordination. This model can be used to uncover the different roles associated with the work.

**Sequence model** Sequence models show which sequence of action is needed to carry out a given task. They are similar to flow charting or task models.

**Artifact model** Finally, artifact models describe the artifacts used in the work, including their structure, usage, and intent.

Together these five types of models give a complete description of the work. The models and interviews provide a very large amount of information, so contextual inquiry also supports a way to condense the information. Affinity diagrams play a key role in this process. In these diagrams related notes from the interviews are clustered together. This technique allows a structure to surface from the collected data, instead of imposing one from the start. For each type of model a single consolidated model is created based on the individual models of that type that were created based on collected customer data. These consolidated models describe current work practice and can serve as a starting point for redesign.

In fact, Contextual Inquiry was later extended to become Contextual Design [BH98]. Amongst other things a new model was introduced, the *User Environment Design*, which is similar in several ways to WONDER described in Chapter 4.

**Activity theory** Activity theory is a theory of people and the work they do [Nar96a, Rae91]. It has its roots in Soviet psychology originating in the 1920s. The object of activity theory is to understand the unity of consciousness and activity. Activity theorists argue that consciousness is located in everyday practice, i.e., “you are what you do”. Activity theory aims to understand the interpenetration of the individual, other people, and artifacts in everyday activity. It tries to evoke understanding by providing a set of perspectives on human activity, and a set of concepts for understanding that activity.

This very brief description already makes clear that activity theory is not a cut and dried theory, ready to be applied. Instead it is more a philosophical framework which can be used to come to terms with the activities we humans carry out. This also provides the link with interaction design. For instance, Greif notes that “artifact design is essentially activity design” [Gre91]. Activity theory provides a very basic understanding of this strong symbiosis between the artifacts we use, the activities we carry out with them, and the way this makes us perceive things. Due to this strong foundation, activity theory proponents claim that it has the best chance of becoming a good theoretical basis



underlying most of the field of HCI, instead of other theories such as cognitive psychology and situated action [Nar96b].

In the context of this thesis activity theory is not directly applicable. In its current state it does not provide enough guidance to help answer the research questions directly. Still, activity theory is interesting for two reasons. First, it affirms the trend towards paying more and more attention to work context when designing interactive systems. Second, it provides a strong focus on the symbiosis between artifacts, activities, and people. Both of these issues can provide valuable, although indirect, input when answering the research questions presented in this thesis.

### 3.2.6 Discussion

None of the task modeling methods described in this section support both of these characteristics. TKS does provide for contextual knowledge to be added to the task model, while Bots' decision structures provide an explicit description of ill-structured work.

Other developments in analysis support, such as ethnography, contextual inquiry, and activity theory, do not provide improvement over task modeling in this sense. These developments all have in common that they strongly focus on finding and describing contextual information, thus addressing one of the two important characteristics for good task modeling. None of these developments seems to acknowledge explicitly the existence of ill-structured work. At any rate, they do not provide any means to deal with it within their methods.

In summary, two important characteristics for a task modeling method for interactive systems development are:

- the task model allows contextual knowledge to be added;
- the task model can describe ill-structured work explicitly.

## 3.3 Support for conceptual design

The I-DIANA case study shows that support for conceptual design is both important, and lacking in HCI textbooks. Design theory states that design tools are needed to tackle complex design problems [Jon92]. Use of design tools can have negative side effects, in particular because it can easily limit the design space under consideration, or cause details not supported by the design tool

to be overlooked [Law90]. No design tools for conceptual design were found in the HCI textbooks used during the I-DIANA case study. The search for such design tools is continued in this section, first by examining how several design tools, either generic or aimed at architecture, can support the design process. Subsequently several design tools for conceptual design of interactive systems are discussed. Finally some requirements based on a real world study of designers are presented.

### 3.3.1 Examples of design tools in general

Jones provides an overview of design tools [Jon92]. Some of these design tools are very simple and straightforward; at first sight we might not even consider them to be design tools. Examples include literature searches, interviewing users, and brainstorming. All of these simple activities can be considered tools, as they help to structure and guide the design process. Jones also lists several more complicated and formalized design tools. One such tool is Alexander's method of determining components in a design [Ale64]. This design tool provides a method for finding the right physical components of a physical structure such that each component can be altered independently to suit future changes in the environment. The method consists of a number of well defined steps in conjunction with a specific representation.

**Architecture** Many parallels can be found between architectural design and interactive systems design. Some of the design tools used by architects show how similar techniques could be useful for design of interactive systems.

The use of design patterns is an example of an architectural design tool [AIS77, Ale79]. Design patterns aid in the design of coherent architectural structures from the very coarse level of city planning all the way down to the details on doors and windows. The design tool consists of a set of patterns, each of which describes the relations between the common architectural problems, its context, and a possible solution [Arg80]. Each pattern can be thought of as distilled design knowledge. Using these patterns to guide design means reusing all this design knowledge embedded in the patterns. This allows more informed design decisions to be made, and more alternatives to be considered. The application of design patterns for interactive system design is currently being investigated [Eri]. The software engineering community is already actively using this design tool [GHJV94].

Architects also design incrementally, using several layers to add more and

more detail [Won93]. They start out by making several rough sketches. By adding layers with more detail they can easily explore several options while still working on the basis of the original sketches. When one avenue of exploration does not work out, each layer can easily be removed. This use of layering as a design tool has two advantages. One, it always keeps the original rough sketches available. This keeps the design from deviating too much from the original plan. Two, it encourages exploration of different alternatives at several levels of detail, thus allowing the designer to try a wider variety of design solutions. Wong proposes that similar techniques be made available for interaction design.

### 3.3.2 Examples of design tools for interactive systems

There are only a few design tools which address explicitly the conceptual design of an interactive system. Three of these design tools will be discussed. The discussion will focus in particular on the target domain, the requirements on analysis, and on the way in which the conceptual model is described.

**Task oriented approach** Seaton and Stewart have described a task oriented approach [SS92]. They create their conceptual design using tasks which they have found during analysis. Only a few levels of tasks are included in this task model. The highest level consists of Departmental Tasks, which represent the main mission of the department. The next level of tasks are the Primary Tasks, which represent the business objectives of the department. Primary Tasks are composed of Secondary Tasks, while each Secondary Task is composed of one or more subtasks. The whole hierarchy is depicted in Figure 3.3. Subtasks are assigned to a role, and associated with specific documents or screens or both.

So far this looks like just another HTA derivative with a fixed number of levels and some additional knowledge attached to the leaf tasks. What makes this approach interesting is the philosophy of the authors that a system design based on task analysis will not be very usable. They argue that task analysis is effective for specifying functional requirements, but that it does not pay attention to usability. They also stress the danger of simply re-implementing the existing situation, instead of really creating a new design.

To counter these problems the authors use strong user involvement and very early prototyping. As soon as possible a prototype is created for one of the primary tasks, and implemented in the work environment. Step by step additional functionality is added to the prototype, partly based on the task

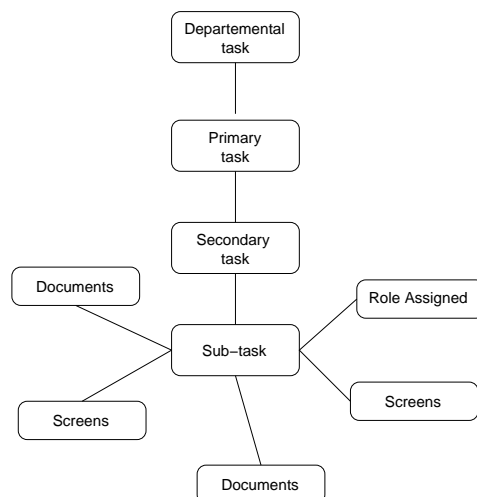


Figure 3.3: Seaton and Stewart's task hierarchy

model, and partly on user feedback. By using this mix of user involvement and structuring through the task model the authors claim to get a more usable and buildable system.

This proposed way of working is strikingly similar to that of film making. Heckel describes the process of animation film making at the Disney studios [Hec91]. This process starts out with a rough storyboard which outlines the main themes of the film, cf. the rough task hierarchy. From this storyboard a first, very rough, film is made, cf. the initial prototype. This rough prototype film is then gradually enhanced. Scenes are drawn in more detail, music is added, etc. Each addition is placed in the film prototype immediately, so that it can be viewed in context of the film as a whole. A similar step-wise refinement is proposed by Seaton and Stewart.

Although not explicitly acknowledged, this method appears to be aimed at the business domain. This focus can be inferred from the examples of the authors and from their comments in general. It is particularly visible in the task model, for instance because of the term 'departmental task'.

The method poses no specific requirements for analysis. Seaton and Stewart do not believe in creating a large and complete task model, but for reasons unrelated to ill-structured tasks. They argue that a detailed task model will most

likely reflect current work practice instead of describing a new and improved design. A coarse task model is used instead to define *structure*, and prototyping is used in close cooperation with the users to create a design within the boundaries of the task model. Thus the support for ill-structured tasks is implicitly added.

Seaton and Stewart only provide a very sketchy description of a conceptual model. The task model provides a good description of the structure, but is very coarse, while the detail is implemented directly into prototypes, documents and screens, each of which goes beyond mere conceptual design.

**MUSE/EXPOSE** A different approach is taken in MUSE (Method for User Interface Engineering) [GVQD93]. The MUSE method tries to comprise all of the development process of a user interface, but it is mainly focused on design and implementation. MUSE is contrasted with the approach of Seaton and Stewart in that it does try to create a user interface through a *structured* approach. To ensure usability of the final result, MUSE is guided strongly by software ergonomic principles such as those detailed in the DIN NORM 66234/8 on computer dialogues [DIN87]. A tool called EXPOSE is available to work more easily according to the MUSE method [Gor95].

Four phases in the development process are recognized in MUSE: conceptualization, structuring, concretion, and realization. The concretion and realization phases are not discussed here, because they are not aimed at the early, conceptual, design. The conceptual phase contains the analysis of the current work system and the subsequent design of the future work system. This analysis does include aspects of the work environment, work distribution, and communication needs. The result of this conceptual phase consists of tasks, workflow, and an initial tool interface. These results are expressed in an informal way; with several diagrams and techniques for expressing these results being loosely suggested.

The structuring phase builds on the results of the conceptual phase by selecting one of four abstract dialog forms: data request, multiple choice, command request, and object manipulation. The dialog structure is then defined by sequencing these abstract dialog forms according to the tasks and workflow found during conceptualization.

The conceptual model is defined by a number of different concepts, but the main structure of the model is provided by a specification of tasks. The conceptual design is created by selecting one of the objects found during analysis, its related task, taken from the detailed task hierarchy, and the users who have to

perform the task, as specified by their roles. A strong point of EXPOSE is that it contains domain specific advice in addition to generic ergonomic advice. This advice consists of suggestions for commonly used objects in the domain, of suitable methods for these objects, and of information on the different roles.

The ergonomic advice based on the role description can be used to provide suggestions for additional functionality to support the selected task, for instance, if a large degree of freedom in sequencing the subtasks is needed to carry out a task, then some kind of control function needs to be designed to facilitate this. Suggestions for meta-functions, e.g. providing help, and adapting functions, e.g. changing fonts or moving windows, can be suggested in a similar way. These suggested functions are then further evaluated based on ergonomic principles present in EXPOSE.

MUSE and EXPOSE are aimed at the business domain. This has the advantage of being able to provide specific advice for the domain, but it has the disadvantage of being less flexible for other domains. The business domain focus, i.e. a focus on transaction processing and form-based interfaces, is particularly clear in the structuring phase.

The main requirement for working with EXPOSE is the existence of a detailed task model. This model is explicitly required, and plays a central role in the remainder of the conceptual design activities. The detail of this model leaves little room for the support of ill-structured tasks, although some flexibility can be found in specifying the control functions, for instance by not specifying a particular sequence for subtasks.

Finally, EXPOSE's description of the conceptual model is quite rigid, and does not allow for much ambiguity or vagueness. The model is created by using software tools which may make it easier to work with, and more understandable for team members from different disciplines. The fact that EXPOSE requires a detailed task model up front also makes incremental design less needed, thus alleviating some of the need for ambiguity in the conceptual design model.

**UIDE** The User Interface Design Environment (UIDE) provides a different approach to designing and implementing an interactive system [FKKM91]. It is one of a number of similar systems, all of which are aimed at providing a single description of an interactive system which can be used to generate different interfaces for the system; an example of a system with a similar goal is HUMANOID [SLN93]. The main idea behind UIDE is that a single repository contains all information needed to describe the interactive system. Different

modules operate on this repository, facilitating different design and implementation tools, for instance, the information in the repository can be used to generate context sensitive help [SF90, dG92, MSN94].

Foley and Wallace proposed a four layered view on interactive systems well before the UIDE project was started [FW74]. This view consists of a conceptual, semantic, syntactic, and lexical layer. Each of these layers describes a different aspect of an interactive system, while together they provide a complete description of the system. Use of this layering allows the designer to shift attention between different aspects of the system by selecting different layers. In this sense these four layers resemble the use of layers in architectural design. A successful use of layering during design is demonstrated by the MOVE project [BHvdMS93].

The conceptual layer describes the interactive system at an abstract level, making it quite suited for conceptual design. Foley sees the conceptual layer as the user's model of the application [FvDea82, p. 82]. This layer typically provides definitions of objects, properties of objects, relationships between objects, and operations on objects. These definitions are reflected in UIDE's database structure, shown in Figure 3.4.

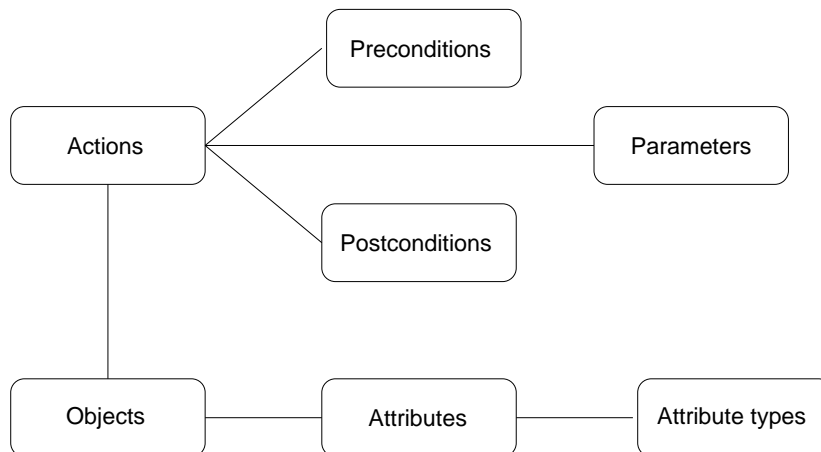


Figure 3.4: UIDE database structure [FKKM91]

UIDE provides us with two interesting points for discussion: an explicit way

to express the conceptual design, and the usability problems of UIDE.

The conceptual design in UIDE is represented as a collection of objects and associated functions. This representation is independent of a user interface: no particular presentation has been selected for an object, and functions can still be assigned to menu items, buttons, or something else. This makes the representation useful in terms of the second research question: no particular user interface elements are needed to represent it.

UIDE has some major usability problems, however, so this representation might not work as well as seems at first glance. The problems make UIDE, and similar systems, extremely hard to use by designers for real world design problems. Most of the problems are caused by its underlying philosophy, which causes the conceptual model to be so rigidly defined. This does not encourage gradual growth; everything needs to be specified precise right from the start, leaving no room for doubt or vagueness during design. This does not facilitate explorative design early on, even though this is very important in the early stage of design. Control over the mappings of the conceptual model to a graphical user interface, and over the appearance of the user interface itself, are also not very well developed, but this is a problem with the particular implementation of UIDE, not because of the philosophy behind it.

UIDE is not associated with a specific target application domain, although most of its examples are small direct manipulation applications. Use of UIDE also does not pose specific requirements on analysis, although good knowledge of objects and functions is needed to create the conceptual model, and thus implicitly requires knowledge of contextual information and a task hierarchy.

UIDE's representation of a conceptual model is very rigidly defined, and does not support any ambiguity or inconsistency. While this model is very precise and explicit, it does not facilitate incremental design, comments on the design, potential alternatives, etc. All of these will have to be handled outside of the conceptual design representation, potentially leading to loss of this information.

### 3.3.3 Representation of interaction as a design tool

Design representations can provide a very powerful tool for design, because their use allows the designer to focus on specific parts of the system while still keeping an overview of all of the design, and because these representations encourage easy exploration of possible solutions [Cro89, Jon92]. Tetzlaff stresses two objectives for a design representation for interaction design [TM91]. First,



it should model the procedural richness of the environment, task, or artifact to make sure that the solution will meet the user's needs. Second, it should abstract the distinctions and similarities pertinent to the development of an effective, conceptually coherent, and usable interface.

There are some inherent problems with design representations [Law90]. For instance, the designer needs to be able to interpret correctly the representations in terms of what the final product will look like. In addition, each representation has an inherent focus and bias; using only one representation makes it easy to dismiss other points of view and their associated constraints and problems. Another problem is that it is often easy to conjure up a compelling image in the design representation, while not paying attention to objections and concerns which might arise as a result later in the development process, for instance, by designing a user interface that requires too much computation to be feasible. These problems seem to be inherent to the use of representations for design. Not much can be done to the representations to alleviate the problems.

**Comparing interaction design representations** An analysis of six interaction design representations used in interactive media projects reveals some interesting information [Kol]. The following representations were used in the cases under study: technical reports, flow charts, state transition diagrams, linear demos, interactive demos, and hardware demos. The study reveals that none of these representations stands on its own; each one is embedded in the whole development process. Furthermore, once one representations builds on a previous one, the previous representation loses value, and moves into the background.

The study found several characteristics to be of particular importance:

**Revisibility** The iterative design process calls for continuous revisions of representations. Some representations make it very hard to track revisions, while others allow small revision easily, but not structural revisions.

**Incompleteness** It is almost impossible to create a complete representation. Instead each representation appears to focus on one particular aspect of the design.

**Roughness** It is hard to strike the right balance between keeping a representation rough so that people will assume it is open to changes and suggestions, and smooth so that people will experience the representation well enough.

**Accuracy** This is not an issue early on in the design process, but it does become more important to be able to verify or assess the accuracy of descriptions once the design progresses towards implementation.

**Interpretability** This means that representations made at one level of the design process can not be interpreted by a next level. This in turn implies that a manual interpretation or translation needs to be done later in the process, perhaps several times in sequence. Furthermore, this aspect also inhibits going back to previous design representations.

It is unclear to what extent these characteristics hold for conceptual design representations, though. The representations included in this study all focused on a specific implementation detail such as the visual interface, or the interactions between user and interface. For a conceptual design several of these characteristics are less important.

**UAN** Representations for interaction design currently focus primarily on the physical interaction with the user interface. State transition diagrams and event handlers [Gre86] are early examples of such representations. A more current, but similar, representation is User Action Notation (UAN) [HH93]. The goal behind UAN is to describe in detail the interactions of the user with the interface. According to the authors,

“UAN is intended to be written by someone designing the interaction component of an interface, and to be read by all developers, in particular those designing and implementing the user interface software”

UAN does provide a very fine level of detail, which is very useful given its goal. It allows the designer to specify the interactions very precisely, not leaving any guesswork to the programmer.

UAN describes the interaction for each task both with user actions and the corresponding interface feedback and interface state changes. An example of UAN notation is shown in Figure 3.5. In this example the cursor is moved to a specific file icon, and the mouse button is depressed. In response, the icon is highlighted, and the corresponding file is selected. To finish the task, the mouse button is released.

From this example it is clear that UAN provides a very low level description of the interactions of the user with the interface. UAN does provide some capabilities to create higher level descriptions, for instance by summarizing all

Task: <b>select file</b>		
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE
^[file icon'-!] Mv	file icon'!	selected = file
M^		

Figure 3.5: UAN example—select file [HH93, p. 167]

the interactions from Figure 3.5 into a single `select file` construct. When describing these higher level tasks the system feedback is not included, as this is guided by the lower level interactions.

Whether interaction representations such as UAN can help when addressing the research questions of this thesis is doubtful; UAN does not appear to be useful for supporting conceptual design for several reasons. For instance, UAN only partly addresses one of the two objectives by Tetzlaff mentioned above: it only provides a rudimentary mechanism for abstraction, which really is nothing more than summarizing interactions in a hierarchical fashion. This only provides a partial abstraction. For instance, feedback is not a part of this abstraction because it is generated by lower-level interactions. More importantly, UAN does not provide for any modeling of the richness of the environment and its artifacts, i.e. the work context.

UAN also resembles HTA methods in several ways, such as using a hierarchy of actions, and a hierarchical task structure which becomes more and more detailed. Despite this resemblance, UAN does not satisfy the two major requirements for task analysis methods discussed in Section 3.2: modeling of work context, and providing support for ill-structured work.

Taken together, these problems make it clear that using UAN for support of the conceptual design is not a viable alternative. Other methods for representing interaction suffer from the same problems, because they too are grounded in the low level interactions between the user and the interface. Similar arguments can be brought against their use for conceptual design.

### 3.3.4 Real world requirements for design support

A study amongst real world interaction designers has revealed a number of requirements on design support tools [RMK87]. Two types of development with two associated types of tools are found in this study. The first development

model is incremental; the design continuously grows from a small core into a full-fledged design. For this type of development there is a need for rich and modular environment, which can support the very first stages of prototyping up to the final system as it will be delivered to its users. The second type of development is phased development, where the steps from one part of the design process onto the next may be more radical. The authors claim that a simulation tool for simulating behavior of the system at subsequent phases of the development process is quite important here, because it allows the designer to get things right at a time when the design is most flexible.

Furthermore the analysis found a distinction between two different classes of tools. For well-specified problems a structured tool that guides the designer toward the best implementation of chosen interaction techniques may be best. For ill-structured problems, however, an integrated, relatively unconstrained environment where interface and functionality may evolve together will be best.

The results of a study into the use of HCI techniques in a real-world project also provide interesting insights [BSMH95]. Several lessons regarding the practical application of HCI theory are presented:

- make modeling less of a 'black art'; understanding the process is one step, but being able to apply it is another;
- provide positive as well as negative design critiques; design methods should also reinforce good design decisions, and not just focus on problems and broken or bad parts of the interface. This does not only yield a more balanced design tool, but also provides a better social grounding;
- sharing representations: context and participation; when sharing representations, it is also important to include or make otherwise available the context of those representations, especially if the representation is terse such as in diagrams;
- concretizing theoretical principles; abstract notions and diagrams of a design are often not enough, and some concrete examples such as sketches can help understanding and reflection;
- design culture and design formalisms; bringing more formal methods into a culture that is not used to them can pose problems of acceptance, and an active effort needs to be put in two-way communication about their use. Also, adaptation to a specific culture may be needed in a specific design job.

Karat and Bennet also see several useful techniques for supporting design while contributing and maintaining perspective on the user [KB91]. Some of the examples they give are:

1. tabular formats for representing system objectives.
2. abstract statements of a guiding vision for design (e.g. style guidelines).
3. early design prototypes as a basis for design iteration.
4. focus on the objects important to the user, and the actions on them.
5. scenarios of use.

They also pose several requirements for such a technique:

- fit into existing design activities;
- display obvious value for designers;
- emphasize early design, but results are useful throughout the design process.

### 3.3.5 Discussion

If one conclusion can be reached, then it is that support for conceptual design is far from complete. Each of the example methods presented in this section has its own set of problems.

The choice of a target domain appears not to have significant consequences, although the business domain allows a much more rigid description of the conceptual mode, because there will be less ill-structured tasks. This is caused by an implicit assumption that these systems will be transaction or form based.

Both EXPOSE and UIDE pose strong requirements on the information collected during analysis. These requirements are reflected in the rigidity of the representations of the conceptual model in both methods. In making these assumptions on the information available from analysis, the developers of both methods have pushed the conceptual design activity back into the analysis activity. This has the very real danger that no conceptual design will be done at all, for instance because the tasks found during analysis are used directly as the conceptual model for the new design. In addition, it breaks the important iterative and symbiotic relation between analysis and design as described in Section 3.1.1.

Seaton and Stewart are much more flexible in their requirements for their method, but their conceptual model is not extensive enough. By going to prototyping quickly they make a whole range of the design implicit in their prototypes. The model they have, i.e. the task models, only defines the structure

of the application, but nothing else. All other information will be implicitly encoded in the prototypes. While this works around the problems with ill-structured tasks, it also makes it much harder to evaluate the design, and work on it iteratively.

Finally, the conceptual models of all methods contain the same kind of ingredients, i.e. objects and functions, in some way. It is clear that both play an important role in defining the conceptual design in some way.

The more general discussion of design tools does not provide much additional possibilities for supporting conceptual design. While it contains some good generic hints on design tools, it does not provide much guidance for the design of interaction. It gives some good criteria for evaluating design tools, but nothing more. Similarly the discussion of design representations for interaction design, with UAN as a good representation, does not provide much material to work with. At least it makes clear that existing representations for interaction design do not help with supporting conceptual design, because they are too detailed, too rigid, and too much oriented towards the user interface.

## 3.4 Multi-disciplinary design teams

Design of interactive systems is inherently an interdisciplinary project [Rhe95a, for example]. Many different skills are needed to develop an interactive system. This large number of skills implies that the development of an interactive system has to be a team effort. Some information on potential problems with such multi-disciplinary teams, and an exploration of the different roles present in such teams, are presented in this section.

### 3.4.1 Disciplines and their cultures

Kim explains clearly why interdisciplinary work can be very hard, and what can be done to alleviate these problems [Kim90]. He argues that different disciplines are like different cultures; one has to learn the language, traditions, and values of a discipline before it can be appreciated.

Kim argues that disciplines are formed as soon as people meet recurring situations. To survive these situations people give priorities to different things, based on their needs. These priorities become invisible when a discipline becomes a way of life, however, and that is where the trouble starts. When the

priorities are questioned or threatened, for instance by somebody in a different discipline, who has different implicit priorities, people perceive this as a threat, because when the priorities of *their* discipline are threatened, survival of the discipline is also threatened. Since people usually identify themselves with their discipline, the threats are perceived personally. Talking about these issues would help to alleviate this problem by making priorities of a discipline explicit, but because the priorities have become invisible, it *seems* there is nothing to talk about.

The solution obviously is to become aware of this process, to understand your own and other disciplines' approaches, and to work from there. Kim provides a lot of concrete steps towards becoming an interdisciplinary person, and for bridging the gaps between disciplines based on this assumption. Karat describes a development process where the differences between disciplines were effectively dealt with [KB91]. In this development process a 'war room' was used for all of the design activities. In this room, each wall signified a particular aspect of the design process, and all information related to an aspect was stuck on the relevant wall. This made the progressing design very tangible for all the design team members, and greatly contributed to overcoming differences between different cultures. In addition, the team is much more productive when working in this manner [TCKO00].

#### 3.4.2 The constituents of a design team

Who ends up in a design team is often determined by what people are available, the budget of the project, the skills needed, etc. Some suggestions for the different people who can be part of the design team and the roles they can take are explored in this section.

**The team leader** Based on experience with a number of educational software design projects, van der Mast suggests a manager for the whole project who is not an 'independent outsider' [vdM95a, vdM95b]. He states the manager should be able to make quality evaluations personally, and this is only possible with a suitable background of both the development process, and of the domain being developed for. Van der Mast compares this manager role with that of the director during film making.

This view is reinforced by Fred Brooks. He proposes the 'surgical team' approach where a team is fashioned after a surgical team [FPB95]. This approach recognizes that one man in the team is really doing the bulk of the work, and

all the others on the team are there to allow him to do so. In a real surgical team, the surgeon is the one leading and primarily carrying out an operation. The rest of the team make this possible by providing the right instruments at the right time, making sure the patient remains stable, etc.

**Team members** Depending on the needs of the project many different people can be involved in the design of an interactive system. In any case van der Mast suggests using skilled professionals for all tasks [vdM95a, vdM95b]. Although new tools may exist which can achieve similar results to those of a professional, professionals still are recommended. As an example consider the use of a screen painter for prototyping. While most people would be able to operate the screen painter, only a professional will achieve a satisfying and professional result.

Tognazzini thinks three roles in particular are very important in the design team: an HCI expert, a graphics designer, and a writer [Tog92].

**The HCI expert** is a logical choice, although it is not entirely clear what the tasks of this role are according to Tognazzini. Given his background, however, the HCI expert should be able to use the data collected during the analysis phase, and to make informed design decisions based upon them.

**The graphical designer** is chosen by Tognazzini because of his capabilities as a communicator, in particular for visual communication with the user with a minimum of expression.

**The writer of manuals** is an unexpected selection. Tognazzini argues that the writer is the one who has to explain the system to the users in writing. This gives him the best opportunity to gauge whether a specific part of the system is going to be very hard to understand, or whether a part of the system will cause confusion. The length of prose to describe a feature, and the number of exceptions to be noted in the text provide good clues for this.

### 3.4.3 Discussion

The explanation of Kim was also observed during the I-DIANA case study. The problems that were encountered with the descriptions of the conceptual design (See Section 2.5.2), are an illustration of the differences in priorities and



values between cultures. The textual description did not come easy to each team member; cultural differences also became apparent, for example, there were heated discussions between the user interface team member and the software engineer over the approach to follow and the potential impact of technical problems. In hindsight both team members were trying to defend their values and priorities, just as Kim describes.

The issues surrounding the team members are much less clear. Van der Mast's reasoning for an informed team leader seems logical. His analogy with the position of director in film making is valid. Tognazzini's most important roles in the design team are much less clear, and open to discussion. Importance does not seem to be a good criterion anyway, as it depends so much on one's point of view. His choice for the manual writer is surprising yet apt, but his choice for the graphic designer is less convincing, and seems to indicate a specific preference for graphically rich interfaces. In the context of this thesis this is not really relevant, as it does not focus on the early conceptual design.

What really is most surprising is how *little* has been said about the members of the design team. Most methods, for instance, do not indicate what role or roles in the design team they would like to support. Other methods silently assume 'a designer' or something similarly unspecified. Hardly anything has been published on the design team, and how it can be supported best.

## 3.5 Conclusions and requirements

The preceding sections provide an overview of the literature on HCI, and in particular on analysis and conceptual design. Taken together with the DIANA case described in Chapter 2, it is now clear that there are still many open issues that need to be addressed to get better support for the conceptual design of interactive systems. A number of these issues are described below.

It should be clear by now that what we are really looking for is a design tool. This is the main question underlying the research questions posed in Section 1.3. This design 'tool' need not be a piece of software; tool is used here in a more generic way. We are looking for something that provides support for the design of interactive systems facilitating ill-structured work. Starting with the open issues, it is explored what requirements there are for such a design tool.

### 3.5.1 The underlying paradigm

First, however, let's briefly revisit the paradigm mentioned in Section 1.3 on page 15. The literature cited in Section 3.1.2 confirms this paradigm, as can be seen with the quote from John Seely Brown and the levels of interaction as described by Buxton. Newman and Lamming also stress this, pointing out that the crucial property of an interactive system is that it provides support for human activity.

A similar conclusion was drawn in Section 2.5.2 in the overall evaluation on page 54. There it is concluded that a bad or unusable system can only be improved marginally by a good user interface.

Finally, Section 3.3.3 also supports the paradigm, although implicitly. By showing that a detailed description does not provide enough abstraction to discuss the conceptual design of an interactive system, it makes it clear that a view is needed that is more than just the sum of the individual user interface elements.

### 3.5.2 Open issues

**Design tools** Section 3.3.5 makes it clear that there is no design tool which supports the conceptual design well. The balance between rigidity and flexibility in particular seems to be a hard combination to make. This is discussed in more detail below on page 91.

The general discussion of design tools in Section 3.3.1 is useful in that it provides stimulating food for thought. It can not help directly in solving the specific problems associated with the conceptual design of interactive systems. At best this discussion provides directions to look in, and generic hints for designing design tools. This is also provided by the study into real-world design requirements presented on page 84.

**Dealing with context** Section 3.2.6 shows that there are some tools or theories that incorporate contextual information. Recent theories on analysis in particular all seem to focus on using contextual information in some way. This shows that it is important to include contextual information, but what is that exactly. This is less clear. In general it is information about the context of someone's work, but how this can be described, or which aspects are particularly important, remains unclear. Activity theory tries to answer these questions, but at this time it is not quite practical enough to base a whole design tool on. The issue is what contextual information to include in which way.

**Dealing with ill-structured work** Section 3.2.4 describes a task modeling approach which takes ill-structured work explicitly into account. This approach is not aimed at the design of interactive systems, and in fact it is seen more as an analysis tool for business processes than anything else. The issue here is how this explicit description of ill-structured work can be brought into a design tool for interactive systems.

**The precarious balance between rigidity and flexibility** The design tools discussed in Section 3.3.2 don't have a good balance between being rigid and being flexible, between being structured and being free. They either lean all the way to one side or to the other. The truth, as usual, lies somewhere in the middle.

The freedom is needed to anticipate on all of the complexity of the design, and the specific requirements this may bring. Early during the design in particular the designer needs freedom to express what is being found, even if it doesn't make sense yet. This advice also emerges from the trenches, especially when ill-structured work is considered.

The structure is needed to eventually make sense out of all of the design, and to slowly work towards something which can be implemented. When structured information is available it can be used by automatic tools, for instance to provide ergonomic advice or generated help text.

The issue is how to make a combination of these two which will support the design of interactive systems, instead of working against it.

### 3.5.3 Requirements and suggestions

With the paradigm mentioned in Section 3.5.1 as a guiding principle, and the open issues from Section 3.5.2 as potential areas to address when describing a new design tool, this section lists a number of requirements for such a design tool. Thus, it sets the stage for the next chapter where a theory based on these requirements will be described. From the whole discussion in Section 3.3 it is clear that a major part of this theory will be a design representation.

**Include artifacts and activities** If there is one thing that all the design tools described in Section 3.3.2 have in common, it must be the use of a combination of objects and functions to describe the design. While this is no guarantee that this is the best basis, it does provide strong support for it.

In Section 3.2.5 the argument is made to include artifacts into the design representation. Furthermore, one of the theories presented in the same section takes activity as its central tenet: “you are what you do”. This is reinforced in Section 3.1.2 to be crucial to the design of interactive systems.

Now it becomes clear why all these design tools are using objects and functions; using them as abstractions of the artifacts and activities in the real world. Clearly it is a requirement to include both artifacts and activities in one form or another in the design tool. Both theory and current practice agree on this.

**Include contextual information and support for ill-structured work** Why modeling of ill-structured work is important is explained in Section 3.2.2 when traditional task analysis methods are discussed. Section 3.1.4 explains that a model-based approach works best for dealing with ill-structured work. Thus the design tool should contain some kind of model or design representation. Most likely this will be similar to the one found in Section 3.2.4.

The need to include contextual information crops up in a number of places in this chapter. Section 3.1.2 introduces the importance of including the environment of an interactive system in the design process. In Section 3.2.2 it is argued that this is one of the main problems with the traditional task analysis systems, and Section 3.2.3 describes the example of TKS, in which task structures are enhanced by adding contextual information. An important bit of contextual information in TKS is on the artifacts that are used in conjunction with the tasks. Again artifacts and activities are brought together as described above.

In addition to artifacts, knowledge about less tangible things of the work environment need to be included as well. The recent developments discussed in Section 3.2.5 show that this kind of information is increasingly considered to be important input for the design activities.

The discussion of interaction design representations in Section 3.3.3 further reinforces the importance of including contextual information.

**Balance between structure and freedom** The balance between structure and freedom is a precarious one, in real life as much as for a design tool. It is clear that neither can win alone. The trick is to find the right balance, and this is still an open issue [Gla95]. There are actually several issues here, for all of which this balance is important.

Section 3.3.1 shows that there is a need for incremental design, which primarily stresses the freedom to add information to the design representation

when it becomes available. In Section 3.2.3 it is argued that it is important to add detail gradually as the design progresses. The design representation thus needs to take this into account: it will have to allow only a coarse outline in the beginning, while still allowing additional detail to be added over time. This need for incremental design is reinforced in Section 3.3.2 where it is made clear that the lack of support for incremental design is one of the major problems in the tools discussed. A specific example for architecture is shown in Section 3.3.1 where several layers with more and more detail are included in the design representation. Letting the design evolve in an organized manner is also called progressive design [GCR98].

Ambiguity and inconsistency are shown in Section 3.3.2 to be important in the design process. On first impression this seems to argue for freedom more than for structure, but this is not actually true. Having structure merely means that there is some kind of organization involved, which can for instance be used by automated tools to deal with. This organization could still leave room for ambiguity and inconsistency, in the same sense that humans can agree to disagree. In fact, when a design representation is structured, ambiguity and inconsistency can at least be detected automatically.

**Build on the analysis results** Section 3.2 makes it clear that task analysis, while having its problems when dealing with ill-structured work and contextual information, is still a very useful way to determine the work to be supported. The hierarchy of objectives and goals which shape the work are a powerful and well understood way of describing work. Additionally, Section 3.3.2 shows that all the tools discussed there base their design on a task hierarchy uncovered during analysis.

**Multi-disciplinary design teams** Section 3.4.2 suggests that it is important to be clear about responsibilities, priorities, and objectives of each design team member. Being specific about these issues when describing the design activities can alleviate potential problems in this area. Furthermore the design team members need to be able to identify with the design representation, as is made clear in Section 3.4.1.

**Design representation** First of all, it is important to make sure the design representation will be usable by the design team. Based on the observations in Section 3.3.2 it is interesting to see how unusable some of these tools can be.

If the design tool itself is not usable, then what hope is there for the systems designed with it?

The design representation can also cause some problems. One of these problems is that it can be very tempting to just describe current practice, instead of designing a new way to support work. Related to this is the possibility for the representation to limit the design space. Also, care needs to be taken to keep the design feasible.

There is also conflicting advice on selecting a proper representation. On the one hand a representation should be chosen which can be interpreted in terms of the final product, according to what is found in Section 3.3.3. On the other hand in the same section it is argued that interface details need to be kept abstracted at this point. Hopefully it will be possible to satisfy both issues at the same time.

## CHAPTER 4

---

### A Workspace-Oriented Design Representation

---

A new design tool called WONDER<sup>1</sup> is described in this chapter. WONDER provides support for the conceptual design of interactive systems. The central concept in WONDER, as the name indicates, is the workspace. This chapter starts with an informal introduction to this concept, based on an analogy. Then a more formal description of WONDER is given. This description is organized in several separate sections on the way of thinking, controlling, modeling, working, and supporting of WONDER. This separation is based on a framework for understanding a design tool. Finally, the requirements presented in Section 2.5 and Section 3.5 are revisited, and a set of assumptions are presented which will be checked against real-world use of WONDER in Chapter 6.

#### 4.1 WONDER's central concept: the workspace

The term 'workspace' denotes a place where a specific job can be done. An interactive system can be seen as a collection of such places put together in a meaningful way, to produce a set of spaces for work: workspaces. An early use

---

<sup>1</sup>WONDER is short for Workspace oriented design representation

of the term can be found in the ‘Rooms’ project [CAH87]. There, each Room is seen by the authors as a workspace. This view does not do much justice to the term, however, because in Rooms each room is empty, providing just the bare environment for carrying out work, leaving it up to the user to make each room useful.

In this thesis the term ‘workspace’ encompasses more than just a ‘physical’ location or the direct representation on the computer of such a location. It is seen as a whole, rich, environment aimed at supporting a particular job or set of jobs. The term ‘workspace’ is being used in a similar way in other literature [Mar96b, MPH<sup>+</sup>97]. Before getting into more specific descriptions of workspaces and their contents, it is illustrative to look at some real-world examples of workspaces. The workplace of a watchmaker provides an excellent introduction.

#### 4.1.1 The watchmaker’s workplace

The workplace of a watchmaker provides a good analogy with many interactive systems for several reasons. First, watchmakers carry out several different jobs as part of their work. This is the same for many information workers. Second, not all of their jobs are always straightforward, for instance, finding what the problem is with a broken watch is an ill-structured task. This is also the same for information workers, who often work on Type II work (See Section 3.1.4), which consists of ill-structured tasks. Third, watchmakers have many tools to their disposal. Again, this is the same for information workers, who’s interactive system provides them with many different ways of manipulating data.

Watchmakers have a head start on information workers for three reasons, though. They have been around much longer, and consequently have had more time to organize and optimize their workplace. Their object of work is hardware, forcing them to work within the laws of nature. They are also bound by physical space. This is an advantage in the sense that physical space is often easier to grasp and organize than the ‘virtual space’ within an interactive system. This latter point is also where the analogy breaks down: because of its virtual nature, an interactive system can provide a much more complex and varied organization of the workplace. It also has to *interface* with the real world context that remains, unlike the watchmaker’s workplace, where *everything* is placed in the real world. Still, it remains interesting to see how watchmakers organize their workplaces.

Figure 4.1 shows a typical workplace of a watchmaker [Dan81]. The first



#### 4.1. WONDER'S CENTRAL CONCEPT: THE WORKSPACE

---

impression is overwhelming, in particular because of the huge amount of specialized tools and raw materials required to repair watches. A second look reveals a very structured organization, however. The whole workplace is divided in smaller spaces, each of which has a specific purpose. Each of these work spaces is surrounded by the appropriate tools and materials needed for the specific job that that space supports. These individual spaces are all related to each other, and to their environment. Thus, contextual information plays an important part in the organization of the watchmaker's workplace.

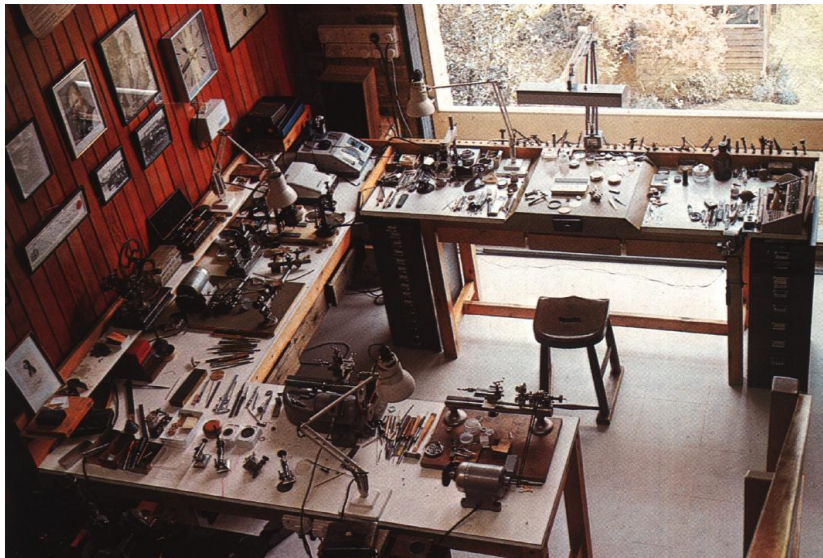


Figure 4.1: Example workplace of a watchmaker [Dan81]

Let's look at some specific examples of these workspaces. Consider for instance the workspace in the top of the picture, just below the window. This workspace, a finishing bench, is shown in more detail in Figure 4.2. The main objective associated with the finishing bench is the examination and assembly of watches. The raised platform in the center provides a demarcated area to examine the watches and spread out the parts. The rack behind this area contains a large number of small hand-tools, which are used specifically for work at the finishing bench. The available tools and the physical shape of the workspace provide ample means to reach the objective, without providing distractions.



Figure 4.2: Watchmaker workspace—finishing bench

In the bottom part of the overview picture another workspace is visible. This workspace consists of two lathes which are used to create parts for the watches. The storage drawers within the table top contain additional accessories for the lathes. The lamp can be used to provide light on the lathe where needed.

The wall shows a small and perhaps not easily detected workspace. The clock which hangs on the wall in the top left part of the overview is not just there to tell the time in general. It is also used to adjust the assembled watches, and to check whether their time-keeping is now accurate. The clock is tuned to a radio-controlled time signal, so that it will always keep accurate time. It is shown in more detail in Figure 4.4.

The contextual relationships of these workspaces are interesting. For instance, the finishing bench is located at the window, so that ample natural light will assist the to see properly when working with the watches. Moreover, the window is facing north, because indirect sunlight provides the best light for working with watches. The finishing bench is carefully located within the workplace. The lathes are located close by the finishing bench, because the watchmaker will need to change from one to the other occasionally. The current arrangement makes this possible easily. Finally, the location of the clock on the wall again shows the importance of context: its location allows the watchmaker to check the current time quickly from any location in the workplace,

#### 4.1. WONDER'S CENTRAL CONCEPT: THE WORKSPACE

---



Figure 4.3: Watchmaker workspace—lathes



Figure 4.4: Watchmaker workspace—calibrated clock

thus supporting the other activities. Still, its location also allows more detailed time examinations with the additional use of some specific timing machines located directly underneath the clock.

#### 4.1.2 Analysis of the watchmaker's workplace

The workplace organization of the watchmaker is in stark contrast with many interactive systems. Imagine, for instance, what a word processor would look like when moved to a physical space. For example, there would only be one place to work; a single document view is used for all word processing work. All the tools would be located in a circle around the open space, in several big piles. Some of the tools would be concealed from view as they are located in tool belts or cabinets. Only opening them will reveal what is inside.

This comparison is not completely fair. After all, word processing is not confined to the physical world. Even so, word processors usually only provide a single place to carry out the work: the document. Whatever the current objective of the user, it has to be reached by working in the same place. Surely the document is important, and creating it is normally the main objective when working with a word processor, but its generality makes it harder to reach some of the intermediate objectives. For example, the objective 'write new text' has a very different focus than the objective 'create good layout', and the availability of all the layout tools during the process of writing new text can prove to be very distracting.

The confinement of the watchmaker to the physical world has two properties that need not be transferred to the virtual world of an interactive system. First, the physical organization is governed by the laws of physics, which can appear to be alleviated in an interactive system. For instance, in interactive systems things can appear in more places than one, or look different depending on where they are, or change their proximity to each other depending on circumstances. Second, in an interactive system the organizational structure need not be revealed solely through 'physical' means such as separate windows or widgets. It is the logical organization which is important, not the visual manifestation thereof.

#### 4.1.3 The workspace as the basis of a design representation

The use of the workspace as a basis for the design representation allows the fulfillment of several of the requirements mentioned in Section 3.5.3. Each workspace is centered around an *objective*, which can be reached by carrying

out one or more, possible ill-structured, tasks. This satisfies the requirement to build on the analysis results by using the objectives found in a task analysis. The relations between objectives found in the task analysis, hierarchical or otherwise, can be used to define the relations between workspaces.

The objective is reached by manipulating the *materials* in the workspace using *tools* that are available there. The materials are a representation for the artifacts, and the tools provide a means for carrying out the activities, thus satisfying another requirement listed in Section 3.5.3 to include artifacts and activities in the design representation. Defining tools and materials in this way is similar to the use of these terms in the Tools&Materials Metaphor [BCNS92, RZ95].

## 4.2 Way of thinking

The way of thinking describes the philosophy behind WONDER, part of which has already been described in the previous section [Wij91]. The main idea is to use the ‘workspace’ concept to structure an interactive system more according to the different objectives that it needs to support, instead of just to the raw functionality it offers. The discourse should be related to what the user is doing at that time, and this should be reflected in the interactive system [Rei89]. These objectives can be determined during the analysis phase of the design process, for instance using task analysis. Apart from providing more structure in general, workspaces also much better facilitate support for ill-structured tasks, thus fulfilling several of the requirements listed in Section 3.5.3.

### 4.2.1 Foundations of the workspace concept

A number of ideas and concepts underly the workspace; they are listed in this section.

**Mental models** The use of workspaces to provide structure for an interactive system also ties in with the mental models users have of a system. There is no precise definition of a mental model, but Carroll describes it as:

“a representation (in the head) of a physical system or software being run on a computer, with some plausible cascade of causal associations connecting the input to the output.” [CO88]

That is, the mental model describes the image the user has of the system and its workings. It is important that this model is for the most part correct if the user wants to work efficiently with the system. If the system responds or acts in a way the user's mental model did not predict, then the user will be, at least temporarily, confused. This is why Newman and Lamming stress the importance of defining the 'intended mental model' as a first step during conceptual design [NL95]. Workspaces provide a good vantage point for creating such a mental model because there is so much correspondence between the work of the users and the organization of the system in workspaces.

**Hierarchies of objectives** An underlying assumption for WONDER is that the structure of an interactive system is best defined by hierarchies of objectives. Task modeling will produce these hierarchies with a focus on the tasks needed to reach the objectives. Once the focus has been shifted from the tasks to the objectives, and the objectives have been evaluated and cleaned up, the resulting hierarchies provide the skeleton for the interactive system.

The plural 'hierarchies' has been used so far because an interactive system need not be composed of just a single hierarchy of objectives. A single system may pursue different objectives partly using the same means, for instance, or a secondary objective may be instrumental to, but not directly related to, a primary objective.

**Artifacts** Artifacts have a special place in WONDER because they have some special properties in the workplace. First, they are the center of the objectives. Artifacts can be used to provide additional structure because they are normally common to several objectives. Second, the artifacts are almost always tied to the real world outside the interactive system, either because they are physically present in the workplace or because they are also needed outside of the system, for example a form which needs to be sent to another organization.

Both these properties cause the artifacts to be invariant for the design. This can help guide the design by providing some fixed boundaries to work with, thus constraining the design in a reasonable way. This also explains why the artifacts are described using fully independent representations in WONDER.

**Ill-structured tasks** Ill-structured tasks are described by collecting several types of information into a single 'task space': a description of a virtual space which contains all the tools and information needed to reach an objective. Such a task space description lists all necessary ingredients for the

task except the task itself, which, being ill-structured, can vary. This description adds to Bots' decision structures in that all possible operations are also listed. This is important for the design of interactive systems because it facilitates a better understanding of the interaction and of the possibility of carrying out tasks.

**Dealing with ambiguity and inconsistency** Ambiguous descriptions will always be present in the early conceptual design. Not all design decisions can be made immediately, for instance because not enough information is available for all decisions, or simply because not all decisions can be made at the same time. Some descriptions simply can not be resolved when they appear in the design, and they will have to remain vague until they can be. It is important that these vague and ambiguous descriptions are also part of the design representation, so that all design team members can see them easily. Otherwise they can easily be forgotten or be assumed to be solved. A good representation for ambiguous information is free text. As the design evolves, the ambiguity needs to be resolved as much as possible. A gradual transition to a more structured description, e.g. with a fixed format, or by using specific keywords, can be used to support this.

A similar case can be made for inconsistencies in the design. These, too, will appear in the early design. While a support tool can easily keep such inconsistencies from appearing in the first place, this would not prove very useful. Often inconsistencies alert the designer to different approaches to or possibilities for dealing with an issue, or they indicate that conflicting forces need to be resolved. In any case an inconsistency provides useful feedback into the design process, and such inconsistencies should be allowed in the design representation. As with the ambiguities, inconsistencies need to be resolved as the design progresses. Supporting tools can help this process by alerting the designer to inconsistencies, and provide reports on them if needed.

#### **4.2.2 The elements of WONDER.**

Based on the ideas described in Section 4.1 and Section 4.2.1, WONDER uses three concepts to describe the conceptual design: workspace, material, and action. These concepts are described in detail in Section 4.3, but their meaning is described here, along with some analogies into the watchmaker's workplace described in Section 4.1.1.



**Definition 1** A *workspace* is a place where getting a particular job or set of jobs done is supported.

Two examples of workspaces in the watchmaker's workplace are the lathes which support making small parts for the watches, and synchronized clock which supports timing the watches. Workspaces are directly tied to one or more objectives, and since they also have relationships with other workspaces, they will embody the *hierarchies of objectives* mentioned above.

**Definition 2** *Materials* are the data that needs to be changed or used to accomplish the goals.

Materials are the stuff that is being worked on. For instance, the watchmaker uses springs, levers and wheels as material. With interactive systems the material is always data. Often it is changed, but it can also be used as input to make decisions or to change other materials. Materials embody the *artifacts* mentioned above. While materials can be used in more than one workspace, they are independent of them. The same material can take a different appearance in different workspaces. What is an image to one workspace may just be a blob of data to be mailed out to another.

**Definition 3** *Actions* are generic operations that apply to more than one material, or that needs to be present in more than one workspace.

Defining actions separately from workspaces and materials helps to make sure that later on in the design similar things are done in similar ways. The watchmaker has a number of default actions which he applies to all his materials, such as picking up a material or putting it in a cupboard. Actions provide hints towards the actual interaction in workspaces which will be designed later, and they help to ensure *consistency* as the design progresses.

**Definition 4** A *tool* provides a means to inspect or change a material in the context of a particular workspace.

In addition to these three elements, there is a fourth one that also warrants a description here, even though it is formally defined as part of the workspace description. Tools are directly tied to particular workspaces. Tools arrange a set of actions and their own additional interactions within a workspace so that materials can be shown or changed. Tools provide all the means needed to carry out the *ill-structured tasks* mentioned above.



## 4.3 Way of modeling

This section contains a description of how workspaces are modeled. First types of models are discussed in order to select the right type of model to use, and then the same is done for the media used in the model. Finally, all three elements, *workspace*, *material*, and *action* are defined, keeping in mind the design representation pitfalls mentioned in Section 3.5.3 on page 93. A representation of the conceptual design can be created by describing these elements.

### 4.3.1 Types of models

The requirements for the structure and organization of the design representation are concerned primarily with the flexibility needed to support an iterative design activity that starts with uncertainty. The type of model used to describe the design greatly influences this aspect. Wijers recognizes four types of modeling representations with respect to information systems development [Wij91]. *Free models* provide no restrictions: anything goes, the only constraints are those imposed by the designers. Free models provide great flexibility and expressiveness, but can not be checked or supported in an automated way. *Structured models* are constrained by the types of concepts they use and the properties of these concepts. Typical examples are diagrams, tables, and structured text. Structured models can be checked for consistency in as far as this can be determined from the structure. *Mathematical models* are based on mathematical constructs, and are particularly useful for deducing system characteristics. Finally, *dynamic models* such as simulation models and prototypes offer experimentation facilities, and can be used to explore dynamic behavior.

Dix makes a strong case for formal representations of interaction [Dix95], which confirm to what Wijers calls mathematical models, based on the argument that a formal representation forces the designers to be more explicit, and that formal representations allow automated analysis of these representations. This allows a whole set of errors to be caught early in the design process. There is a lot of truth in this claim, but it does not apply to representations for conceptual design for two reasons.

First, being explicit within a formal framework takes a lot of time, and leaves no room for ambiguity. While this latter argument is very important later in the design process, it actually hinders the conceptual design process, simply because not everything is crystal clear right from the start.

Second, exactly which representation should be described formally? It was made clear in the previous chapter that just describing the interactions with

the interface is not useful for the conceptual design; even summarizing the interactions will not help, because this summary will not lead to useful building blocks, and because interaction at a conceptual level should really be designed top-down, cf. the discussion of UAN in Section 3.3.3.

Dynamic models are often used, mostly as prototypes. When they are used in interactive systems design or user interface design, dynamic models can take up a lot of resources, for example time or manpower, before they become useful. Creating a convincing visual appearance or programming convincing interaction requires a level of detail not usually attainable during conceptual design, or that is simply too expensive. In addition, dynamic models, often implemented as prototypes in interaction design, do not facilitate the automated checks that are possible with structured and mathematical models. This makes it hard to ensure consistency towards the end of the conceptual design.

This leaves free and structured models. It is tempting to rule out free models on the basis that they have no structure, can not be checked, and might be open to different interpretations. This leaves structured models, which seem to be most suited: they can be checked automatically, while still leaving some room for inconsistencies in the models. Structured models do have the drawback that they do not allow ambiguity. Options always have to be clear because of their rigidly defined structure, and while such options can be inconsistent with each other, they can not be ambiguous. A better solution seems to be a combination of both free models and structured models, with as much as possible a gradual transformation from ‘free’ to ‘structured’ information.

### 4.3.2 Media for models

The requirements in Section 3.5.3 relevant to the selection of media only show that the representation need not be graphical, and that it needs to be usable by the design team. Possible choices for media are either text, graphics, or both. Other media, such as audio or video, are not considered because they are too demanding on common hardware<sup>2</sup>, because their creation often involves much work which is not likely to pay off later in the design process, and because they are less easy to access.

Currently conceptual design is often supported by sketching a first impression of an interface, so pictures seem to have a natural advantage, also because interactive systems are often characterized by their graphical appearance. This

---

<sup>2</sup>When these decisions were taken in 1996.

advantage may be misleading, though. It has been argued above that structure, as specified by the goals and objectives, is more important than the way the interface will eventually look. Continuing that line of reasoning, it makes sense to focus on this structure first, and not be confined by one of a number of potentially suitable graphical representations. For this reason text is chosen as the media of choice for the representation of 'free' information in WONDER. The same choice is made for 'structured' information, although some graphical diagrams may also be included here.

### 4.3.3 WONDER elements

The three elements of WONDER are *workspaces*, *materials*, and *actions*. In this section each of them is described in detail, as are the relationships that exist between them.

**The workspace representation** Several groups of information, each addressing different aspects of a workspace, taken together make up the workspace representation. Table 4.1 shows the different groups and their elements.

The first group is associated with the position of the workspace in the overall system and with general information on the workspace. The *goal* is described here, in one or two succinct sentences. This defines the purpose of the workspace, and indicates what needs to be accomplished there. In addition to the goal, the *context* is also described. The context indicates the relation of the workspace to the whole job and to the work environment. Typical things described here include the importance of the goal and the reasons for this importance, the triggers or circumstances which can lead to this workspace, the specific circumstances under which the job in this workspace is carried out, etc.

The second group describes the structured relationships with other workspaces. Three types of relationships exist. The '*enclosed in*' relationship lists the workspaces that the current workspace is a part of. The '*contains*' relationship lists which additional workspaces the current workspace contains. Together these relationships sketch a hierarchy of workspaces. The third type of workspace relationship is the '*works with*' relationship. It indicates that the referred workspace works closely with the current workspace, for instance by providing common supporting tools, or because the jobs of the workspaces are related to each other but still distinct.

The third group describes the *materials* relevant to the workspace. Since these materials are invariant with respect to the workspaces, i.e. they remain

the same materials no matter in what workspaces they are presented, only simple references in the workspaces are needed to list the materials. More information is needed, though, even though the materials remain the same regardless of the workspace they are used in, the way they are used and needed can differ greatly between workspaces. For instance, only particular attributes of a material might be important, or the material need not be changed.

The fourth group discusses the *tools* and the way they interact with the *materials*. Since tools are unique to a workspace, they need to be described in detail here. They can not be described as separate elements. For each tool, the affected materials need to be listed, and the way in which each material is affected needs to be described. The effects of the tool shall be described in terms of existing actions as much as possible to allow a consistent interface to be designed later. This group also contains a brief textual description of the type of interaction best suited for this workspace.

The fifth group contains *housekeeping information*. This includes general remarks about, and the history of, the workspace. The remarks include 'to do' items and similar unresolved issues or warnings. They can be seen as a staging area for the next layer of detail to be added. The *versioning information* associated with the evolution of the workspace is very important. While in general only the most recent description of a workspace is used, it may be necessary to look at the changes made to the workspace over time, in particular when the changes to the design consist of important design decisions. Documenting both the reasons for these changes and the changes themselves will help the designer understand the conceptual design in later stages of the development process.

**The material representation** The definition of a *material* representation is given in Table 4.2. Four groups of information make up the representation.

The first group contains the *identification* and general description of the material, including a reference to its physical counterpart, if any.

The second group describes the *current use* of the material. If possible the description should be accompanied by one or more examples of the material as it is being presently used, e.g. through photographs, scans, or example screens. In addition to the description, contextual information also needs to be added, for instance whether this material is also used to communicate with people outside the scope of the interactive system.

The third group lists the *relationships* which materials can have. Some of these relationships are explicit in that one material can contain another material

<b>General information</b>		
Goal		Describes the goal which this workspace should support. The goal is described in terms of the worker.
Context		The context describes what the circumstances are for the use of this workspace, such as the trigger which led to this use.
<b>Workspace relationships</b>		
Encloses		Lists the workspace(s) of which this workspace is a part.
Contains		Lists the workspace(s) contained within this workspace.
Cooperates		Lists those workspace(s) which can be used to support tasks within this workspace.
<b>Materials</b>		
Materials		The materials the workers will use to reach their goals, and the context in which these materials will be used.
<b>Tools</b>		
Tools		The tools, operating on the materials, which the workers use to reach their goals.
Interaction		Brief textual description of the type of interaction which seems most suited.
<b>Housekeeping information</b>		
Remarks		Additional remarks about this workspace, such as possible pitfalls with particular design solutions, additional emphasis for aspects of the workspace, etc.
Version history		Log of changes to the workspace, including brief explanations of these changes.

Table 4.1: Workspace representation definition

as an attribute. These direct relations are captured as attributes. Relationships may also be more implicit, for instance when materials are indirectly related, or when strong contextual links exist between two materials even though this is not expressed explicitly within the interactive system. These relationships are called *associations*, and they are included so that the design team can be aware of them.

The fourth group consists of *housekeeping information*, and as with the workspaces, contains both room for remarks and versioning information.

<b>General information</b>	
Description	A description of the material, in particular how it corresponds with the real world, and its position in the work of people.
<b>Links with current artifact</b>	
Current use	Current use of this material, as found during analysis of work context. Note that information can also be used implicitly.
Example	Example of this material. Based on current use if possible.
<b>Relationships</b>	
Attributes	The attributes of the materials. Can refer to other materials.
Associations	Materials are often associated with other materials, for instance because there is an implicit relation between them.
<b>Housekeeping information</b>	
Remarks	Additional remarks about the material.
Version history	The revision history of this element including comments.

Table 4.2: Material representation definition

**The action representation** The action representation is kept as simple as possible. Its main purpose in the conceptual design is to ensure that similar operations will be implemented in a similar way across workspaces. The action representations can also be used to determine which workspaces contain similar actions. During conceptual design the most important function of the action representation is therefore to be a placeholder. During later design activities actual implementations will be proposed for each action.

The action representation is shown in Table 4.3. Since actions mostly serve as placeholders at this stage of the design, no specific details are given in the definition. Materials are not included in action representations, even though actions always operate on materials. This makes it easier to keep actions consistent across different materials. By specifying just a single ‘select’ action, for example, all selections can be made in a uniform way, regardless of the material being selected.

Goal	The goal of the action. What will it accomplish.
Extensions	A list of possible extensions for the action. An example of an extension is 'multiple selection' as an extension of the 'select' action.
Remarks	Additional remarks about the action. Mostly used to indicate hints and issues for later design activities.
Version history	The revision history of this element including comments.

Table 4.3: Action representation definition

**How workspaces, materials, and actions are combined** There is no overall structure in which the workspaces, materials, and actions are combined. Instead, these combinations follow from the individual representations of workspaces, materials, and actions. Figure 4.5 shows a diagram which clarifies these relationships. The diagram is consistent with Predicator Set Model (PSM) [tH93]. Appendix B contains a diagram explaining the model components of PSM. PSM was chosen for the model because it can deal easily with the relationships which can exist between different instances of the same type of element in WONDER.

Not having a pre-imposed structure allows the model to be inconsistent, because one workspace may contain one end of a two sided relation, while the other workspace referenced in the relation may not contain the other part. In this case, this lack of forced consistency is a virtue, and in fact a requirement as outlined earlier in Section 4.2. These inconsistencies, however, can and should be checked and noted during later design activities.

The set of material representations is independent from the workspace representations. While materials can point to each other through the associations, they do not point to workspaces. This is in accordance with the philosophy that the materials exist independently from the workspaces. The relationship between materials and workspaces exists by having the workspaces point to the materials.

The workspace representations do refer to the materials, and they also refer to other workspaces, forming combinations with them. The most common combination of workspaces is the hierarchy, where one workspace contains several smaller, more detailed workspaces, each of which can in turn also contain one or more workspaces. Such a hierarchy clearly resembles a goal or task hierarchy, where one top-level goal is split into several subgoals, and so on. It is important to note that several such hierarchies can exist at the same time if there are several main goals to be reached through the interactive system.

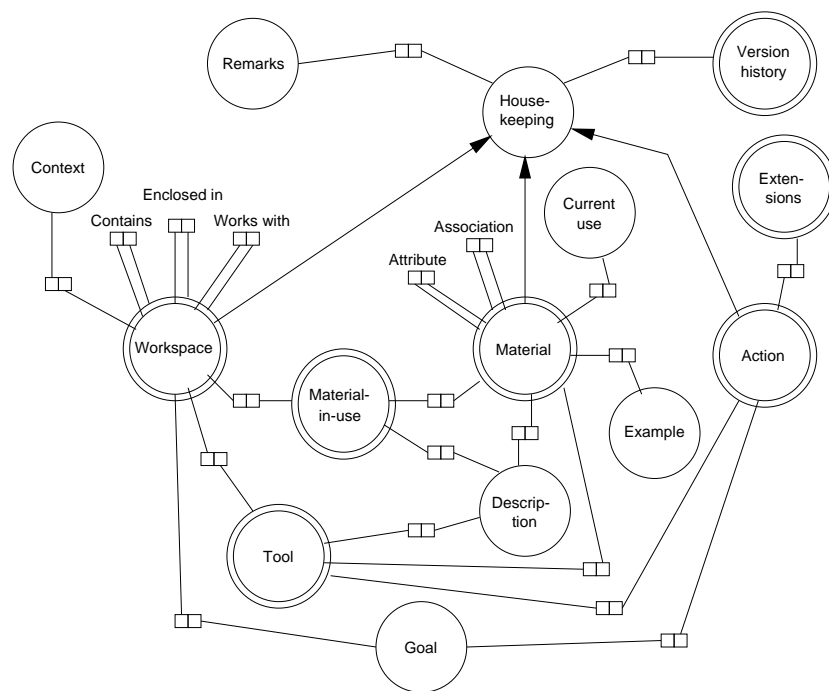


Figure 4.5: PSM diagram of workspaces, materials, and actions



Individual workspaces can also be linked together through the ‘works with’ relationship. These relations provide a second structuring in addition to the hierarchical structure. They can bring together workspaces from different hierarchies, or they can emphasize which workspaces are closely related within a hierarchy.

Descriptions of actions, like those of materials, are also independent from workspaces. Their only relationship with workspaces is through the tool descriptions within the workspaces.

**Evolution of the representations** The definitions of the different representations presented above only show the final state of each representation. It is important to note that the representations need not confirm to these final definitions from the start. This has to do with the requirements in Section 3.5 for a flexible representation which initially needs to allow ambiguity and inconsistency. A migration from free to structured information can support this by using free text initially to give a rough description, and then gradually move towards a more structured representation which is less ambiguous, and which can be cross-checked for consistency when needed. Table 4.4 shows which attributes of each concept remain free text, and which migrate towards a structured representation.

<b>workspace</b>		
free		goal, context, remarks, interaction
free → structured		materials, tools
structured		enclosed in, contains, works with, version history
<b>material</b>		
free		description, current use, example, remarks
free → structured		associations, attributes
structured		version history
<b>action</b>		
free		goal, extensions, remarks

Table 4.4: Structuredness of elements

## 4.4 Way of working

Creating the conceptual design for an interactive system is often seen as a creative activity which can not be easily controlled or written down: an ill-structured task. Ill-structured tasks can, however, be written down to some

extent, and it is important to do so, because doing so makes it clear what has to be done to create the workspace model, which decision have to be taken, and what criteria need to be used.

Three major activities can be recognized in the process of creating a workspace model. Figure 4.6 shows these activities. During the first activity workspaces are found in the data collected during analysis. During the second activity each workspace is assessed. The purpose of this second step is to evaluate whether each workspace still makes sense within the whole of the design. The third major activity refines the workspaces as much as possible. The other WONDER elements, materials and actions, follow from the workspace descriptions, and should be described during the refinement activity.

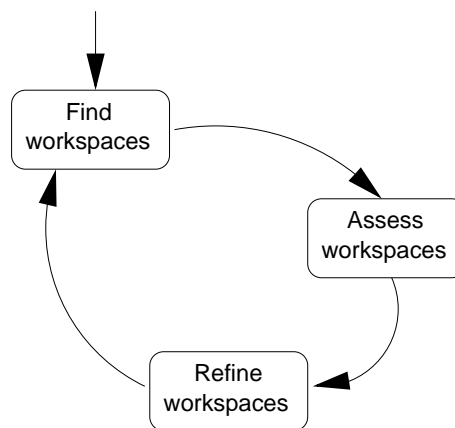


Figure 4.6: Task structure—creating a workspace model

Together these three activities make up the process of creating the workspace model. The order shown in the task structure should be taken with a pinch of salt. In practice it is likely that all three activities will continuously be carried out in no particular order, depending on what the design team encounters during design.

#### 4.4.1 Finding workspaces

Workspaces can be found in two ways. The initial set of workspaces is based on the goal-hierarchies found during analysis. Each goal in the hierarchy can be seen as a potential workspace. Just describing the workspace title, the associated objective, and some context, such as a trigger for the objective, is needed to turn a goal into a prospective workspace. The collection of workspaces created in this way will need to be evaluated and undergo further changes, but provides a good starting point for the design nonetheless.

Once an initial set of workspaces is present, additional workspaces can be found by examining current tasks and artifacts, and as a result of the assessment activity.

#### 4.4.2 Assessing workspaces

Given that many iterations are needed to create a conceptual design, each description will need to be assessed frequently to see whether it still fits in with the remainder of the design. During assessment a number of criteria are used to determine whether a potential workspace needs to remain one, or whether it can better be split, removed, or taken together with other workspaces. Figure 4.7 shows this process using a task structure. During assessment a potential workspace can be in three states: it is either too simple, too complex, or just about right, rather like the porridge [Ano].

**Too simple** If a workspace is too simple, than it is really not a workspace. This is the case when the workspace's objective can be achieved in a straightforward way, i.e., if the tasks within it are not ill-structured. Most likely it is a tool, or it should be bundled with several similar small workspaces. An exception can be made for very lengthy tasks with many distinct actions. In this case additional navigation support may be needed, and a workspace can be a more appropriate approach.

**Too complex** A workspace can be too complex when it tries to satisfy several goals at once. This will only work well if the goals are strongly related, or when the tools needed in the workspace are used for both goals. Otherwise it is better to break the workspace up into two separate workspaces, one for each goal. Even when a workspace only seems to support one single goal, it might be too complex. This condition will usually manifest itself by having two or

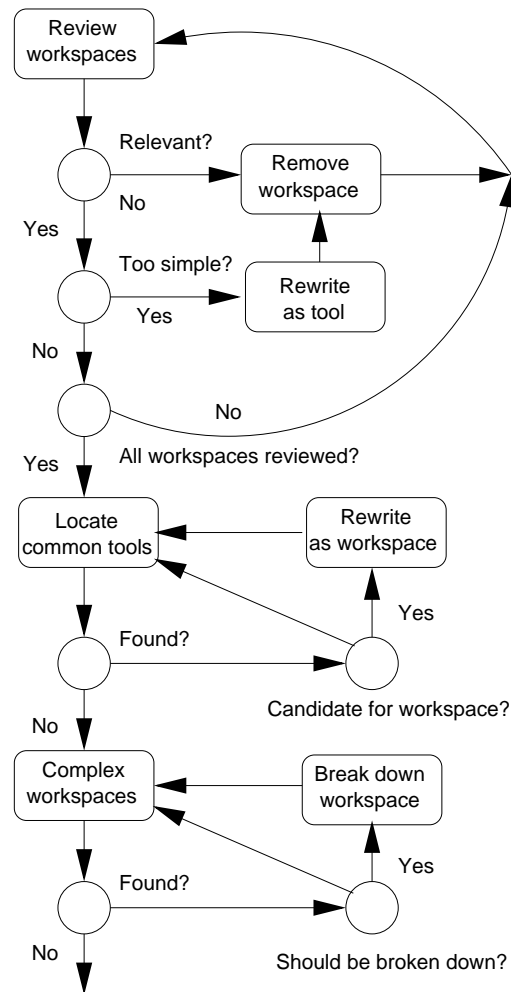


Figure 4.7: Task structure—Assessing workspaces

more separate groups of tools that are not strongly related to each other, and which support different sub-goals.

**New workspaces** Assessing existing workspaces can lead to completely new workspaces as well. This happens when common tools are found in several workspaces. If the same tool or set of tools is used in a number of workspaces, then it might be a good candidate for a new supportive workspace; to make this decision it is important to consider how the users regard these tools in each of the workspaces. If the tools are seen as an integral part of the tasks in each workspace, then it might be better to leave them as they are. Otherwise they can be separated out.

The final assessment criterion is one of completeness. The purpose of this criterion is to determine whether all workspaces have been found. The domain expert (See Table 4.5 and Section 4.5) is the only design team member who can really make this assessment. Some assistance can be provided by the results from the consistency checks, for instance because a workspace or material is referenced but not currently included in the descriptions.

#### 4.4.3 Refining workspaces

After a workspace has been assessed, it can be refined further. This refinement process serves two goals: it tries to make the description of each workspace as complete as possible, and it tries to ensure no errors or questions are left within a workspace description at the end of the conceptual design activity. The former goal will be most important during the first iterations, with the latter becoming more important towards the end of the conceptual design process.

Each workspace description starts out with only the most minimal of information, and then it is gradually expanded to a complete description following Table 4.4. While no particular expansion is mandatory, a natural order of expansion can be recognized, which is shown on the right-hand side in Figure 4.8. The left-hand side of Figure 4.8 shows a task structure for adding each of these description elements. The interaction between Domain Expert (DE) and Interaction Designer (ID) is clearly visible.

**Cleaning up workspace descriptions** Once a workspace description is starting to be complete, it needs to be ‘cleaned up’ regularly. As they evolve, the descriptions often accumulate questions, problems, and errors. This is good:

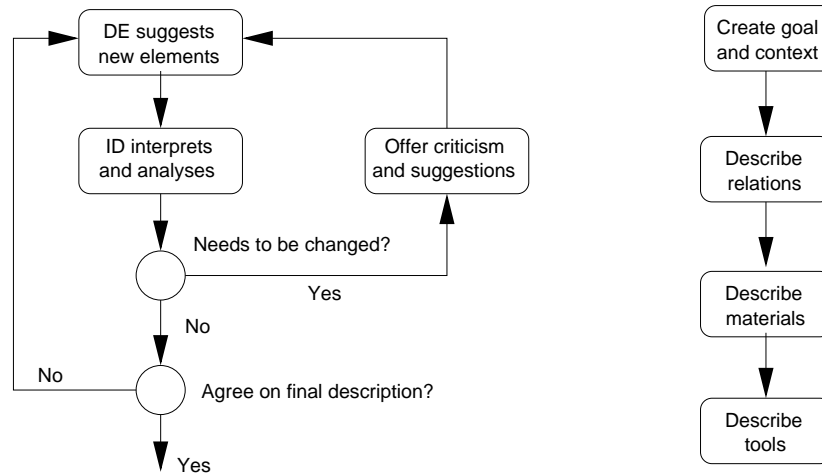


Figure 4.8: Refinement step 1: creating workspace descriptions

allowing the design team to express their problems and questions directly in the design representation keeps them visible, and will cause them to not be forgotten. It also helps the design team by allowing them to concentrate on the issues at hand, knowing that there is a record of those problems, questions, and errors. Eventually, however, these problems and errors will need to be resolved. This process usually involves decision taking and cross checking with other relevant parts of the design.

Sometimes this process can have dramatic effects. Whole workspaces may have become obsolete, or changes must be made in many workspaces. Such large and unsettling changes need to be made and supported. At this point in the design process making such sweeping changes is cheap. Once implementation has started it will be very hard to make similar changes without incurring significant cost.

This process of cleaning up the design is shown in Figure 4.9, which shows a cyclic process in which workspace descriptions and their relations are scrutinized time and again. The act of scrutinizing implies a critical look, and this is exactly what is needed at this time. Anything which is not clearly specified, that is represented by vague descriptions, needs to be investigated further, and

to be made more explicit and more concrete if possible. Open questions, remarks, suggestions, and errors noted in each workspace should be treated in a similar manner. The relations amongst workspaces and the connection between workspaces and materials should be scrutinized in a similar vein. Inconsistencies should be noted and if possible resolved.

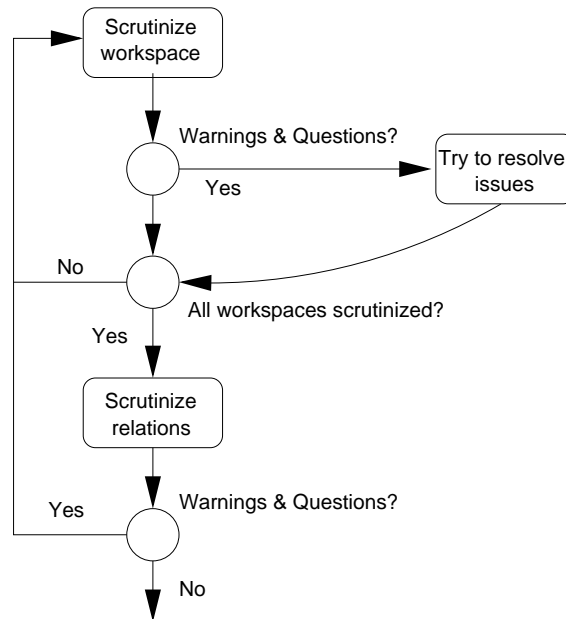


Figure 4.9: Refinement step 2: cleaning up a workspace

## 4.5 Way of controlling

The way of controlling deals with the management of the development process based on the way of modeling and way of working presented earlier, including planning and plan evaluation and regular project management aspects including quality assurance [Wij91]. Most of these issues are not particular for the use of WONDER. Regular project management, for instance, is needed in any project, and generic project management literature can be used [BB92, for example]. Van Meel argues that for design projects an adaptive control strategy is

useful [vM94]. Such a control strategy takes the design project as an adaptive process of learning which needs to be incremental. This strategy ties in with one of the requirements listed in Section 3.5.3.

In this section only those aspects of the way of controlling which are particular to WONDER are described. In fact, the main difference when working with WONDER is with the people. The need for, and unique properties of, a multi-disciplinary team are made clear in Section 3.4. In addition, quality assurance is an interesting topic. Who will have the final say in the quality of the design?

### 4.5.1 The design team

Section 3.4 makes it clear that interactive systems can only be designed by multi-disciplinary design teams, and it also describes that such teams have their own set of issues to be dealt with. One of the conclusions repeated in Section 3.5.3 on page 93 is that it is very important to be clear about the responsibilities of each of the design team members.

The selection of roles presented in Table 4.5 does not correspond directly to the findings presented in Section 3.4.2. This is because the few recommendations presented in that section are aimed primarily at designing and creating the *user interface*. Creating the conceptual design requires a different set of skills. The set of roles for WONDER is based on an inventory of skills needed and on experiences from the case study presented in Chapter 2.

**Overview of the design team** Table 4.5 contains an overview of the six roles that are important when creating a workspace based design. It is important to realize that these roles need to be divided amongst the team members, and that a good fit might not always be possible. The exact allocation will always depend on the team at hand, but it is important to make clear beforehand who gets assigned which roles, and consequently, which responsibilities. The domain expert and the interaction designer are the two key roles. Both roles require a lot of work, and it is their interactions which makes the design process work. This explains why it is important that these interactions are visible to everyone involved, which is what happens when the WONDER representations are used to record the results of these interactions. For this reason it is strongly advised not to assign their roles to a single person, even if this person is qualified for both roles. Combining the roles would lead to a lot of implicit communication within one persons head. This would make the design process more visible, and also more prone to errors because there is no need to be explicit



Interaction designer	<p>Coordinates design efforts of all team members. Designs for and guards the usability of the design</p> <ul style="list-style-type: none"><li>• creating design descriptions</li><li>• overall design of the interactive system</li></ul>
Domain expert	<p>Guide the creation of the conceptual design. Tries to translate knowledge of the work into design descriptions by communicating this knowledge to the interaction designer.</p> <ul style="list-style-type: none"><li>• design domain concepts, functionality, and support</li><li>• assess evolving design</li></ul>
Project manager	<p>Coordinates the project. Makes sure that facilities and people are available, monitors the progress of the project. Keeps track of administrative issues.</p> <ul style="list-style-type: none"><li>• finish the project in time and on budget</li></ul>
Visual designer	<p>Given the design descriptions, the visual designer tries to find proper visualizations for those concepts.</p> <ul style="list-style-type: none"><li>• assess the evolving design with respect to visualization and the visual interface</li></ul>
Software engineer	<p>Given the design descriptions, the software engineer tries to identify potential implementation problems, and has to assess the feasibility of the design concepts.</p> <ul style="list-style-type: none"><li>• assess the evolving design with respect to software engineering principles</li></ul>
Participating users	<p>The users give continuous feedback on the design as it evolves. They can also add to the design, for instance by making suggestions based on solutions they are currently using or would like to use.</p> <ul style="list-style-type: none"><li>• assess the evolving design with respect to their work</li></ul>

---

---

Table 4.5: Roles in the design team

and thus there is much less of a confrontation with the design ideas. This appears to be less of a problem with the other roles, as long as the team member is aware of the different sets of responsibilities.

**Roles in the design team** The *domain expert* has two major duties. First, they need to find and refine the workspaces by analyzing the collected information, and by making additional judgments based on their knowledge of the domain. Their efforts should primarily focus on the structure and completeness of the design. Second, they need to assess the evolving design with respect to the domain. Examples of this assessment include whether all goals can be satisfied with the current design, and whether the current design is in accordance with the view of the users on their work.

The major duty of the *interaction designer* is the creation of the actual design description of the workspaces and materials. There are two reasons for having the interaction designer do this. First, the interaction designer has more experience with these descriptions, in particular when it comes to the more detailed descriptions, and the way in which they will eventually evolve into actual user interface specifications. Second, by separating discovery and definition into two separate roles, the designers are forced to make the design descriptions, ideas, and assumptions explicit. This greatly reduces the possibility of misunderstandings and discrepancies within the design.

The role of the *project manager* is a supporting role. The main task of the project manager in WONDER is to facilitate the design team in accomplishing their design on time and within budget. It is important to stress that the project manager is not the team leader, but rather an administrative and managerial aide to complete the project successfully.

The *visual designer* and the *software engineer* are similar roles, and neither can make significant contributions to the design at this point. This is true for the visual designer who can not start with any visual design, because this is explicitly not a part of WONDER. It is true for the software engineer, who also cannot contribute to the design. The reason they are part of the design team despite not being able to contribute directly, is that they can evaluate the workspace descriptions from the point of view of their own expertise. Even if they cannot contribute now, they will still have to extend the WONDER design later in the development process, so that their early evaluation can be used to find bad solutions early on, and correct them while it is still cheap to do so.

Finally, the *participating users* provide a similar role; they will eventually do their work with the system, and their feedback on working through their

daily tasks given the workspaces can provide useful feedback to the design team. This feedback is only useful when the workspaces have been somewhat finalized, and will be particularly useful in combination with scenarios and simple prototypes. These users can be seen as informants providing various contributions at times relevant to the design process [SRAD97].

**Who leads the team** As mentioned above the project manager is not considered to be the team leader. There are two reasons behind this view: the project manager is seen as a supportive facilitator in WONDER, and it is assumed that the project manager does not have the ability or knowledge to make the qualitative assessments that guide the project. Perhaps a better term for this role would be a facilitator or administrator [FPB95]. The arguments behind not having an independent outsider be team leader have been discussed in Section 3.4.2.

This leaves the domain expert and the interaction designer as viable candidates for the position of team leader. While the interaction designer has the advantage of knowing WONDER and its associated activities well, there are two compelling reasons for not making the interaction designer the team leader. First, the stake of the interaction designer is always smaller than that of the domain expert, for whose area of expertise the system is being developed. Second, a lack of knowledge about the domain will make it much harder for the interaction designer to make qualitative judgment calls. Yet quality control is an important aspect of the job that needs to be done carefully, and by someone who can judge the quality not just of the overall work, but also for the situation it is going to be used in. This leaves the *domain expert* as the best candidate for being team leader.

## 4.6 Way of supporting

The creation of a conceptual design using workspaces can benefit greatly from good support. In fact, some kind of computing support is almost mandatory, as will be explained in this section. A research instrument providing this kind of support is described in more detail in Chapter 5. The remainder of this section contains an explanation of the need for computer assistance, and a list of requirements for such assistance.

**The need for computer assistance** The ways of modeling and working described in Sections 4.3 and 4.4 do not contain explicit references to a need for computer assistance. Yet, in practice working with WONDER representations in a meaningful way requires computer assistance. The requirements listed in Section 3.5.3 explain this. There it is asserted in the discussion on freedom vs. structure on page 92 that the use of a structured design representation allows ambiguity and consistency to be detected at the least. As is shown in Table 4.4, several of the elements in WONDER representations are structured, thus allowing these checks.

These checks can and should be automated. For instance, automatically creating overviews of current problems and issues helps to make visible what still needs to be done. The same holds for automatically generated lists of inconsistencies. For instance, by comparing how workspaces refer to each other, inconsistencies in the design can be noted. Some examples of these overviews are shown in Section 5.2.4. While it is possible to do these checks manually, this does not make sense, because the time spent on doing this can not be used for real design work, and because the work is quite tedious, which will likely result in it not being done as often or careful as it should be.

**Additional benefits of computer assistance** There are a number of additional benefits that can be reaped from the use of computer assistance. For instance, it will facilitate browsing through the design in a much more varied way than can be achieved when the representations are organized in a single way, for instance as a stack of paper in a folder. As an example of this browsing, consider the use of the many connections between the representations as a means to move from one to another.

There are several overviews which can be generated from the collection of workspace and material representations. These overviews, showing for example which materials occur in which workspaces, can provide additional insight into the complete design, and as such are very useful. Automating the creation of these overviews makes them available all the time without additional effort from the design team.

Computing support also makes it easy to maintain a useful history of the design. If people remember to add notes saying what was changed and why, they often do not include all the small changes that they made, for instance because some changes are not deemed important at the time. Having automated version control which prompts for an explanation of the changes made, and records all of these changes in detail, alleviates this problem.

## 4.7 The design process following WONDER

The use of WONDER provides only the start for the design process. Using it results in a conceptual design, but then much additional work needs to be done before the interactive system can be implemented in the work environment. In this section several popular design techniques are discussed that can be used in conjunction with WONDER.

**Scenario-based design** In scenario-based design a scenario, i.e., “a narrative description of what people do and experience as they try to make use of computer systems and applications” [Car95, p. 3], is used to guide the design activity pro-actively [Car94, Car96]. In this context a scenario is used as it is in film making, to denote a script that will be followed. In scenario-based design a number of these scripts are created, each of them describing a particular work situation and how this situation is played out, for example a letter coming in. Then, all of these scripts are played out using a particular design or idea to see what the implications of the design are for the different aspects of the work as described by the scenarios. Note that this is different from another use of the word ‘scenario’, where for instance the use of different types of systems would be called the ‘scenarios’.

WONDER provides a good starting place for scenario-based design, because it brings several of the requirements for scenarios together in a single place. The focus of scenarios matches that of workspaces fairly well, because both are centered on work-related objectives. Therefore, workspaces form a natural starting point for scenarios. Furthermore, the presence of tools and materials in the workspace descriptions allows for additional benefits for the use of scenarios. Their presence makes it easy to verify the correctness of a workspace, while at the same time allowing exploration of how these materials and tools can be represented best using user interface components, given normal work situations.

**Prototyping** The workspaces also provide a good starting point for a variety of prototyping activities. The structure of the workspaces makes it easier to tackle individual parts of the interactive system. The presence of materials and tools in the workspace descriptions serves as a checklist of those things that need to be represented in the prototypes or sketches. Two design approaches are particularly suited. Prototyping can be used in combination with scenario based design to provide a rich exploration of the system. Prototyping can also

be used in the sense of Seaton and Stewart (See Section 3.3.2), where workspace after workspace is added to a working system installed in the workplace.

**Design Space Analysis** Finally, Design Space Analysis (DSA) is also a useful technique to further the design into an implementation. DSA provides a structured way to explore different alternatives for a particular design problem, and for judging the trade offs that need to be made [MYBM91, MBYM91]. DSA works best for specific and focused decisions, where certain alternatives need to be considered [vACM95]. This explains why using workspaces as a start works well for DSA. Once a workspace is defined, there are many specific choices to be made, for example “what kind of interaction is needed for this tool?” or “what kind of representation is needed for this material?”. DSA can help to ensure that all options are evaluated and the more optimal design solution is chosen.

## 4.8 Reflections on WONDER

WONDER has now been described. This section contains a look back to the requirements and suggestions formulated in the previous chapter. Next, a number of assumptions are stated which can be used to test WONDER in a real-world case. The section concludes with some general observations on WONDER.

### 4.8.1 A look back at the requirements

Section 3.5.3 contains a number of requirements and suggestions based on the experiences in the case study presented in Chapter 2 and the literature presented in Chapter 3. In this section it is shown how these requirements and suggestions have been addressed in WONDER. Table 4.6 on page 128 provides an overview.

**Include artifacts and activities** Activities and artifacts are a common theme in all the design tools described in Section 3.3.2. WONDER includes both as described in Section 4.2: tools as a representation for activities and materials as a representation for artifacts. Also, these items are amongst the elements of WONDER’s representations described in Section 4.3.3.

**Include contextual information** All recent developments include contextual information in one way or the other, as described in Section 3.2.5. WONDER includes some contextual information in its representations. This is particularly clear in the workspace representation, where there is an explicit field for contextual information. In addition, contextual information is also encoded in the goal of each workspace, and in the relations between the workspaces. Furthermore, in the material workspace the current use of each material is described, and an example of its use is given.

**Include support for ill-structured work** The importance of support for ill-structured work is demonstrated in Section 3.2.2. In Section 3.1.4 it is argued that a model-based approach works best for this purpose, and indeed WONDER is model-based. The approach WONDER takes for dealing with ill-structured work is based on the ideas underlying the method described in Section 3.2.4. This idea is to describe the task hierarchy top-down as best as one can, and then instead of trying to zoom in further on ill-structured parts, describe the actions and the information needed for them. WONDER implements this by describing the high-level task hierarchy of workspaces, and by describing the materials and tools within each workspace as a boundary for the ill-structured work.

**Balance between structure and freedom** Whether or not WONDER implements the right balance between structure and freedom can not be said without testing it. Two issues regarding this balance are described on page 92.

The way of working described in Section 4.4 clearly describes the *iterative approach* that is encoded in WONDER. There is explicit support for starting with a very rough sketch of the design and then adding more and more detail as it becomes available.

WONDER also incorporates *ambiguity and inconsistency* where appropriate. Table 4.4 shows how each type of representation gradually moves from a free to a more structured description where appropriate. The way of supporting makes explicit that inconsistencies should be allowed present in the design. Supporting tools can show them, and thus provide guidance towards resolving them.

**Build on the analysis results** Section 3.5.3 refers mostly to the task analysis results obtained during the analysis phase of a project. These results provide the input for the initial workspace hierarchies, as described in Section 4.4.1.

**Multi-disciplinary design teams** The way of controlling described in Section 4.5 lists the design team members and their roles and responsibilities. Where appropriate explicit references to these roles are also indicated in the diagrams that are part of the way of working, for example in Figure 4.8. In addition, on page 93 it is also noted that each design team member needs to identify with the design representation. This has not been fully achieved. In the discussion about the media of the model in Section 4.3.2 it has already become clear that not all design team members will adopt the current representations without hesitation. How well this really works will need to be tested.

**Design representation** The major requirement for the design representation is to be usable. While this has been one of the goals when constructing WONDER, it is not possible to tell whether this has succeeded without testing it.

Requirement or suggestion	In WONDER?
Include artifacts and activities	yes
Include contextual information	yes
Include support for ill-structured work	yes
Balance between freedom and structuredness	yes, needs testing
Build on the analysis results	yes
Support multi-disciplinary teams	yes, needs testing
Usable design representation	unclear, needs testing

Table 4.6: Requirements and suggestions implemented in WONDER

#### 4.8.2 Assumptions on the use of WONDER

In the previous section it is shown how WONDER addresses the requirements and suggestions posed in Section 3.5.3. While this indicates that WONDER is on the right track in solving the research questions posed in Section 1.3, using WONDER on an actual case will show whether all the required parts fit together into a workable whole that yields the results it is intended to.

Chapter 6 contains the description of using WONDER on such a real-world case. In order to facilitate a proper evaluation of this case, this section poses a number of assumptions about WONDER. Table 4.7 on the next page shows these assumptions.



*Way of thinking*

- 1 The model described with workspaces conforms to the mental model of the users towards their work.
- 2 Each workspace allows the users to accomplish the goals associated with the workspace.
- 3 The description of ill-structured work matches the perception of that work by both the domain expert and participating users.
- 4 Explicitly allowing ambiguity and consistency in the design process is beneficial.
- 5 The WONDER representations consist of the right ingredients: workspace, material, action, tool.

*Way of modeling*

- 6 Text is a good medium for a model during the early design phase.
- 7 The evolution from free to structured information as indicated in Table 4.4 works well.
- 8 No crucial information related to the early design process is kept outside of WONDER.

*Way of working*

- 9 The design activities described in Section 4.4 are a correct representation of the actual use of WONDER.
- 10 The different design team members can all work with WONDER.

*Way of controlling*

- 11 The roles defined in Table 4.5 match actual use of WONDER.
- 12 The domain expert shall be the team leader.

*Way of supporting*

- 13 Computer assistance is needed to work efficiently and effectively with WONDER.

*Other assumptions*

- 14 The time taken to use WONDER is worth the results it yields.
- 15 Compared to tools currently in use a WONDER workspace description provides at least the same level of support.

Table 4.7: Assumptions on the use of WONDER

### 4.8.3 Observations about WONDER

Before WONDER is put to the test in practice in Chapter 6, reflecting on WONDER yields some observations. For example, WONDER will likely work best on large projects. In those projects the strategies for dealing with the work and its associated decisions are not always clear. WONDER does not transfer well to small problems, because these can often more easily be designed without specific support, simply because there are less loose ends and design possibilities. A large problem implies a large design space, and much more structuring and organizing is needed. This means that the design team can less easily keep track of all the design issues, and hence a design tool like WONDER will be a useful tool in making the design explicit.

Using WONDER is just one step in the whole development process. This process is a complex activity which requires the cooperation of people in several disciplines, and which is interlaced with additional analysis and design activities. The use of WONDER cannot be separated from these other activities, even though it is described here in relative isolation.

Finally, compared with the design methods described in Section 3.3, WONDER brings together some of the best things of these methods. It will be interesting to see how this combination works out in practice.

## CHAPTER 5

---

### Computer support for workspace design

---

It was made clear in the previous chapter that there is a need for computer assistance when working with WONDER, the workspace oriented design representation. In this chapter an example of such a system is presented and evaluated. Recommendations for future implementations are given at the end of the chapter.

#### 5.1 Introduction

Section 4.6, in which the way of supporting WONDER is described, contains five reasons for the need of computer support with WONDER. In summary, these reasons are:

- manual maintenance of descriptions is often too tedious;
- manual version control does not work well;
- creating and maintaining relations between elements is too much work;
- creating overviews manually is hard and takes too much time;

- checking for inconsistencies is hard and it is easy to make mistakes with it.

### 5.1.1 Main areas of computer support for WONDER

These reasons clearly show that computer support is needed to work with WONDER effectively. Three main areas of support are needed: editing, browsing, and reporting.

**Editing** The functionality needed to edit workspace descriptions needs to be focused primarily on ease of use. This is an important requirement because editing needs to be done by *all* team members, and because editing should not consume much time, instead leaving more time for design itself.

The functionality for editing needs to facilitate easy retrieval of the right workspace description. Once the proper description is found, editing needs to be straightforward. Besides text editing facilities, functionality is needed to easily identify and insert components of a workspace, and to mark specific parts of each component as needed. Examples of such markers include warning, which signify that the current description may cause problems later on in the design process, and questions, which indicate that not enough information is currently available to complete part of the description. Each change in a workspace description needs to be documented, so that a history of changes can be perused later. Mandatory entering of explanations for these changes needs to be facilitated, but should also be as unobtrusive as possible. Finally, results from editing should be available to all team members quickly to facilitate iterative design.

**Browsing** Browsing the current state of workspace descriptions is an important activity which probably is used most often when working with WONDER. Functionality for browsing falls into two categories. First, navigation between workspaces needs to be supported in several ways, such as through an alphabetical list, and through hierarchies mimicking the workspace structure. Navigation from one workspace description to another also needs to be supported, for instance through references between workspaces. Second, browsing a given workspace description needs to be supported. In particular, it needs to be easy to read the description and its history. Finding specific components needs to be facilitated.

**Reporting** In addition to browsing, functionality for creating reports is also needed. The generation of automatic reports compliments browsing by bringing issues to attention which can not easily be detected by browsing manually through individual descriptions. Each of these reports needs to focus on a particular aspect of WONDER descriptions and their relations. Specific examples of such reports include:

- Overview of all available descriptions. This overview can serve as a table of contents to all descriptions. Optionally it can include versioning information.
- Overview of workspace relationships. This overview shows all the relationships workspaces have with each other through the 'contains', 'enclosed in', and 'works with' attributes. Inconsistencies in these relations can also be shown.
- Overview of workspaces per material. This overview shows in which workspaces each material is used.
- A list of warnings and questions marked in the descriptions, including enough context of the description itself to quickly assess the warning or question.

### 5.1.2 Requirements

The reasons for providing computer assistance mentioned above lead to the following requirements for computer assistance:

- It should be easy to maintain for all design team members. In particular this implies that no specific markup commands should need to be used, apart from those denoting the structure of the representation. All markup, references, and overviews need to be generated automatically from this description.
- Allow for sufficient markup and reference capabilities. Even though the team members should not be bothered by specific markup, the descriptions should still contain sufficient markup capabilities to show clearly the structure and different types of information. This will facilitate comprehension and quick scanning of the descriptions.

- Recent results should be available to all design team members. The design representation can only be useful when the most recent version is always available to all team members.
- Provide automatic provision for version control. Version control needs to be automatic. Team members should be prompted for their reasons behind any change, and their explanations should be recorded along with the changes automatically.

## 5.2 Computer support for WONDER

An example implementation for supporting working with WONDER is presented in this section. This implementation has been created as a research instrument to help carry out the case presented in Chapter 6. First, selections for tools and platforms are made based on the three categories of requirements presented above, followed by an overview of the design. Then, the example implementation is shown, as are some examples of it in use and a brief set of instructions.

### 5.2.1 Selection of tools and platforms

**Browsing** The combination of markup facilities and hypertext support suggests use of a World Wide Web (www) browser, such as Netscape's Navigator or Microsoft's Internet Explorer. The HyperText Markup Language (HTML) that is used to display information on the www contains facilities for both markup and hyperlinks [Rag97]. It is also widely used, which makes it easier for the design team members to navigate with it through the descriptions. Use of the www also allows easy access to the descriptions from any of the team members' workplaces, regardless of platform and location.

**Editing** The choice for HTML as the presentation medium unfortunately goes against some of the requirements for editing. Editing HTML codes is not easy, in particular when a non-trivial structure, for example using nested tables, needs to be created. This can be alleviated somewhat by providing templates, but will remain tedious. Creating all the hyperlinks is even more tedious for a casual user who suddenly needs to know details such as file names and directory structure. Apart from these issues, editing HTML simply takes too much time

away from the design process. These issues imply that either another presentation mechanism is required, or that editing needs to be done in some other format, which can be converted automatically into HTML. The latter approach was taken with the prototype. A very simple text-based description was defined, which can easily be converted into HTML automatically. An example of this description is shown in Figure 5.2.

Version control was implemented using a set of standard Unix' tools collectively known as the Revision Control System (RCS) [Tic85]. These tools allow an unlimited number of revisions to be kept, easy retrieval of any version or the differences between two versions, and they automatically calculate the differences between versions. Furthermore, RCS allows comments to be added to each version, and will prompt for this information whenever a new version is created.

**Reporting** The system is implemented in Perl, which allows easy prototyping and has excellent capabilities for text manipulation [WCS96]. In addition to the overviews listed in Section 5.1.2, the conversion from simple text to HTML is also implemented in this way. Only one command is needed to bring all descriptions and overviews up to date after making changes to the text-based description files.

### 5.2.2 Design

The design of the research instrument is kept as simple as possible, focussing on flexibility and enabling rapid prototyping. It is fully based on the technologies described in the previous section.

**Makefile** The design is centered around a simple one-to-one translation of source files into HTML files. An example of a source file is shown in Figure 5.2; the corresponding HTML is shown in Figure 5.3. A standard Unix Makefile drives this process, using rules to run the appropriate translation scripts. An excerpt of the Makefile showing the rules to translate workspace and material descriptions is shown in Figure 5.1.

**Translation scripts** The translation scripts, written in Perl, are straightforward. The input file is read and parsed, and then a HTML page is written based on the parsed information. In addition to the translation itself, some links to other information are also created, and the current version control status is also

```
%.html: $(SRC)/%.workspace $(BIN)/workspace2html
        $(BIN)/workspace2html $(SRC)/$*.workspace

%.html: $(SRC)/%.material $(BIN)/material2html
        $(BIN)/material2html $(SRC)/$*.material

workspace-hierarchy.html: inventory.w $(BIN)/workspace-hierarchy
        $(BIN)/workspace-hierarchy -s $(SRC)
```

Figure 5.1: Makefile (excerpt)

included. All the input files are parsed by the same software module, so that consistency between for instance workspace and material source files can be ensured.

The translation scripts also maintain several inventories; for instance for materials or tools. These inventories are updated whenever new information is found for them. For example, the file `inventory.m` contains the associations to other materials in a material description. These updated inventory files will trigger a rule in the Makefile which is shown at the bottom of Figure 5.1. A separate script is then run to update the summaries and overviews based on these inventories. Examples of these are shown in Section 5.2.4.

### 5.2.3 Examples of presentation and editing

To further clarify the use of simple text descriptions and a HTML based presentation, this section contains some annotated examples. Note that the actual content of the examples is not relevant here.

**Text based descriptions** Figure 5.2 contains an example of a text based description for a WONDER workspace. Issues such as layout and order of the items in the file are not important, because they will automatically be made consistent during the conversion process.

The attributes of each description type are identified by keywords. A keyword is preceded by a dot to distinguish it from normal text. Some of the fields contain unstructured text, which will be reproduced ‘as is’ in the HTML presentation of the workspace. Examples of these fields shown in the figure



```
.name Insert activity

.goal

    Insert a new activity into the current plan, with respect to the
    constraints of both the activities and the plan. Try to keep the
    changes in the plan to a minimum.

.enclosed_in small-changes

.contains

.works_with delete-activity precedence-manipulation
             activity-manipulation

.remarks

    This workspace can only deal with insertion of a few activities at
    the most. A new project should be inserted using a planning engine.

.materials

    - m_activity

      A new activity to be inserted (has just been created)

    - m_activity, m_precedence, m_resource, m_skill

      The existing activities and their precedence relations which
      constrain the insertion of a new activity. In particular the
      attributes which allocate resources and skills to activities.

.tools

    - Create activity

      a_create: a new m_activity. a_edit-attributes: of the new activity
      The new activity will not be part of the plan after creation, but
      needs to be inserted separately.

    - Find room in plan

      Support finding room for a new m_activity. Assign a m_resource to
      the activity (based on the associated m_skill). Try to place new
      activity at promising location in the current m_view
```

Figure 5.2: Text based workspace description (excerpt)

are *goal*, *context*, and *remarks*. The three fields which define the relationships with other workspaces are examples of structured fields. Each of these fields contains a list of relevant workspaces, as indicated by their internal name. If no relevant workspaces are needed, then the field is simply left blank.

The list of *materials* used in this workspace is not a simple structured field, because additional information on the use of the material is needed in this workspace. Materials are always itemized with a line starting with a single dash. After the dash one or more materials are listed, followed by some contextual information. Note that a material can occur more than once if it is used in different ways within the same workspace. For example, the workspace shown in Figure 5.2 makes a distinction between an activity which is going to be inserted in a plan versus the activities already present in that plan.

Each *material* is referenced by its internal name, and preceded by a special code indicating it is a material. This explicit encoding allows better consistency checking, and keeps the different namespaces for all WONDER elements separated. Each material also has a brief description of its use in the workspace, and an indication of the amount of items which will normally be present in the workspace.

Tools are described in a similar itemized way. The unstructured description of each *tool* contains references to actions which can be used to implement (part of) the tool, and to the materials it can operate on. It also describes the purpose and possibilities of each tool.

**HTML presentation** Figure 5.3 contains an example of the workspace description from Figure 5.2 as it is presented using HTML. In this presentation the workspace description is shown in an ordered and consistent fashion. Different parts of the description are separated by horizontal lines. The order in which items are presented is fixed, and independent from the order in the text files. The formatting of the presentation in tables makes it easy to locate all information on the page. Hyperlinks to other descriptions are underlined, but also have an indicator for the type of description they point to, for example a 'W' for a workspace description. This is particularly helpful for the tool descriptions, where material and action links are both used in the same text.

#### 5.2.4 Examples of overviews

Several overviews are generated automatically after changes have been made to the individual descriptions. The purpose of these overviews is twofold. One,

Go to: [Table of Contents](#), [Summaries](#), [Workspace Hierarchy](#).

---

### Workspace: Insert activity

<b>Goal</b>	Insert a new activity into the current plan, with respect to the constraints of both the activities and the plan. Try to keep the changes in the plan to a minimum.	
<b>Context</b>	Due to changes in a project or on the yard, small changes sometimes occur. As part of these small changes, new activities might have to be inserted in the current plan.	
<b>Enclosed in</b>	<a href="#">Small changes W</a>	
<b>Contains</b>	(none)	
<b>Works with</b>	<a href="#">Activity manipulation W</a> <a href="#">Delete activity W</a> <a href="#">Precedence maintenance W</a>	
<b>Materials</b>	<a href="#">Activity M</a>	A new activity to be inserted (has just been created)
	<a href="#">Activity M, Precedence relation M, Resource M, Skill M</a>	The existing activities and their precedence relations which constrain the insertion of a new activity. In particular the attributes which allocate resources and skills to activities.
	<a href="#">Plan scope M, Plan version M</a>	A (optional) scope and version in which the new activity can be inserted.
<hr/>		
<b>Interaction</b>	Interaction within this workspace must be based on direct manipulation, because this will give direct feedback on the suitability of a particular location. Immediate feedback is also needed when room is being created by moving activities around.	
<b>Scenarios</b>	<a href="#">Design Alternatives</a>	
<b>Tools</b>	<b>Create activity</b>	<a href="#">Create A</a> : a new <a href="#">Activity M</a> . <a href="#">Edit Attributes A</a> : of the new activity. The new activity will not be part of the plan after creation, but needs to be inserted separately.
	<b>Find room in plan</b>	Support finding room for a new <a href="#">Activity M</a> . Assign a <a href="#">Resource M</a> to the activity (based on the associated <a href="#">Skill M</a> ). Try to place new activity at promising location in the current <a href="#">View M</a> .
	<b>Move activities in plan</b>	Support moving an activities in the current plan around to create room for a new activity. <a href="#">Select A</a> : an existing <a href="#">Activity M</a> . Move it around the <a href="#">Plan scope M</a> and <a href="#">Plan version M</a> , and get feedback on its new location.
	<b>Insert activity</b>	Support the insertion of a new activity in the plan version. Assumes enough space is available. Assign <a href="#">Resource M</a> to the new <a href="#">Activity M</a> , and add <a href="#">Precedence relation M</a> .
<hr/>		
<b>Remarks</b>	This workspace can only deal with insertion of a few activities at the most. A new project should be inserted using a planning engine.	

Figure 5.3: HTML presentation of workspace description

they allow easy access to the individual descriptions, and two, by rearranging and summarizing information they make clear additional structure and consequences which are not immediately apparent from the descriptions themselves.

**Overview of all available descriptions** This overview, shown in Figure 5.4, primarily shows all available workspace, material, and action descriptions. It only provides a simple alphabetically ordered list for each of these categories. Some additional information can be presented in the list because of the use of versioning tools. This makes it possible to show both the current version number and date of last change; also indicated is whether a description is currently being worked on. The names of the descriptions are links to the description page, while the version information provides a link to the version history of the description.

#### Workspaces

<a href="#">Activity manipulation</a>	<a href="#">1.11, 3 months ago</a>	<a href="#">Design Alternatives</a>
<a href="#">Insert activity</a>	<a href="#">1.11, 6 weeks ago</a>	<a href="#">Design Alternatives</a>
<a href="#">Maintain plan</a>	<a href="#">1.10, 6 weeks ago</a>	
<a href="#">Maintain scopes</a>	<a href="#">1.11, 2 weeks ago</a>	
<a href="#">Plan validation</a>	<a href="#">1.9, 3 weeks ago</a>	<a href="#">Design Alternatives</a>
<a href="#">Progress monitoring</a>	<a href="#">1.7, today</a>	<a href="#">Design Alternatives</a>
<a href="#">Maintain views</a>	<a href="#">1.11, 3 weeks ago</a>	

#### Materials

<a href="#">Activity</a>	<a href="#">1.10, today</a>	
<a href="#">Actual Data</a>	<a href="#">1.3, 3 months ago</a>	<a href="#">Design Alternatives</a>
<a href="#">Capacity Allocation</a>	In Progress <a href="#">1.1, 3 months ago</a>	
<a href="#">Precedence relation</a>	<a href="#">1.6, 6 weeks ago</a>	
<a href="#">Plan problem</a>	<a href="#">1.6, 3 weeks ago</a>	<a href="#">Design Alternatives</a>
<a href="#">Plan scope</a>	<a href="#">1.7, 2 weeks ago</a>	<a href="#">Design Alternatives</a>
<a href="#">Skill</a>	<a href="#">1.5, 3 months ago</a>	<a href="#">Design Alternatives</a>
<a href="#">Plan version</a>	<a href="#">1.3, 3 months ago</a>	<a href="#">Design Alternatives</a>

#### Actions

<a href="#">Create</a>	<a href="#">1.2, 4 months ago</a>
<a href="#">Delete</a>	<a href="#">1.2, 4 months ago</a>
<a href="#">Select</a>	<a href="#">1.3, 3 months ago</a>
<a href="#">Start</a>	<a href="#">1.1, 4 months ago</a>

Figure 5.4: List of all available descriptions (excerpt)

**Overviews of workspace relations** These overviews shows the relations between workspaces in several ways. These relations need not be consistent, because they are defined through the individual descriptions. For instance, one workspace may list that it contains another workspace, while this other workspace does not indicate that it is contained in the first. Such inconsistencies will be visualized in these overviews.

In the first overview, all relations indicated in each workspace are summarized in a single table. An excerpt of this table is shown in Figure 5.5.

#### Inter-workspace relations

Workspace	Enclosed in	Contains	Works with
<a href="#">Maintain plan W</a>		<a href="#">Check plan W</a> <a href="#">Fix problems W</a> <a href="#">Insert project in plan W</a> <a href="#">Progress monitoring W</a> <a href="#">Small changes W</a>	<a href="#">Maintain scopes W</a> <a href="#">Maintain plan versions W</a> <a href="#">Maintain views W</a>
<a href="#">Progress monitoring W</a>	<a href="#">Maintain plan W</a>		<a href="#">Plan validation W</a> <a href="#">Solution finding W</a> <a href="#">Maintain views W</a>
<a href="#">Maintain views W</a>	<a href="#">Maintain plan W</a>		<a href="#">Plan validation W</a>
<a href="#">Status information W</a>	<a href="#">Maintain plan W</a>		<a href="#">Check plan W</a> <a href="#">Fix problems W</a> <a href="#">Plan validation W</a>
<a href="#">Maintain plan versions W</a>	<a href="#">Maintain plan W</a>		<a href="#">Maintain scopes W</a>
<a href="#">Delete activity W</a>	<a href="#">Small changes W</a>		<a href="#">Activity manipulation W</a> <a href="#">Insert activity W</a> <a href="#">Precedence maintenance W</a>
<a href="#">Insert activity W</a>	<a href="#">Small changes W</a>		<a href="#">Activity manipulation W</a> <a href="#">Delete activity W</a> <a href="#">Precedence maintenance W</a>

Figure 5.5: Table of workspace relations (excerpt)

A second overview, shown in Figure 5.6, shows all workspaces using their hierarchical relations. Several hierarchies can co-exist because there need not be a single root workspace. To create the hierarchies, workspaces which do not claim to be enclosed in other workspaces are collected. Each of these workspaces is considered to be the root workspace of a hierarchy. Then, for each of these root workspaces, a hierarchy is created by following the *contained* field in each workspace. Workspaces which claim to be enclosed, but are not in fact

mentioned in any *contained* field are shown in a separate list as orphans.

## Hierarchical workspace relations

Top-level workspace: [Maintain plan](#)

- [Maintain plan](#)
  - [Check plan W](#)
    - [Plan validation W](#)
    - [Status information W](#)
  - [Fix problems W](#)
    - [Planning engine W](#)
    - [Solution finding W](#)
  - [Insert project in plan W](#)
    - [Planning engine W](#)
  - [Progress monitoring W](#)
  - [Small changes W](#)
    - [Activity manipulation W](#)
    - [Delete activity W](#)
    - [Insert activity W](#)
    - [Precedence maintenance W](#)

Top-level workspace: [Solve capacity problems](#)

- [Solve capacity problems](#)

Figure 5.6: Workspace hierarchy (excerpt)

A third overview, shown in Figure 5.7, shows the other relations between workspaces. These relations are indicated in the *works with* relationship field. There are two possible situations. First, if both workspaces indicate that they work with each other, then they are considered to have an ‘exchange’ relationship. This usually indicates a strong symbioses between the two workspaces. Second, if only one of the two workspaces indicates the *works with* relationship, then it is considered a ‘depend’ relationship. This usually indicates that a workspace provides a service to one or more other workspaces. Both types of relationships are shown in Figure 5.7.

**Overview of workspaces per material** The final overview, shown in Figure 5.8, shows workspaces per material. This overview connects the *material* and workspace descriptions. All workspaces in which the material occurs are listed for each material. Creating this overview accomplishes two goals. One,

### Exchange relations

[Activity manipulation W](#) <-> [Delete activity W](#)  
[Activity manipulation W](#) <-> [Insert activity W](#)  
[Activity manipulation W](#) <-> [Precedence maintenance W](#)  
[Delete activity W](#) <-> [Insert activity W](#)  
[Delete activity W](#) <-> [Precedence maintenance W](#)  
[Delete activity W](#) <-> [Activity manipulation W](#)

### Depend relations

[Status information W](#) --> [Check plan W](#)  
[Plan validation W](#) --> [Fix problems W](#)  
[Planning engine W](#) --> [Fix problems W](#)  
[Planning engine W](#) --> [Insert project in plan W](#)  
[Planning engine W](#) --> [Maintain scopes W](#)  
[Small changes W](#) --> [Maintain scopes W](#)  
[Fix problems W](#) --> [Maintain scopes W](#)  
[Progress monitoring W](#) --> [Plan validation W](#)  
[Maintain views W](#) --> [Plan validation W](#)  
[Status information W](#) --> [Plan validation W](#)  
[Solution finding W](#) --> [Plan validation W](#)  
[Solution finding W](#) --> [Planning engine W](#)

Figure 5.7: Workspaces per material (excerpt)

it shows which materials do not appear in any workspace. This can happen when a material is initially created to serve as a workspace, but later removed from the workspace because of a changed view on that part of the design. Two, it can point to similarities between different workspaces, because of the use of the same (set of) workspaces.

**Workspaces per material**

<u>Skill M</u>	<a href="#">Insert activity W</a> <a href="#">Plan validation W</a> <a href="#">Solve capacity problems W</a>
<u>Resource M</u>	<a href="#">Insert activity W</a> <a href="#">Solve capacity problems W</a>
<u>Deliverable M</u>	<i>(Does not appear in any workspace)</i>
<u>Activity M</u>	<a href="#">Status information W</a> <a href="#">Delete activity W</a> <a href="#">Insert activity W</a> <a href="#">Precedence maintenance W</a> <a href="#">Activity manipulation W</a> <a href="#">Solve capacity problems W</a>

Figure 5.8: Workspaces per material (excerpt)

**5.2.5 Search facilities**

Searching through the descriptions in a flexible way provides a good means for finding information in the design descriptions. A simple search facility is included in example implementation, and shown in Figure 5.9. The search mechanism allows searching by keywords, and presents a list of all matching descriptions. The matched keywords are turned into links pointing at the relevant places of the descriptions. The results for an example query are also shown in Figure 5.9.

Two pre-defined searches are available. By convention warnings and issues are marked with a special notation in the descriptions. Warnings are marked with !!, and indicate potential problems or pitfalls with the current design solution or its future implementation. Issues are marked with ??, and indicate design issues that have not been resolved yet. The use of these two default searches makes it easy to locate those parts of the design which need further attention. The results are shown in a manner similar to those in Figure 5.9.



### Directory Text Search

You may repeat your search with a new regular expression. Searches are not case sensitive.

Search term:

---

These are the matches for the regular expression **'tolerance'**.

- **Title:** [Workspace: Progress monitoring](#)  
**Matching lines:**
  - The parameters used for checking progress: [tolerance](#) for detecting
  - deviances; [tolerance](#) for resolving deviances by manipulation.

Figure 5.9: Search facility including search results

## 5.3 Evaluation and recommendations

The system described in the previous section has been created as a research instrument only. There are a number of obvious improvements, some of which are outlined in this section. First, however, a brief evaluation of the use of this implementation is described.

### 5.3.1 Evaluation of use

As described in Section 5.2.1, the components of the system were specifically selected because of an ability to satisfy some of the requirements. Therefore it may seem obvious that these requirements, such as having sufficient markup capabilities or the need for having version control, have been fulfilled. This is not automatically true for all requirements, though, and in particular ease of maintenance had its problems.

The simple text format described in the previous section and shown in Figure 5.2 was not difficult to maintain, but still required too much low level knowledge about the format and its underlying assumptions. For example, the convention to prefix each reference to another description with a letter indicating its type and an underscore is not immediately intuitive, and is easily forgotten. Matters like this inhibited other design team members from making their own changes directly: requests for changes were always communicated to the team member who developed the research instrument.

The simple text format is not only to blame for a lack of maintainability; the

fact that the source of the descriptions, i.e. the simple text files, and the destination, i.e., the HTML pages, were separated also made maintenance harder. When something which needs changing is found in a description, a number of non-trivial things need to be done for the changes to be made. First, the set of simple text files needs to be found. Then the correct file needs to be identified. Next, a text editor needs to be started with the text file, and only then can the changes be made.

In fact, the above description leaves out the use of version control, which further complicates the maintenance process. The use of RCS as the version control software greatly facilitates keeping any number of versions and annotated differences between versions. The drawback of its use is that it is not integrated into the editing part of the prototype, which means that a team member who makes a change to a description will need to invoke the RCS tools manually to ensure that the changes are properly recorded. This puts an additional burden on editing and is easily forgotten or 'postponed' when changes need to be made quickly.

Finally, the current editing structure caused one other problem. The use of simple text files made it convenient to use the name of each file as an identifier for the descriptions. Other descriptions could then use this file name as a reference which is translated into a hypertext link in the HTML presentation of the description. This practice did not cause any problems until descriptions needed to change their names to better reflect their actual purpose, causing massive changes throughout all descriptions which referred to the original description. This can easily inhibit making such changes, especially when working under time pressure. As a result, additional confusion will occur later when names do not always correctly identify descriptions.

### 5.3.2 Recommendations

Most of the problems with maintainability mentioned above can be alleviated by integrating both presentation and editing into a single environment. Given the good results with presentation and availability, use of WWW pages is a good choice of platform for this integration.

Instead of using a simple text format separated from the presentation, items can be added directly through the web pages, for instance using HTML forms. Such an approach would also facilitate the stronger integration of version control into the editing system. An example screenshot of a system which could provide this integration is shown in Figure 5.10.

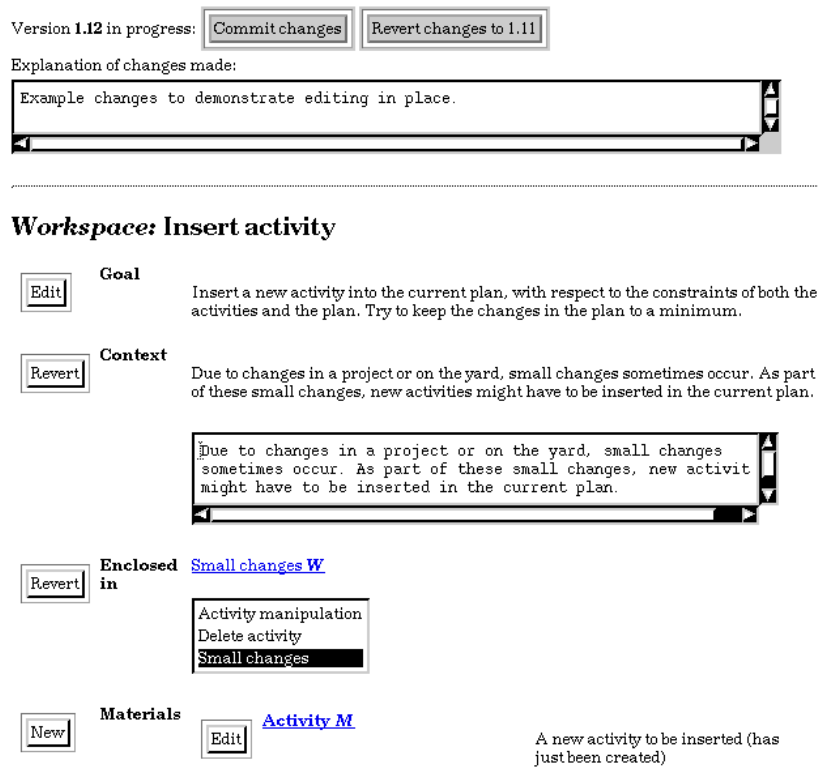


Figure 5.10: Example of integrated editing system

In this example a new version is currently being edited. The top widgets allow the current set of changes to be committed, or to revert to the previous version. A text box can be used for the comments on the changes that were made for this version. Each individual item can be edited with the 'Edit' button in front of it. When this button is pressed, the corresponding item changes. The button changes to 'Revert', allowing a safe way back to the earlier version. An input widget also appears, its shape depending on the data being edited. As an example the *context* and *enclosed in* fields are being edited in the screenshot.

Providing such an integrated environment will facilitate editing of the design representation by all team members, and will enhance maintainability.

## CHAPTER 6

---

### Use of WONDER in a design process

---

This chapter contains a qualitative evaluation of WONDER when applied to a real-world case: a shipyard planning system called SHIPSHAPE<sup>1</sup>. Both the process of using WONDER and the product resulting from it are included in the evaluation. Finally, the assumptions brought forward in Chapter 4 are discussed in light of the results from the case.

#### **6.1 SHIPSHAPE case description**

This section contains an introduction to shipyard planning in general and the specific issues in the SHIPSHAPE case. This is followed by a discussion of the suitability of this case to a qualitative evaluation of WONDER.

##### **6.1.1 Shipyard planning in general**

Designing and building ships is a very complex business which requires a great deal of planning. At a typically sized yard five ships may be constructed simul-

---

<sup>1</sup>SHIPSHAPE is short for the Shipyard Shared Planning Environment

taneously, with each ship featuring at least 600 distinct high-level construction and engineering activities. Organizing this vast amount of work is bound by many constraints such as the available resources and their cost, dealing with bottlenecks such as the ramp, the building process of a ship, and the delivery of crucial parts such as the engines. The planners on a shipyard need to advise whether or not new ships can be built, and they need to monitor progress of the projects and manage available and requested capacity.

The key to planning in general is to schedule the activities in such a way that all the constraints are met. Most often these constraints have to do with the capacity of resources. These resources include the work force, but also elements like the ramp, floor space and use of equipment such as cranes. In shipyard planning work is often subcontracted to avoid critical problems. Subcontracting is expensive, though, and often inflexible. This makes it a less desirable solution. By properly managing capacity, optimal use can be made of the workers at the yard, thus saving money and retaining flexibility.

This emphasis on optimal use of the workers also indicates the need for progress monitoring. Unexpected delays or problems are easily encountered and can have severe consequences for the capacity planning. Closely monitoring progress of each project can help to predict and avoid many of these problems, but this is not current practice because progress monitoring is not well supported with current tools.

### 6.1.2 SHIPSHAPE issues

The goal of shipyard planning is to support the main process in a shipyard: the design, engineering, and construction of ships. To accomplish this the production control department of a yard controls this process by giving directions to the other departments of the yard. This control is complicated by a number of problems which will occur: internal deadlines are missed; pressure mounts near the end of the project to get everything done in time; material or information is not available at the right time; and capacity allocation is erratic.

The issue here is that all of these problems change the initial plan. The key is not to disallow or ignore changes to the plan, but to facilitate making changes to the plans all the time. Making all the decisions on the yard explicit in the plans and then coordinating the activities in the yard based on the plans will help to create more reliable plans, and thus a more reliable production process. The next section explains how this was reflected in the design created with WONDER.

Another issue is that current planning tools are not well suited to shipyard planning. This is evident in two ways. First, current planning tools don't recognize the special characteristics of some of the resources in a shipyard. The best example of this is floor space. Not only do some activities need floor space, but they also need particular spaces depending on other resources such as cranes or on other parts of the ship under construction. Changing the order in which parts of a ship are assembled can have significant impact on the use of floor space. Second, current planning tools don't recognize the different goals which planners need to accomplish, and the fact that different tools are needed to support each of those goals.

### **6.1.3 A good case for applying WONDER**

A system for shipyard planning provides a good case for using WONDER. Shipyard planning is a good example of ill-structured work. There are no standard recipes for solving planning problems. The many dependencies and relations within a plan can easily cause a small change in one part of the plan to have significant effects in other parts. In addition, planning is also based on value judgments of the planners. Not all criteria and constraints are easily quantified, and judging which solution is best given the current situation of the whole yard is typically the responsibility of the planners. The issues mentioned in the previous section provide good challenges for WONDER.

Furthermore, a planning system is far from a small toy system. It is large and complex enough to allow several different but interconnected workspaces to surface in the design.

Finally, each yard already has some kind of support for the planning process, ranging from a plan board on the wall to a collection of computer tools. It will be interesting to see how WONDER compares to these tools. Given the issues and WONDER's aim, the design should result in more focused workspaces.

## **6.2 SHIPSHAPE's design in WONDER**

This section contains an overview of the WONDER design for the SHIPSHAPE case, listing the workspaces and materials and focusing on an example of joint evolution of a workspace and material throughout the design. It is an example of real-world use of WONDER and it serves as background information for the remainder of this chapter. The actual design representations for workspaces and materials can be found in Appendix C.

### 6.2.1 Describing a WONDER design

Describing the WONDER design of the SHIPSHAPE case in the confines of this thesis is not easy. Two important aspects of WONDER conspire to make this difficult: the hyperlinks between representations and the evolution of the design over time.

**Representing hyperlinks** Hyperlinks link all of the WONDER representations and several overviews together. Section 4.6 argues the need for computing support to cope with this in an effective way. The example implementation shown in Chapter 5 makes clear that a good and inclusive overview of all the relations and dependencies can only be formed by browsing and exploring the design. In Sections 6.2.2 and 6.2.3 some of these relations will be shown using figures and tables.

**Describing time** There is no finished SHIPSHAPE design made in WONDER. Instead the design grows within the WONDER representations initially, then gradually moves to other design activities and artifacts as described in Section 4.7. This is not a discrete process but rather a evolutionary move from just a set of goals to a more developed description and eventually an implemented system. As a result it is hard to present *the* SHIPSHAPE design in WONDER.

To explain this further, a schematic time line of the design process is shown in Figure 6.1. At the start of the design process diagrams are created as discussed in Section 6.3.1. Next, WONDER representations are used to create the initial design. As a separate effort a data model for shipyard planning was developed. Bringing this model together with the WONDER representations resulted in a number of scenarios, followed by a prototype implementing them. Meanwhile, the WONDER representations were kept somewhat up to date. Not being the only focus of the design team anymore, maintenance slowly petered out over some time, with the WONDER representations being updated and referred to less and less as the design was fleshed out more and more in the other design tools.

**Selecting a point in time** In the remainder of this section the SHIPSHAPE design will be described as it was at the time of the bold circle shown in Figure 6.1, just after creating the WONDER representations. At this point in time the design team felt it could move on to additional steps in the design process



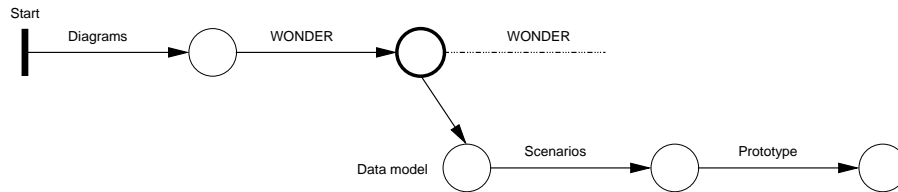


Figure 6.1: SHIPSHAPE design timeline

such as working on the user interface while still having a complete and coherent set of representations. As such it shows the SHIPSHAPE design in WONDER in its most complete form before additional design activities were carried out.

As a consequence of selecting this particular point in time, some of the descriptions in Sections 6.3 and 6.4 will refer to workspaces or materials not described in this section. For example, the inclusion of the workspaces Maintain floorplan and Determine construction order was triggered by the yard events described for the presentation. At that point we found out that these particular problems would benefit from special solutions, i.e. taking into account the layout of the floor and the way in which separate parts of the ship would be fitted together. Initial analysis had not shown this specific need. The background of the Capacity view workspace is explained in more detail in Section 6.3.4. It too is not present in the snapshot below while it is present in the presentation. Finally, we changed one name of a workspace, Small changes in the presentation so that it better reflected its purpose: Maintain activities.

### 6.2.2 SHIPSHAPE workspaces

Table 6.1 on page 155 shows the workspaces in the SHIPSHAPE design and where the full workspace representations can be found. Figure 6.2 on the next page contains a schematic overview of the same workspaces. Two types of relationships have been drawn. Solid lines represent the *enclosed in* and *contains* relationships. Gray lines represent the *works with* relationship, with the origin of the arrow at the workspace which mentions the relationship.

Overall the hierarchy is self-explaining. The only surprise is that Planning engine is contained in both Insert plan and Fix problems. This makes sense because in both workspaces complex changes to the plan may need to be made, which is what the planning engine supports. The three maintenance workspaces are not part of the main hierarchy because they are not tied to the goals

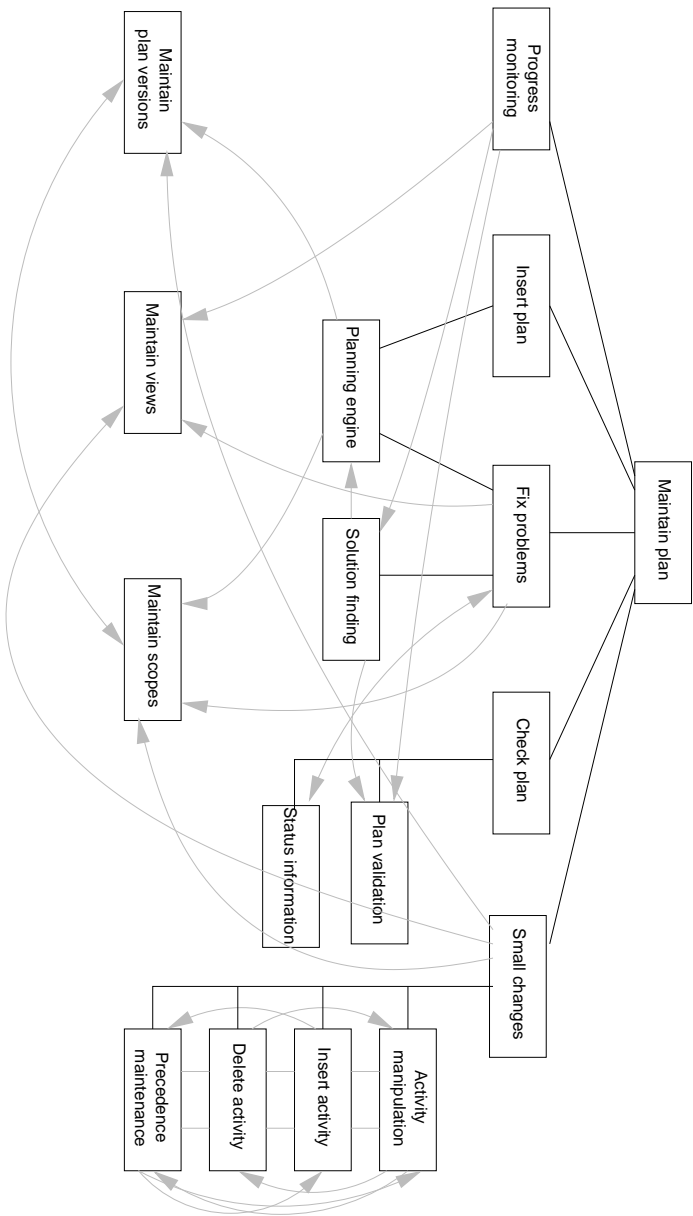


Figure 6.2: High-level hierarchy of workspaces for SHIPSHAPE

of the planners directly.

The *works with* lines paint a more complex picture. There is a lot of interdependence between the workspaces in the Small changes, for instance. In fact, at a later stage in the design these four workspaces were merged back into one workspace called Maintain activities, as is discussed on page 166. The maintenance workspaces are connected to the rest of the design with *works with* lines. The explicit link between Solution finding and Status information is interesting. Status information is an important component of fixing problems. Other workspaces often have specialized status information tools though, which lead to the removal of the status information workspace as discussed on page 166.

**The SHIPSHAPE design and shipyard planning** Several design issues are raised in Section 6.1.2. It is clear from the workspace hierarchy that SHIPSHAPE is designed differently from other planning tools, which usually take something like the small changes workspace in combination with a strong focus on views as their starting point. In the SHIPSHAPE design the goals of the planner are more central to the design. Workspaces such as Small changes and Progress monitoring show that adjusting the plan and monitoring the deviations with reality is designed into SHIPSHAPE.

Workspace	Workspace
Activity manipulation	Maintain views
Check plan	Plan validation
Delete activity	Planning engine
Fix problems	Precedence maintenance
Insert activity	Progress monitoring
Insert project in plan	Small changes
Maintain plan	Solution finding
Maintain plan versions	Status information
Maintain scopes	

Table 6.1: SHIPSHAPE workspaces

### 6.2.3 SHIPSHAPE materials

Table 6.2 on the following page contains an overview of the materials. An example of these representations can be found in Appendix C. The relations between workspaces and materials are shown in Table 6.3, indicating which materials show up in which workspaces.

Material	Material
Activity	Plan version
Actual data	Precedence relation
Capacity Allocation	Project
Deliverable	Resource
Deviance	Skill
Plan domain	Solution
Plan problem	View
Plan scope	

Table 6.2: SHIPSHAPE materials

Some of the materials, such as Plan scope and Plan version are used in many workspaces. This correlates with the *works with* relations shown in Figure 6.2. Many workspaces use the scope and version concepts to delimit which part of the plan is shown, causing these materials to show up in many workspaces. Some of the materials are very specialized, only appearing in one or two workspaces. For example, Deviance and Actual data only appear in Progress monitoring. This could be an indication that this material is very specialized, or that it should really be part of a tool.

#### 6.2.4 Joint evolution of *workspace* and *material*

The evolution of a WONDER design can not be captured easily by describing the representations at some point in time. In this section an example of evolution is given for the workspace Maintain scope (see page 221) and the material Scope (see page 223). The workspace and the material are obviously linked, which makes it interesting to view their evolution together.

A total of 12 revisions can be identified in which either one or both of the representations was changed. Table 6.4 on page 158 lists these revisions. It shows the revision number along with a description and explanation of that revision. The two columns marked 'W' and 'M' indicate whether the workspace or the material changed in that particular revision.

The table shows that changes can start in the workspace as well as the material. For example, the changes made to the material in revision 4 appear in the workspace in revision 5. The changes made to the workspace in revision 7 appear in revision 8 of the material. This illustrates the possibility to have an inconsistent design in WONDER. Also, the use of questions is illustrated in revisions 9 and 11.

Table 6.3: SHIPSHAPE materials and workspaces

Rev.	Changes and explanation	W	M
1.	Initial version of Maintain scope. At this time it is called 'plan scope'.	×	
2.	The type of relation with Main is made more clear.	×	
3.	Initial version of Scope. A link to this is added in Maintain scope.	×	×
4.	At this time the representations were discussed with the domain expert. Based on this discussion the name of Maintain scope was changed in 'Maintain plan scopes'. In this discussion it also became more clear that a Scope would be a tool to limit the working area, and that it would be the primary selection tool. To facilitate this a 'time window' was added to the associations.	×	×
5.	The workspace name is finally changed to Maintain scope. The changes made to Scope in revision 4 are now reflected in the workspace: a relation with Plan version is defined to make clear that there is a strong relation between a selection and versioning, 'Plan elements' are defined as a material, and the 'Maintain scopes' tool is defined to create and delete scopes.	×	
6.	'Project' and 'Time window' are added to the associations in Scope. This is also reflected in Maintain scope, where they are identified as plan elements. The 'Maintain scopes' tool is split in two separate tools for creating and deleting scopes.	×	×
7.	A 'Select scope' tool is added to Maintain scope. This results in 'time window' now being seen as a constraint or a property of the scope, not as an element.	×	
8.	The changes made in revision 7 are now reflected in Scope. 'Time window' is now removed from the associations, and added to the attributes list, along with others such as 'name', 'selection criteria', 'activities', and 'resources'. Also, the challenges for properly visualizing a scope are made more explicit.		×
9.	A question is added to Maintain scope. The manipulation of scopes is not clearly described in the functional specifications [Guy95b].	×	
10.	In response to changes made in revision 8, a tool is added to Maintain scope which shows which elements of the plan are part of the scope.	×	
11.	The question posed in revision 9 is answered. New tools such as intersect scopes and join scopes are added.	×	
12.	Both Maintain scope and Scope are further refined. Attributes and tools are made more clear and more precise.	×	×

Table 6.4: Revisions of Maintain scope and Scope

### 6.2.5 The WONDER representations

Actual use of the representations can be compared with the formal descriptions in Section 4.3.3.

**Workspace representation** For the most part the workspace representation was used in the same way as described in the theory. Most significantly, the *interaction* field was not used much. Often not enough detail was known yet to propose a particular interaction strategy. This fits with the idea that thinking about specific interaction elements is postponed until the WONDER design nears completion.

Two additional fields were added, with the information initially appearing in the remarks section. A *design alternative* is a description containing text and graphics. It describes how a particular workspace or material might be represented on the screen, or what kind of alternatives could be used to solve a particular issue with a workspace. As such they provided an informal implementation of Design Space Analysis. A *scenario* is used in preparing the presentation described in Section 6.4.1. These scenarios were built on a combination of common events taking place on a ship yard and particular workspaces. As such they were building on the workspace descriptions, while providing a real-world usage example.

**Material representation** The material representation was used according to the theory description. No deviations were observed.

**Action representation** The action representation was not used much. In fact, after some initial trials, this element was pretty much abandoned. The reason for this is that there is simply little added value in using these representations at this stage in the design process. They require too much detail that simply is not important yet.

Consider as an example the action representation shown in Figure 6.3 on the following page. There is not much useful information in this representation. In fact, the remarks are quite generic and also fairly obvious for a 'create' action. This makes the representation nothing but a placeholder of questionable value which will not become more useful until later in the design process. An explanation is that actions are either very trivial, such as select and start, or they are more complex, in which case their implementation may depend strongly on the context and they will likely be described as tools.

<b>Goal</b>	To support creation of a new instance of a material
<b>Extensions</b>	(none)
<b>Remarks</b>	<p>Creating something usually has two implications:</p> <p>1) It should be put somewhere. This also opens up the question whether more elements can be created at the same time, and how this will be handled.</p> <p>2) Its attributes have to be set in some way. This can be done simply by editing its attributes, but it could also be done by context. For instance, because another object is selected, it will be an attribute of the newly created object.</p>

Figure 6.3: Example action ‘create’

## 6.3 WONDER in use

The way of working with WONDER is described in Section 4.4. Working with WONDER requires three sets of activities to be carried out a number of times: *finding workspaces*, *assessing workspaces*, and *refining workspaces*. In this section the use of these three sets of activities is described. Each is also compared to the task diagrams found in Section 4.4. The combined effects of *assessing workspaces* and *refining workspaces* on the evolution of the design are described separately. Finally the individual activities are discussed, and the experiences of the design team are presented.

### 6.3.1 Finding workspaces

Section 4.4.1 contains a description of the process of *finding workspaces*. Two distinct phases are recognized. The first phase, in which workspaces are uncovered based on material collected during analysis, is described below. The second phase is discussed as part of the *assessment of workspaces* in Section 6.3.3.

**Use of analysis results** The basis for *finding workspaces* is to use the material collected during analysis. In particular the hierarchy of *goals* and *objectives* provides a good starting point. The results from the analysis of the shipyard planning activities at a particular yard<sup>2</sup> are described in three documents. These documents, described below, have served as the input for finding

---

<sup>2</sup>IHC Slidrecht, located in The Netherlands



the initial set of workspaces. Table 6.5 provides a summary of their contents. The results found at this yard were later verified against several other yards, but no significant differences were found [Guy01].

The first document contains a broad sketch of the problems found during analysis and the proposed improvements in five major areas [Guy95a]. This document deals with the planning for the company as a whole, and lacks the view of the planners. It does identify a number of high level *objectives* for the yard, and provides *context* for these objectives. A number of important planning artifacts, e.g. *activity* and *resource*, are also described in this document.

The second document contains a list of functional specifications [Guy95b]. Even though this document is aimed at the software engineer who needs to implement the core functionality for the system, it does provide useful information for a workspace based design. Specifically, descriptions of the objects to be used are very useful as an inventory of *materials*, while the description of the functionality hints at both *objectives* and *tools*.

The third document contains a task analysis of the more important parts of the planning process [Guy95c]. The task analysis contains a number of task hierarchies for the most important planning *objectives*. Each hierarchy gives a good indication of the important objectives and sub-objectives for each of these important events. Each hierarchy is also accompanied by a list of people responsible for each task, making it very easy to place the activities in the organization as a whole.

Document	Focus	Objectives	Context	Materials
Broad sketch	Whole yard	High-level	Yes	Few
Functional specs	Core functionality for SE	Some hints	No	Yes
Task analysis	Planners	Yes, but not complete	Yes	No

Table 6.5: Contents of analysis documents

**Workspace discovery** The initial discovery of workspaces followed naturally from an examination of the analysis results. Initially, diagrams were used instead of the WONDER representations. This was not a conscious decision. Some doodling while examining the analysis results led to these diagrams. This does illustrate the fuzzy boundary between analysis and design; without consciously starting the design activities, some had already taken place.

The diagram was created to cluster high-level *goals* of shipyard planning according to the main processes taking place in a shipyard. This clustering was done visually, as can be seen in part from such a diagram in Figure 6.4. In this diagram the clusters of related *goals* are indicated by broken lines. This clustering provides a crude encoding for the three workspace *relationships* in a WONDER representation. The strength of some of these relations is shown by thicker lines between goals, thus providing some information on context. Additional contextual information is also encoded in the rectangles, which indicate the context for a cluster of goals.

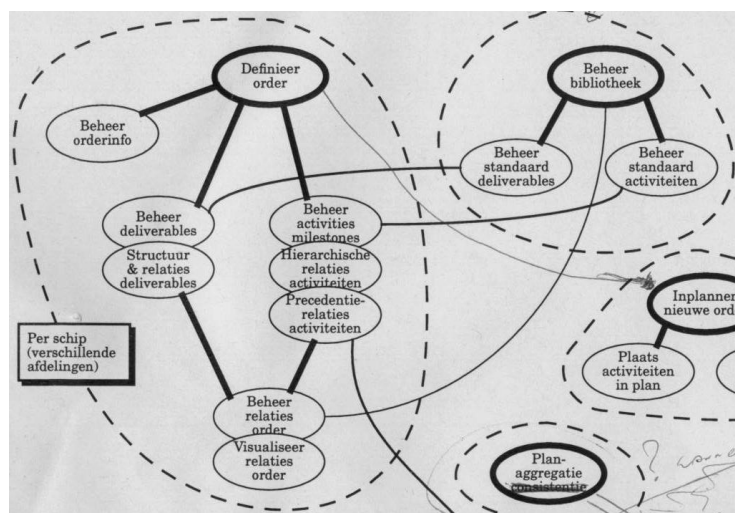


Figure 6.4: Part of an early design diagram

It is interesting to see that these diagrams are not part of WONDER theory. According to the theory, workspace descriptions need to be made as soon as the first workspaces are formed. The information captured in the diagrams does conform to the information normally captured during the creation of WONDER workspace descriptions. The goals and subgoals are the primary attributes of the workspaces, and some context and the relations between workspaces are also indicated. It is primarily the *form* of the description that is different.

The preference for these diagrams over the workspace descriptions results from the diagrams providing a very light-weight description, while at this stage holding all needed information. Such a light-weight description proved

to be quite beneficial, because many changes were made in this stage of the design. Furthermore, because of the compact notation all of the design could be kept on a few pages, which made it easy to keep an overview of the design. Finally, the diagram also was a great tool for discussing the design with shipyard planners, precisely because it was light-weight: it did not look complicated. Figure 6.5 shows the annotations after such a round of discussions. Use of these diagrams facilitated an efficient start of the design process and the information contained in them could easily be encoded in WONDER's representations.

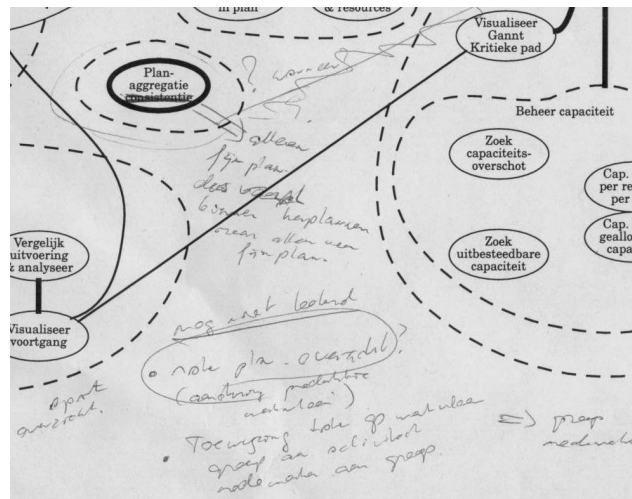


Figure 6.5: Design diagram with annotations

### 6.3.2 Evolution

After finding the initial set of workspaces the design process continues with an iteration of *assessing* and *refining* workspace descriptions. The evolution of the design from the initial starting point onward is reflected best by the combination of these two activities. This section provides an overview of the evolution of the design, while the following two sections contain specific examples of the individual activities.

The effects of the iterative process can be seen by considering the changes

to all representations in a structured manner. Each change in the descriptions, as recorded in the version control system, can be attributed to one or more of the following categories:

**Result of an assessment (a)** Based on the task structure in Figure 4.7, an assessment will mostly be indicated by the removal or introduction of a workspace. Removal of a tool is also an indication of assessment, but the introduction of a tool can be the result of either assessment or refinement.

**Result of refinement (r)** According to the task structures in Figure 4.8 and Figure 4.9, refinement is indicated either by the addition of subsequent description elements, or by meaningful changes in current descriptions, especially when previously noted questions or warning disappear at the same time.

**Insignificant changes (i)** Changes insignificant for the design. This usually includes spelling corrections, formatting fixes, and minor reformulations of the text.

Mapping these three categories of changes over time for each of the WONDER representations can provide insight into what way WONDER supports the evolution of the design. This mapping is shown in Table 6.6 on the facing page. In this table the days worked on the design are mapped against the workspace representations of the case, showing the types of changes each workspace has gone through.

Two observations can be made about the pattern of the changes. First, on a given day, changes occur in many representations, or only in a few. This can be explained by the fact that some changes have profound effects on the whole of the design, affecting a large number of representations in some way, while specific detailed work on a representation can be seen as isolated from the rest of the design. Second, the occurrences of the massive changes diminishes as the design progresses in time, which indicates a convergence in the design process.

### 6.3.3 Assessing workspaces

While the previous section provides an overview of the evolution of the design, it is also instructive to look at the two processes responsible for this evolution in more detail. The next section contains examples of the *refinement process*. In this section some specific examples of the *assessment process* are described. During assessment, workspaces can be added to and removed from the design.

Activity manipulation	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Check plan	r		r	r	r	ir	r	r	r				ir	
Delete activity			r		i			a	ir				i	
Fix problems			r		i				ai				i	
Insert activity			r	r	r		r	r					i	i
Insert project			r	r	r	r	r	r	r		r	r	i	
Main		r												
Maintain scope			i		i	i			a	r			r	
Plan validation			r		r	r		r	r	r			i	
Planning engine		r	r		ir			ir	ir	r			ir	
Precedence manipulation			r		ir	r	r	r	ir	ir			i	
Progress monitoring														
Small changes			r		i				ir	r			i	a
Solution finding	i		r			r			ir	a			i	
Status		ir	r		i				r	r		i	i	
Versioning														
Visualization			r			ir			r				i	
days (continued)	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Activity manipulation				r	r	r	i		r					
Check plan								ir						
Delete activity														
Fix problems								r						
Insert activity				ir				r						
Insert project				r										
Main	i			r					a					
Maintain scope						i		r			r	r		
Plan validation				r					r			r		
Planning engine	i			r							r			
Precedence manipulation				r										
Progress monitoring				r							r			r
Small changes														
Solution finding	i			r			a		r			r		
Status														
Versioning	r		r			i		r			i			
Visualization	r	ir				i						r		

Table 6.6: Categorization of changes to design representations

**Workspace to tools to workspace** One noteworthy change instigated by discussions with real planners concerns the ‘small changes’ cluster of workspaces. Initially the small changes workspace was split into a number of smaller workspaces associated with the specific small changes that can be made to the plan, e.g., insert new activity and maintain precedence relations.<sup>3</sup> This seemed a logical choice, because each of these changes required somewhat different tools and some of these tools could be fairly complex. Discussion with planners revealed that they did not find this split logical at all; these workspaces belonged together and would be used interchangeable. Given the amount of overlap the four workspaces were merged back together again into small changes.

**Removing a workspace** The status information workspace was removed after discussion between the interaction designer and the domain expert. The initial objective of this prospective workspace was to provide status information on the plan. After a number of iterations we found it increasingly harder to make this workspace more concrete: what kind of status information would need to be displayed and what tools, if any, would be needed? Finally we realized that status information is always related to a particular objective and that just “getting status information” didn’t make sense without an objective. Already all kinds of specific status-related items had found their way into other workspaces, so we simply abandoned the status information workspace.

**Tools become a workspace** The maintain scopes workspace provides an example of a common set of tools turned into a workspace. A scope in SHIPSHAPE is a selection of activities and precedence relations in the plan. Analysis of current practice had not revealed such an explicit concept and hence no such workspace was added initially. As the design was progressing and workspace descriptions were getting more complete, similar tools were introduced in a number of workspaces to be able to deal with a set of activities at once. Instead of adding the same tools to each of these workspaces, it made more sense to simply create a new material **Scope** and a workspace to maintain them: maintain scopes.

---

<sup>3</sup>Their representations are still present in Table 6.1 and Appendix C for easy reference.

#### 6.3.4 Refining workspaces

Each of the workspaces was also continuously refined and revised, both based on discussion between the interaction designer and the domain expert and based on the emerging relations between workspaces.

**Dropping a workspace from the design** At first there was a workspace called Solution finding in the design. The goal associated with this workspace was to facilitate finding solutions for specific planning problems with the aid of an automated planning engine. For instance, the planning engine might suggest to move activities or schedule overtime in order to solve a problem with over-allocation of a resource. Originally, such suggested solutions were to be presented with the Solution material. While further refining this workspace, it became apparent that the concept of a ‘solution’ was not well defined at all. Upon further investigation a ‘solution’ turned out to be a combination of a Plan version and a Plan scope, both of which were already well-described in the design. This conclusion led to the decision to drop the ‘solution’ concept altogether, and use the already familiar scope and version concepts instead to represent the same information. The functionality of the Solution finding workspace could then be moved to the Planning engine workspace.

**Uncovering missing workspaces** Several ‘views’ on the plan data were not initially included in the design. Views provide an appropriate visualization of a plan. Examples are the GANNT view which shows activities and their relations on a time line, and the capacity view which shows the allocation of capacity of resources over time. Initially there was a workspace maintain views with the associated goal of handling the creation and manipulation of these views. This workspace was written on the assumption that these views would be used to display information only. This is much too limited because the views also provide a good starting point for plan manipulation and investigation. These omissions in the design were detected during discussions with the participating users as part of a workspace refinement cycle. The most important views were turned into workspaces themselves, to be used from other workspaces through the *works with* relationship.

**The role of ambiguity and inconsistency** In general allowing ambiguity in WONDER’s representations is positive. It allows for quick data entry even when

not all information is clear yet. This makes gradual progression easier, as all information can be kept in the same place.

One drawback of allowing ambiguity in the representations is that the ambiguity can sometimes be confusing. After a few days some descriptions are simply not clear, and in this case the issue leading up to the description needs to be revisited. This happens regardless of allowing ambiguity in the representations, but now it is more explicitly part of the design process.

Allowing inconsistencies in WONDER's representations is a big help, because it permits the design team to focus on a single representation at the time and forget about the big picture for a bit. This opens up a bigger design space as more alternatives are considered. It also takes some pressure from the design team, which is then more free to concentrate of the design instead of on the consistency. At a later time making the design consistent becomes a priority and all of the concentration can go to that topic. For example, Figure 5.8 on page 144 shows an inconsistency where a material is not mentioned within any workspace.

### 6.3.5 Design team

This section provides an introduction to the design team of SHIPSHAPE, and the mapping of roles onto team members as described in Section 4.5. Furthermore, the design team is evaluated in part by looking at the case and in part by reflecting on HCI theory presented in Section 3.4.

**The design team members** The design was primarily created by two people. The author of this thesis was one of them, practicing action research as described in Section 1.4.3. He is familiar with WONDER, user-centered design, user interface design and programming. The other designer is doing research on shipyard planning. In addition, he also worked on a shipyard office while working on his MSc. thesis [Guy92]. A student from Delft University of Technology helped implement parts of the design as part of a student assignment, and also gave valuable technical feedback on aspects of the design. In addition we also frequently consulted a planner working at a shipyard and a director of a non-profit organization aimed at introducing innovation in shipyards.

**Mapping roles onto team members** The mapping of the roles onto team members is shown in Table 6.7 on the next page. With the design team being so small, two team members had to take on more than one role. The match



between skills of team members and available roles was, as it turned out, somewhat unfortunate. Some of the side effects of this are discussed below.

Person	Roles
Researcher	Domain expert, Project manager, Software engineer
Researcher	Interaction designer, Visual designer
Student	Software engineer
Planner and Director	Participating users

Table 6.7: Role assignment in SHIPSHAPE design team

The planner and director were only available on occasion, so their role was limited to that of the participating user; all other roles would have cost them too much time. The double assignment of the software engineering role was caused by two things. First, the student did not join the design team right from the start. Second, one of the researchers already had a clear view of some of the software engineering issues based on his earlier research.

**Team leader** According to WONDER the domain expert should be the team leader. In the SHIPSHAPE case this was not strictly observed. Two reasons led to a sometimes confusing situation. First, in this case the interaction designer, having created WONDER, also was seen as the team leader at some times. Second, the domain expert was also involved in another project, and could not always pay full attention to the SHIPSHAPE project. As a result, there was confusion sometimes over who should do what, and whether something was good enough or not. In a few instances the interaction designer made some decisions which were later overruled by the domain expert based on knowledge of the domain. This lends support to making the domain expert the team leader.

**Evaluation** The particular mapping of roles in this case did have some drawbacks. For instance, with three designers having a software engineering background, assigned or otherwise, there was a strong focus on software engineering topics. No design ideas were thrown out later because of software engineering evaluation comments. This seems to indicate that the design space has not been explored well enough. It is plausible that only those ideas appearing to make some sense from a software engineering point of view have been explored. In other words, the software engineering knowledge within

the primary design team members likely filtered some design solutions before they could come up for more general consideration. It appears to be beneficial to have more diverse design team members, which is an obvious conclusion given the multi-disciplinary nature of the design work.

Another problem which was encountered is caused by having a person be assigned to more than one role. When this happens, the communication between those roles becomes mostly implicit. This can cause problems when decisions need to be refined, as there often will not be a record of the decision. Also, by leaving communication implicit, arguments are not always tested very well, resulting in an increased chance that a decision is wrong. An example that illustrates this issue is the fact that some views were initially forgotten in the design (See page 167).

Another example of the latter problem can be illustrated with the Scope concept in the design. The domain expert had one view of this concept, and cleared it internally and implicitly with his software engineering role. The interaction designer had another view of this concept, and cleared it implicitly with his visual designer role. The two views on the concept were not identical, however. Fortunately this mismatch could be resolved early in the design, but it goes to show that verbalizing the design is an important safeguard.

Despite these problems, the design team was happy to use the representations to describe the design. The representations did provide support, and the team did perceive their use to be beneficial. The interaction designer found it a quick yet concise way to describe the evolving design, while the domain expert found the descriptions easy to work with and providing a good grasp of the design and the design issues.

### 6.3.6 Comparing use of WONDER to the task structures

Section 4.4 describes the way of working with WONDER. The three main design activities are described and further illustrated with task structures, shown in Figures 4.6, 4.7, 4.8 and 4.9. In this section actual use of WONDER in the SHIPSHAPE case is compared to these task structures.

**Finding workspaces** There is no task structure listed for *finding workspaces*, because the activity is quite simple. In the SHIPSHAPE case the way of working was followed. The initial set of workspaces was found based on the results of the analysis phase, as reflected by the analysis documents listed in Table 6.5. The main difference between the way of working and what happened in the

case is the way in which those workspaces were described initially. As was already noted in Section 6.3.1, visual diagrams were used as a first step. The finding of additional workspaces, the second part of this activity noted in the way of working, is described as part of the other WONDER activities.

**Assessing workspaces** The task structure for *assessing workspaces* is shown in Figure 4.7. Basically, its goal is to perform a review of workspaces in terms of the whole of the design. The very existence of the workspaces comes under scrutiny. The reverse can also occur, a workspace can be created as a result of assessment, for instance by turning a complex tool into a workspace of its own.

As discussed in Section 6.3.2 and illustrated in Table 6.6, assessment is responsible only for a few changes in the design representations of the SHIP-SHAPE case. These changes are further categorized in terms of the tasks in the task structure in Table 6.8.

Workspace	Day	Type	Description
Delete activity	7	Break down	Moved from Main to Small changes
Fix problems	9	Break down	Moved several tools into their own workspace
Status	10	Not relevant?	Perhaps this should not be a separate workspace, but rather be integrated with other workspaces
Status	14	Not relevant	All tools operate on a <i>Scope</i> , so move them to a scope-related workspace
Small changes	21	Locate common tools	Bring together several smaller workspaces into a single one
Main	23	Tool as workspace	Make progress monitoring a separate workspace

Table 6.8: Assessment changes to workspaces

The task structure contains all possible ways for a workspace to be assessed. It is interesting to see that despite the small number of assessments in the design only one of the theoretical assessment outcomes in the task structure is not encountered: a workspace that is too simple to be a separate workspace. All other outcomes have occurred during SHIPSHAPE's design, as is shown in Table 6.8.

What is perhaps surprising is the small number of assessments. Only six changes in the design qualify as assessments. In Chapter 4 no predictions are

made as to the frequency of the different steps in the design, but six assessments does not seem a lot compared to the number of refinements.

One possible explanation is that in this case the analysis had already uncovered much of the structure of the work, so that major changes were not needed as much during the design phase. Another possible explanation is that making major changes during the design is still a daunting task that is rather avoided, despite being cheap at this phase of the development process and being made easy with WONDER.

**Refinement of workspaces—step 1** The first step of this activity is shown in Figure 4.8. In addition to a task structure, the figure also shows the order in which the elements will be added to the representation.

The ordering is basically correct. Two observations can be made. First, clustering does occur in practice: two or more elements are added at the same time, e.g. both *materials* and *tools*, or both *goal* and *relations*. Second, iterations can occur, i.e. by going back to relations or goal after adding materials. While this is a natural result of the second step of refinement, it is worth noting explicitly. Table 6.9 shows the most common occurrences of changes made to workspace descriptions, either to one part of a description or to several.

Combination	%
tools & materials	20%
relations	14%
materials	14%
goal & relations	10%
tools	9%
goal	8%
housekeeping	7%
other combinations	18%

Table 6.9: Occurrence of different clusters of changes

Two clusterings are apparent: *tools* together with *materials*, and *goals* together with *relations*. Both of them are not unexpected, because they have strong ties between them. Materials and tools are related because the tools need to operate on the materials, so it makes sense to change one when changing the other. The goal and the relations to other workspaces are also strongly related, because they both indicate something about the position of the workspace in relation to other workspaces.

One interesting thing to note is that three different types of workspaces appear to have emerged. Table 6.10 shows how the SHIPSHAPE workspaces listed in Table 6.1 can be mapped to these types. The three types are:

- workspaces that tie things together, often related to the high-level goals,
- workspaces that help to carry out the more detailed goals,
- workspaces that contain the common tools of the domain.

<b>Tie together</b>	<b>Detailed goals</b>	<b>Common tools</b>
Check plan	Activity manipulation	Maintain plan versions
Fix problems	Delete activity	Maintain scopes
Maintain plan	Insert activity	Maintain views
Small changes	Insert project in plan	
	Plan validation	
	Planning engine	
	Precedence maintenance	
	Progress monitoring	
	Solution finding	
	Status information	

Table 6.10: Types of workspaces

The task structure in Figure 4.8 shows explicit cooperation between the domain expert, who suggests which elements to add, and the interaction designer, who then interprets these additions in terms of the overall design and who analyzes the additions in terms of HCI principles. This possibly results in a feedback loop where the interaction designer gives criticism and suggestions to the domain expert as additional input. Otherwise the suggested elements are added. This process is repeated several times until the workspace is fully described.

In the SHIPSHAPE case this was not what happened. In particular, the division of labor was not present; at least, not in the way described in the task structure. What really happened was that the interaction designer proposed the elements to be added, and added them as well. This reduced the role of the domain expert to checking the added elements and giving feedback based on this. Obviously the elements proposed by the interaction designer were based on work done by the domain expert in one way or another, but the key

difference with the task structure is that the involvement of the domain expert has been much more indirect.

This can be explained by two things. First, the research instrument used to create the descriptions was not usable enough, as is explained in Section 5.3.1. Second, the constituency of the design team and the mapping of roles within it had a tendency towards implicit discussion, as is explained in Section 6.3.5.

Furthermore, the lack of division of labor observed in the SHIPSHAPE case has to do with ownership and leadership. The theory description in Chapter 4 does not give enough guidance in this respect. It is unclear who has ownership of which representations. It is implicitly attributed to the group, but this is by no means clear. Also, it is unclear which role has to take the lead, and thus the responsibility and accountability for the different tasks. There is some discussion on this topic in Section 4.5 when the team leader is discussed. There the domain expert is identified as the most likely candidate for being team leader, but this is not strongly stated, and it does not affect the individual tasks.

**Refinement of workspaces—step 2** The second step in *refining workspaces* is aimed at a continuous process of refinement, in which each workspace is put under the microscope from time to time, and an attempt is made to resolve the open issues with the workspace. This process is illustrated by the task structure shown in Figure 4.9. Both step 1 and step 2 of the refinement process happen in parallel, although both steps are not formally connected. As with the previous task structure, this task structure does not reflect the case in several ways.

This task structure describes how to deal with *questions* and *warnings* present in the representations. Obviously, these provide a good starting point for cleaning up a workspace, but it should by no means be the only thing. Progressive insight into the whole of the design can also be a good starting point for refinement, as can be a simple rereading. In effect, the single-handed focus on questions and warnings does not do justice to the full set of possibilities for refinement during actual design activities.

Furthermore, the task structure seems to indicate that this process needs to go on until *all* uncertainty is removed from the design. While this may be a desirable end solution, it may also be too much to ask. Certainly in the SHIPSHAPE case some questions and warnings remained until the end, if only because they were deferred to other design activities further on in the design process.

Another difference between the task structure and the case is that the task “scrutinize workspace relations” is actually much broader than just that. It

also includes all the other explicit and implicit relations which can be asserted through consistency checks such as those described in Section 5.2.4.

## 6.4 Testing the SHIPSHAPE design

This section contains a description of the way in which WONDER's results have been tested using a selection of real planners from Dutch shipyards. First the testing approach is outlined, after which the results of the test are presented. Finally a number of conclusions which can be drawn from these results are described.

As part of the tests a prototype planning environment has been implemented. This provides an opportunity for reflection on how WONDER's representations can be used as input for the remainder of the design process.

### 6.4.1 Test approach

Testing can be conducted by using WONDER in the design process of a large, real-world, system. During the design process the design team can be observed and interviewed, and the WONDER descriptions can be monitored over time. When the design has been created, it can be presented to potential users of the system, and their perception on how the system will support them in their current work can be measured.

Within the confines of the research project it has not been possible to design and build a complete planning system and implement this in several yards. There are several reasons for this. First of all, creating a complete and operational system requires a large investment of money and time, neither of which was available. Building such a system was estimated to cost approximately \$500,000.00 and take about 10 man years of work. In addition lead time for the project was estimated to be at least a year, and likely more than that.

Furthermore, shipyard planning is critical to the operation of a yard. Briefly switching over to another system is simply not possible because of the huge investment and risk involved in such an endeavor. Also, running a shipyard is currently a very competitive business, so not much time can be spared for testing a new planning system.

**Presentation** In order to alleviate these constraints, a computer based presentation has been created [dGGP97]. The presentation consists of two parts. The first part introduces the problems with current shipyard planning practice

and outlines the new approach and issues and solutions associated with it. The second part guides the planner through a number of situations often found in shipyard planning. Table 6.11 on the next page shows the table of contents for the presentation along with a brief description of each module including the workspaces to be tested.

The first part of the presentation is not interactive. The planner is simply taken through the presentation, which consists of a number of screens along with an audio-based narrative. This setup was chosen because it ensures that a consistent story is told to all planners. It also provides consistency with the second part of the presentation. Finally, it allows planners to rerun the presentation at a later time. The pace of the presentation could be controlled by pausing the presentation. This feature was used occasionally when planners had specific questions or when they reflected on the information in terms of their own yard. Figure 6.6 shows a screen from the presentation explaining the activity view when planning part of a project.

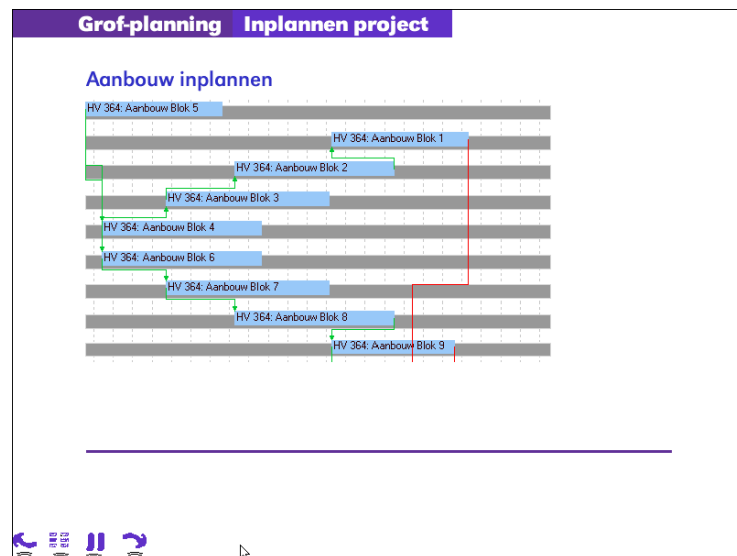


Figure 6.6: Screenshot from presentation

The second part of the presentation follows the same format as the first part, but at a number of places interactive prototypes of planning tools are



Section	Contents
<i>Part One</i>	
Introduction	Gives an introduction of the presentation system and the prototype.
The Yard	Introduces a fictitious yard which is used throughout the presentation. It also highlights some of the issues and problems with this yard.
Problems	Goes over a number of problems and their causes with current planning activities at the yard.
Concepts	Introduces the new planning concepts as mentioned in [Guy01].
<i>Part Two</i>	
Scenario 1	Investigate the current capacity allocations. Determine bottlenecks in the allocation of the ramp, and determine slack with some disciplines. Test <b>Capacity view</b> .
Scenario 2	Create a temporary plan for a new project. This includes creating an activity network within a new plan version using the GANNT view. Also check resulting capacity allocations. Test <b>Maintain activities</b> and <b>Maintain plan versions</b> .
Scenario 3	Create a validated high-level plan for a new project. Use automated tools for fitting the activity network. Check the capacity allocations. Test <b>Determine construction order</b> and <b>Planning engine</b> .
Scenario 4	Create a detailed plan while keeping an eye on the rough plan boundaries. Deal with large amounts of data in the plan. Test <b>Maintain views</b> .
Scenario 5	Check progress of work. Detect (potential) problems and try to solve them. Test <b>Progress monitoring</b> .
Scenario 6	Create a visual floor plan for assembly of the ship, and investigate the possibilities this approach offers. Test <b>Maintain floorplan</b> .
Scenario 7	Create a plan for work groups, and allocate already planned in activities to groups.

Table 6.11: Contents of SHIPSHAPE presentation

added. These prototypes are described in more detail below. This part of the presentation is divided in a number of scenarios likely to occur on a shipyard. Each scenario first introduces the circumstances, after which it is explained how the new planning approach can support the situation. At suitable points in the presentation the interactive prototype can be used to look at planning data or to solve small planning problems. After working with the prototype the presentation can be continued. With the completion of each scenario the planners would fill out a questionnaire, which is discussed below.

**Prototype** A presentation alone is not sufficient to provide planners with a good understanding of the proposed system, in particular for those parts of the system which are very interactive. For this reason a prototype system has been developed which allows the planners to interact with the activity and capacity views. The prototype is limited in that no functionality is implemented outside of these views, and that no special care is given to issues such as reliability, usability, or loading speed and storage facilities. The activity and capacity views are fully operational, though. Most importantly, the elements in the views, such as activities and dependency relations, can be edited and moved in order to allow the planners to ‘play’ with the plan, and in this way get a real understanding of problem solving with the proposed system.

The prototype features an extensive database. The database is both extensive in the different kinds of data it incorporates, as well as in the amount of data entered into it. Both are needed to make the planner’s interaction with the prototype as close to the real world as possible. A wide variety of things all influence the constraints and relations between different data elements. To ensure that the planners will experience such intricate relationships, data for all of these elements needs to be present in the database. As an example, the usage of the ramp was tailored to provide a point in time in which a new project could almost, but not quite, be fitted using the default building approach. As a result, planners needed to make changes to the building approach to get the project to fit; or to conclude that this could not be done.

In a similar vein the database needs to contain a large number of records which together describe the current state of the shipyard plans. Even if only a few ships are to be shown in the plan, a lot of data lies underneath each ship. While the data does not need to match exactly, it does need to provide a believable situation to the planners. To provide the planners with realistic data, plans were collected at a shipyard. This data was used to populate the fictitious yard with 5 ships. Most of these ships were only described at the

project and rough detail plan levels, keeping the number of activities down to about 75 activities per ship. 15 resources common to most yards were also included in the data set.

The capacity view, shown in Figure 6.7, is shown to the planners first. It gives an overview of the different skills available at the yard. For each skill the available resources over time are listed, overlaid with the requested resources. These requested resources are derived directly from the activities in the plan. The capacity view only facilitates some interaction, mostly aimed at rearranging the view itself for easier access to the information. For instance, each skill graph can be collapsed to a thin line, reducing space while retaining key information. The planner can also make changes to the available resources for each skill, for instance by hiring some additional workers for a few weeks. No changes can be made to the requested capacity, because this information is derived from the activities, and therefore needs to be changed in the activity view.

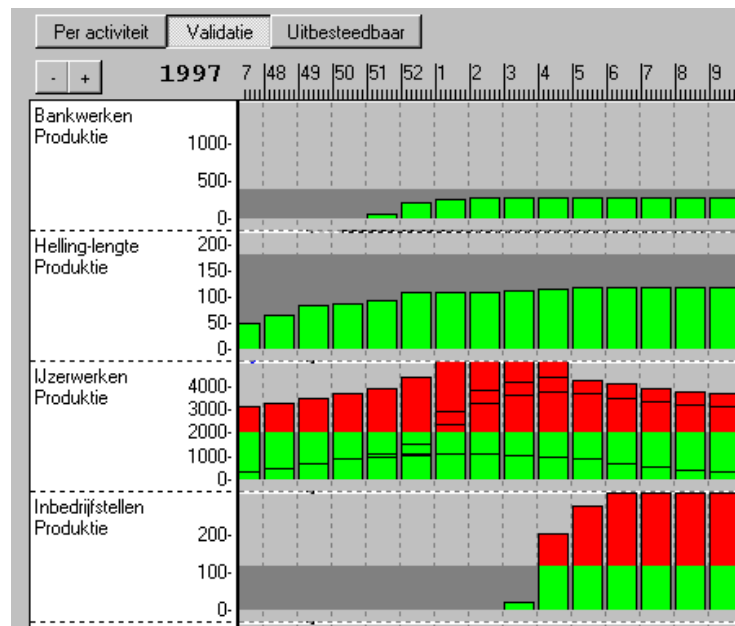


Figure 6.7: Capacity view (screenshot)

The activity view, shown in Figure 6.8, provides access to the activities defined in the plan. The primary purpose of the activity view is to allow access to the activities, and to show them in relation to each other, and to the time line. Interaction in this view is primarily aimed at positioning each activity correctly in time. This involves moving the activities, but also modifying their lead times. A number of indicators are used to show various constraints.

It is also possible to view the activity and capacity view in a synchronized fashion. The result of this synchronization is that scrolling one view in time will also scroll the other, keeping them synchronized in time. Also, changes to the activities are immediately reflected in the capacity view.

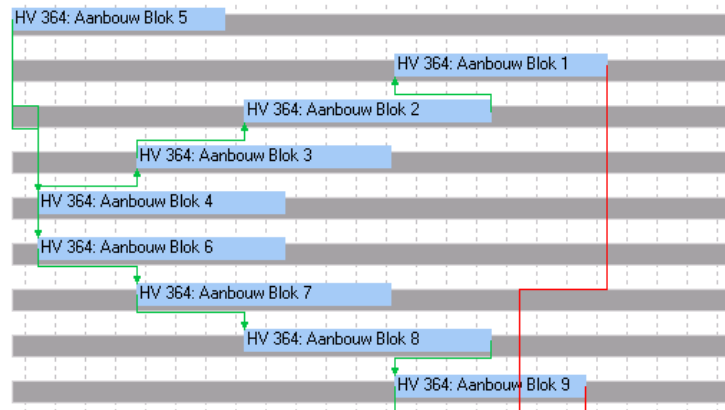


Figure 6.8: Activity view (screenshot)

**The questionnaire** In order to solicit feedback from the planners a questionnaire has been used. A separate questionnaire has been used after each scenario to ensure that the details of that particular scenario, and the aspects of the design on which it is focused, were still fresh in memory. Each questionnaire is structured in a similar way.

First, a general explanation of the questions in the questionnaire is presented. This explanation is identical for all the questionnaires, and is included for easy reference. Then, the purpose of the scenario is stated, much as is done in Table 6.11. This will help to focus the answers on the important and relevant issues of each scenario. In most questionnaires this is followed by a number of questions about the planning problems and proposed solutions. The purpose

of these questions is to test the shipyard planning theory briefly presented in Section 6.1.1. These questions are not directly related to the testing of WONDER.

Next, one or more workspaces are evaluated. The evaluation of each workspace follows a standard pattern. The evaluation starts with a list of components of the workspace. The user is asked to rate how important each of these components is for reaching the goal associated with the workspace. Each component is ranked on a 4-point scale, with the added option of not having a specific opinion about it, for instance because it concerns an area of work with which that particular user is not familiar. The list of components is directly related to the *materials* and *tools* defined for the workspace. After the list of components, the user is asked to describe any additional components which were not mentioned. Finally, the user is asked to compare relevant tools currently being used at their yard with the workspace.

The questionnaire is paper-based. This gives the planners a welcome break from the computer screen. It also makes it easier to write comments or sketch other solutions or current practice. Finally, it also makes the results more reliable than having the questionnaire on the same computer as the presentation. In the latter case the results are in danger of being too positive and not expressive enough [RN96].

**Testing environment** In order to facilitate the planners of the shipyards as much as possible, each yard was visited to carry out the testing. By bringing the test environment to the yards we were able to convince more planners to cooperate in the test. The testing equipment consisted of PCs with 17" screens, and audio capabilities. The machines would be set up in a conference room at the yard, so that interruptions could be minimized. The complete presentation would be cut into three sessions because of its total length.

The first part of the session, including the presentation of the yard, current problems in planning, and proposed solutions, was often done as a plenary session. We found that this facilitated discussion of the problems and their underlying reasons, and also helped to reflect the presentation back to the planner's own yard. A plenary session was used for this part of the presentation because it facilitates interaction within the whole group, and because this setup allows more different opinions to be brought forward.

The remainder of the presentation, consisting of all scenarios, was conducted in pairs as much as possible. This allows the planners to talk to each other about the scenarios and the problems encountered in them. It also makes it easier to work through the new concepts and software prototype, because

two people often see more than one, and because assumptions need to be made explicit for them to be communicated to the other planner. Finally, working in pairs changes the traditional think-out-loud technique into co-discovery learning, which means that more information becomes available in a much easier way [BB99].

The questionnaires were filled out individually, allowing for more, possibly contrasting, viewpoints. Also, different planners have different job descriptions and different experience, making it useful to get each individual opinion.

At all times throughout the session at least one person was available for questions about the software prototype and the presentation.

#### 6.4.2 Test results

A total of 12 subjects have cooperated in the presentation and demonstration. Not all of the subjects were able to fill out all parts of the questionnaire. The presentation and demonstration was spread over several days, because of the time needed to complete it, and not all subjects could be present at all times.

All subjects are involved with shipyard planning in their work in one way or another, depending on their exact job description. The subjects were mostly project planners or working in production support, which includes planning for the yard. The subjects were selected from most of the major shipyards in The Netherlands. These yards make a large variety of ships, with some yards averaging ships of 10,000 tons, while some of the smaller yards average around 6,000 tons. Most yards were working on 2 to 4 projects at any one time, completing an average of  $2\frac{1}{2}$  projects per year.

**Background: current support tools** The kind of support in use by the yards varies greatly. Some yards use a project planning tool such as PRIMAVERA [Pri99] or TIMELINE [Tim99] to support planning. These tools are usually used to support both activity and project planning. Capacity planning is sometimes also supported with these tools, but it is more common to see this being supported by spreadsheets. Smaller yards use spreadsheets for all planning, and sometimes even use manual methods such as a planbord exclusively.

Registration of and control over hours worked is usually done in a separate system. Two of the larger yards use custom software to support this process. Other yards use spreadsheets, or process the hours manually.

Table 6.12 on the next page shows the satisfaction of the subjects with their current planning support tools. Yards using PRIMAVERA, and to a lesser extent

TIMELINE, are quite content with the way it supports their planning process. Custom software used for hour logging and reporting can perform well, but as the rating for TVR<sup>4</sup> shows it does not need to be that way. The large standard deviation for TVR is caused by the fact that TVR performs several functions, some of them good, and others quite bad. Spreadsheets seem to work, but clearly lack the domain-specific features. Finally, manual methods of dealing with planning are not appreciated. The total number of observations is larger than the number of planners because the planners were asked to rate their tools on several points, and some planners use several tools together.

Support tool	$N$	$\bar{X}$	$s_{\bar{X}}$
PRIMAVERA	9	5.0	0.0
TIMELINE	5	3.8	1.1
Proprietary system for hour logging	4	3.7	0.96
Spreadsheet	7	3.1	0.9
TVR	3	2.3	2.31
Manual	1	1.0	0.0

Table 6.12: Support tools and their ratings on a scale of 1–5, with 5 being best

**Workspace test results** Two questions were asked during the test to determine an evaluation of each workspace as a whole, each of which could be answered on a scale of 1–5, 1 being worst, 5 being best:

1. Does the workspace allow you to reach the stated goal?
2. Does this workspace appear to be better at it than your current set of tools?

The answers to these questions can be found in Table 6.13 on the following page and Table 6.14 on page 185. Each table shows the sample size  $N$ , the average response  $\bar{X}$ , and the standard deviation of the sample set  $s_{\bar{X}}$ . These numbers provide an indication for the average answer of the planners to each question, and whether there was much difference in their answers. This indication does not provide any statistical confidence, though. From these numbers alone is not clear whether a given workspace yields a positive answer. The

<sup>4</sup>TVR is a proprietary tool custom built for one of the yards.

fact that not all questions have been answered by the same amount of people further complicates any comparisons. Therefor another column has been added to each table. This column contains a 95% confidence interval around  $\mu$ , the mean of the whole population. This gives the confidence that in 95% of the cases the actual mean for all the shipyard planners lies within the interval [FT89].

Table 6.13 contains the results for the first question: “Does the workspace allow you to reach the stated goal?”. The workspaces are ordered according to the lower boundary of the interval. The table shows that two workspaces are clearly rated good, with even the worse case scoring over 4. Furthermore, two workspaces have worst case scores slightly lower than 4. The remaining four workspaces all score between 3 and 4, even when looking at the low end of the confidence interval. From this we can conclude that all the workspaces do allow the planners to reach the goal stated for it, although with varying degrees of success.

Workspace	$N$	$\bar{X}$	$s_{\bar{X}}$	95% confidence interval
Determine construction order	11	4.5	0.5	$4.1 \leq \mu \leq 4.81$
Maintain views	5	4.0	0.0	$4 \leq \mu \leq 4$
Planning engine	8	4.4	0.5	$3.94 \leq \mu \leq 4.81$
Capacity view	12	4.3	0.7	$3.92 \leq \mu \leq 4.75$
Maintain plan versions	10	4.2	0.6	$3.75 \leq \mu \leq 4.65$
Maintain activities	10	4.1	1.1	$3.31 \leq \mu \leq 4.89$
Progress monitoring	8	4.1	1.0	$3.3 \leq \mu \leq 4.95$
Maintain floorplan	9	3.9	0.9	$3.18 \leq \mu \leq 4.6$

Table 6.13: Results for “Does the workspace allow you to reach the stated goal?”

Table 6.14 shows the results for the second question. Again the workspaces are ordered according to the lower boundary of the 95% confidence interval. Looking at the worst case boundaries it is clear that all but two of the workspaces are at least as good as the tools currently in use. Table 6.12 already showed that there is a great difference in which tools planners use, and this is clear from the variance in the results. If the higher boundary of the interval is considered, all but one tool are rated at least better than the current tools. It is not possible to make a more detailed analysis of these results based on what tools planners use because the sample size becomes very small, and because several planners use several tools together which makes it hard to define



specific groups.

Workspace	$N$	$\bar{X}$	$s_{\bar{X}}$	95% interval
Determine construction order	10	4.3	0.8	$3.71 \leq \mu \leq 4.89$
Planning engine	8	4.3	0.7	$3.66 \leq \mu \leq 4.84$
Maintain plan versions	10	4.2	0.8	$3.64 \leq \mu \leq 4.76$
Maintain activities	10	4.0	0.8	$3.42 \leq \mu \leq 4.58$
Maintain floorplan	9	3.8	1.0	$3.03 \leq \mu \leq 4.52$
Progress monitoring	7	3.9	0.9	$3.02 \leq \mu \leq 4.69$
Capacity view	11	3.3	0.9	$2.67 \leq \mu \leq 3.88$
Maintain views	5	3.2	0.8	$2.16 \leq \mu \leq 4.24$

Table 6.14: Results for “How do the workspaces compare to the current tools?”

The workspaces that are rated best compared to the current tools are indeed workspaces that are not available in current planning tools. The first two workspaces are not available at all, while maintaining versions can be done with some of tools, but only in a cumbersome way, and without the ability to easily compare or move parts of a version to the main plan.

The two workspaces that are rated worst are the ones that are most prominently available in current tools. The versions included in SHIPSHAPE did not differ significantly from those tools in terms of functionality, and the SHIPSHAPE implementation was not as feature-rich or complete as some of the tools currently used.

**Variance analysis** Based on the clear influence of the current planning tools on the answers of the second question, an analysis of variance has been carried out. Table 6.15 on the following page shows the results, taking the currently used tool as an independent variable, and the answers to the question how the workspaces compare to the current tools as dependant variables. The table shows the results of a univariate analysis of variance using a General Linear Model [SPS97, Sta99]. For this purpose the planners have been divided into two groups based on the maturity of the tools: those that are currently using PRIMAVERA or TIMELINE, and those using something else.

The workspaces are ordered according to the significance of the finding in Table 6.15. This significance indicates the chance that there is no statistically significant relationship between the group a planner is in and the ratings given to workspaces. The analysis of variance makes it clear that there is a strong relationship between the tool currently being used and the way the Capacity view

Workspace	SS	df	F	Sig.	R <sup>2</sup>
Capacity view	4.848	1	13.091	0.006	0.593
Maintain activities	1.667	1	3.077	0.117	0.278
Determine construction order	1.350	1	2.274	0.170	0.221
Progress monitoring	1.190	1	1.623	0.259	0.245
Maintain plan versions	0.600	1	0.960	0.356	0.107
Maintain views	0.300	1	0.360	0.591	0.107
Planning engine	0.003	1	0.058	0.818	0.010
Maintain floorplan	0.005	1	0.052	0.826	0.007

Table 6.15: How the currently used tool influences the comparison question

is scored. Further analysis shows that planners using PRIMAVERA or TIMELINE score a 2.67 on average ( $s = 0.52$ ), while the other planners score a 4.00 ( $s = 0.71$ ). Similar relationships for the other workspaces are not significant, but for Maintain activities it is also true that the planners using PRIMAVERA or TIMELINE give a lower score than the others.

This effect can be expected. Obviously the difference in comparison should be smaller if a more similar tool is already being used. For those workspaces that are not available in current planning tools, such as Planning engine or Maintain floorplan, there is no difference in the way the workspaces are compared, because there really is nothing to compare it with. As Table 6.14 shows, these workspaces get the higher scores indicating that there is no statistically significant relation between the group the planner belongs to and the workspace ratings given.

**Qualitative questions** For each workspace two qualitative questions were asked. The first question asked whether anything was missing from the workspace. The second asked to describe the workspace in a few brief words, giving an overall impression.

Table 6.16 on the next page contains an overview of those items listed as missing for the workspaces used in the test. Specific suggestions for the prototype and more general comments have been omitted for clarity. This leaves suggestions for *tools* and *materials*. The workspace Maintain views is not included. It was not actually shown in the prototype, but only suggested, and no useful feedback was received with the open questions.

A variety of items are listed by the planners under the heading of missing items. While all of them indicate useful extensions to a planning system or use-

Workspace	Missing items
Capacity view	Historical data; compact support for high-level decisions; automatically indicate earliest possible start date for new projects
Maintain activities	Move work from one department of the yard to another; make cause of impossible actions more clear; group similar activities together
Maintain plan versions	Simple way to try changes and revert them; use standard plans; version control for quality control
Determine construction order	Include standard building strategies; leveling of resources
Planning engine	More insight into planning process; room for own comments
Progress monitoring	Make sure to keep changes to a minimum; making changes in plans needs consensus of whole yard, is not only a technical problem; undo; automatic planning
Maintain floorplan	Handle fixed floorplan items such as cranes; there are many more factors that determine layout

Table 6.16: Missing items per workspace

ful suggestions for improving the prototype, the real question here is whether they constitute missing tools or materials that should not have been missing in the workspace descriptions.

Obviously this is not an easy question, because there is no one true description for a workspace. Some of the planners' suggestions were considered during the design process, but thrown out for various reasons. An example of this is handling fixed floorplans in Maintain floorplan, which was not implemented in the prototype due to time-constraints. Also, some suggestions are implemented, either in the workspace being commented on, or in another workspace. One example of this is the historical data suggested for Capacity view. Historical data is available through the Progress monitoring workspace, though. Another example is to have automatic planning in Progress monitoring. Automatic planning is part of Planning engine, which does have a relation with Progress monitoring.

For the most part it appears that these suggestions are often rooted in seeing only one part of the system at a time in an artificial setting. A small number of suggestions do indicate missed tools in workspaces, though. For instance, including standard building strategies in Determine construction order, or indication of the cause of impossibly placed activities more clear in Maintain activities.

Table 6.17 contains a list of the quick comments planners were asked to give in response to each workspace, giving their overall impression of the workspace. From the comments it is clear that in general the planners are happy with the workspaces as they are presented. Comments such as 'clear' can indicate that no unneeded clutter is present in the workspaces. Also, some of the workspaces are either at the same level or better than PRIMAVERA, which according to Table 6.12 is the best currently used support tool.

Workspace	Quick comments
Capacity view	simple; clear; similar to PRIMAVERA; direct feedback works well
Maintain activities	user friendly, clear, fast to use
Maintain plan versions	handy, good, good support for high-level decisions; would be good addition to PRIMAVERA
Determine construction order	useful; clear; more testing needed with own data
Planning engine	simple; clear
Progress monitoring	good, but someone needs to enter a lot of data; a lot of work; complete; whole organization needs to adjust to this
Maintain floorplan	might not be usable in practice

Table 6.17: Quick comments per workspace

**Additional comments from planners** The questionnaire also contained several open questions and some planners added comments of their own. This section contains a summary. In general the planners were pleased with our work:

*“The GANNT and activity view are very promising.”*

They also indicated that we were not done yet. For instance, we should have tied all the information even closer together:

*“The links between all the different elements are very important, and they need to be used further. For instance, when an activity is selected, it should be very easy to see the related resources.”*

Also, we had not captured the links with the practice in the yard completely, e.g.:

*“The connection with the yard should be better, for instance resources should be sorted in chronological order related to the way a ship is built. Another example is when capacity problems are solved to sort similar activities together.”*

and:

*“The capacity view should contain the tool to outsource work, that is where it is needed. Also, some upper limit on outsourcing is needed.”*

Finally, while having more information available is good in general, it also depends on the goals of the planner:

*“Sometimes there is too much information. For instance, in the capacity view it might be better to use just one tint of color per project so that the relations between the projects become clear.”*

## 6.5 Testing the assumptions

In this section a more structured reflection on the SHIPSHAPE case is presented. To this end the assumptions presented in Table 4.7 on page 129 are discussed. Further reflection on WONDER and on the research questions will be presented in Chapter 7. A summary of this discussion is presented in Table 6.18 on page 198.

About half of the assumptions are either obviously true or false. The other assumptions are harder to qualify. It can be argued that if an assumption is not completely true, then it should be marked as false. This would assume a black-and-white world, though. It makes more sense to look at the discussion of each assumption, and check why it is that the assumption is not obviously true or false. In most cases it turns out that in general, the assumption is true, but that there are a few cases or exceptions or special situations which put the assumption in perspective, although not invalidating it. An example of this is assumption 6 which states that text is a good medium for the model. While this is true, evaluation of WONDER showed that graphics would still be needed, so text can not be used exclusively. In such cases the assumption is marked 'mostly true'. This leads to the following classification:

- true
- mostly true
- undecided
- mostly false
- false

### 6.5.1 Assumptions about the way of thinking

**Assumption 1** *The model described with workspaces conforms to the mental model of the users towards their work.*

This assumption is tested by asking the users about the workspaces in the questionnaire. Overall the workspaces confirm the way the planners think about their work, making this assumption **true**. Three questions support this conclusion.

One question asks whether the workspace is sufficient for reaching the associated goal. The results for this question are shown in Table 6.13 on page 184. Each of the workspaces scores well.

The second asks whether anything has been forgotten in the workspace. The responses are summarized in Table 6.16 on page 187. Only a few of the items listed there are clearly missing from some of the workspaces. The majority of items in the table are additional enhancements or things which are very hard to implement technically. These suggestions do show that the workspaces are in line with the mental models of the planners.

The third question asks for a brief impression of the workspace in a few words. Table 6.17 on page 188 shows these comments. The response of the planners towards the workspaces is quite positive.

**Assumption 2** *Each workspace allows the users to accomplish the goals associated with the workspace.*

This assumption is strongly related to assumption 1. Planners have been asked whether they feel they can reach their goals with each of the workspaces. Table 6.13 on page 184 contains the answers to this question. Clearly each of the workspaces allows the planners to reach the stated *goals*.

In addition, Table 6.16 on page 187 provides an indication whether all the right tools are contained in the workspaces. This is not completely the case, with some items missing from workspaces that should have been there. Most of the suggestions are for items that are secondary to the goal associated with the workspaces, though, so not having them available does not inhibit the work of the planner. Because improvements are possible in selecting an even better set of *tools* and *materials*, this assumption is **mostly true**.

**Assumption 3** *The description of ill-structured work matches the perception of that work by both the domain expert and participating users.*

This assumption tests whether the representations of WONDER give an accurate description of the work that is being described, in particular for the ill-structured parts of the work. The domain expert partially can be a judge of this, because of the need to describe his view in the work using these descriptions. The fact that these descriptions exist provides proof that the assumption is true for the domain expert. Furthermore, as described in Section 6.3.5, the domain expert acknowledges that the descriptions provide a good way to describe the work.

The participating users sometimes had trouble to get a good grasp of how the workspace would translate into support for their workplace. This seems to be caused mostly by a desire of these users to visualize the workspaces, which is specifically avoided while making the WONDER descriptions. Thus this assumption is **mostly true**.

**Assumption 4** *Explicitly allowing ambiguity and inconsistency in the design process is beneficial.*

The review in Section 6.3.4 indicates that allowing ambiguity and inconsistency in the design process is beneficial. The case is clear for inconsistencies. Ambiguity will always be a part of the design process, but allowing it in the representations makes this explicit. One drawback is mentioned on page 174: it can be hard to eliminate all of the ambiguity and inconsistency because this is not mandatory. Therefore this assumption is **mostly true**.

**Assumption 5** *The WONDER representations consist of the right ingredients: workspace, material, action, tool.*

Section 6.2.5 contains a description of the actual use of the three main WONDER representations. In summary, the *action* representation is hardly used, there are some minor changes to the *workspace* representation, and the *material* representation is used as intended. This assumption is **undecided**.

### 6.5.2 Assumptions about the way of modeling

**Assumption 6** *Text is a good medium for a model during the early design phase.*

Text did prove to be a good medium for the representations during this phase of the design process. No significant problems with text were observed during the SHIPSHAPE case. In the evaluation in Section 6.3.5 the design team indicates using WONDER was beneficial. Two observations need to be added to this conclusion, though.

First, there is one point in the design process, described in Section 6.3.1, where text was not a good medium. Instead, diagrams were used.

Second, text may be a good medium, but it can not be the only one. While the description in Section 4.3.2 is not entirely clear about it, it does seem to imply that text is the *only* medium to be used in WONDER's representations. This is in fact what happened initially when using WONDER in the SHIPSHAPE case. However, some of the concepts of shipyard planning are complex, and some visual representation does help quite a bit in gaining understanding. WONDER supports only textual representations, though, as is argued in Section 4.3.2. This caused the visuals that were relevant to live outside of the WONDER representations. At the end of the case study some of them were added to the representation as possible design alternatives as shown in Figure 6.9 on the facing page, but this was too little too late for the case study. This assumption is **mostly true**.

**Assumption 7** *The evolution from free to structured information as indicated in Table 4.4 works well.*

This assumption is **true**. The evolution indicated in Table 4.4 on page 113 works well, as is partly asserted by assumption 4. Also, Table 6.9 on page 172 shows how these changes are made. It should not be a surprise that changes to the *tools* and *materials* top the list. These categories require most of the changes as their descriptions progress from unstructured to structured information.






Option	Possible values	Example
Type of chart	Line Strip	The Strip chart shows capacity allocation as a single line, indicating total allocation with colors: 
Allocated capacity	Show Don't show	Allocated capacity is shown as a colored line in the graph: 
Available capacity built-up	Show Don't show	Show different elements of capacity built-up (e.g. vacation, sick people, scheduled maintenance) in graph: 

Figure 6.9: Visual addition to WONDER representation (detail)

**Assumption 8** *No crucial information related to the early design process is kept outside of WONDER.*

This assumption is **false**. Some important information was kept outside of WONDER descriptions. This has happened for the most part at the start of the design process and towards the end of the process supported by WONDER.

In the beginning, before using WONDER but after the analysis process, several diagrams were used to communicate the sometimes complex relations between tasks, departments at the yard, and workspaces. Examples of these diagrams are shown in Figures 6.4 and 6.5.

The workspace diagrams do have a strong link with the workspace representations, and in particular with the *relations* section in it. A diagram, however, works much quicker and easier at this early stage in the design process, also because not much information is attached to each workspace yet.

Towards the end of the design process supported by WONDER, the need for additional information also becomes bigger. Figure 6.9 on page 193 shows how an illustration is included in a WONDER representation, even though this is not normally done. This is also discussed as part of assumption 6.

Finally, knowledge about planning and scheduling in general, and within shipyard planning in particular, was needed to correctly interpret the descriptions. This information can not be described as part of the representations, so additional documents, along with site visits and discussion was needed to provide a sufficient context in which the representations could be fully understood.

### 6.5.3 Assumptions about the way of working

**Assumption 9** *The design activities described in Section 4.4 are a correct representation of the actual use of WONDER.*

This hypothesis tests whether the task structures in Section 4.4, more specifically in Figure 4.6 to Figure 4.9, present an accurate description of the use of WONDER during design. Section 6.3.6 compares actual use with the task structures presented in Section 4.4.

It is clear that the task structures do not provide a complete match. A number of smaller and larger differences are described in Section 6.3.6. For instance, often clustering does occur, where several elements of a workspace description are added at the same time. Also, the order in which activities are carried out does not always match the task structures. This is not surprising, and is in fact to be expected [PC86].

All of these differences do not contain a fundamental deviation from WONDER's way of working. Some can be attributed to the task structures being an ideal representation, where for instance an iteration goes on until *all* uncertainty is solved. In practice this is not always done because it may be too time-consuming or because it is not appropriate at the time.

There is one other issue that is related to the description of the design activities. It has to do with responsibilities and ownership. By not making this explicit in the way of controlling in Section 4.5, this can easily cause problems during the design process, for instance because it is not clear who should update what parts of the descriptions.

Finally, it turns out that no sweeping changes have been made to the design once the first round of activities had been done. Most likely this is because these sweeping changes had already been carried out during construction and evolution of the diagrams shown in Figure 6.4.

In summary the design activities presented in Section 4.4 do provide a good approximation of the actual use of WONDER, but not a perfect representation, making the assumption **mostly true**. Given that using some parts of WONDER is a ill-structured task, the latter does not come as a surprise.

**Assumption 10** *The different design team members can all work with WONDER.*

This hypothesis tests whether the WONDER descriptions can be used by all design team members. This usage can be separated into two aspects. First of all a distinction needs to be made between working with WONDER *descriptions* in general, and working with the current *support tool*. Second, a distinction

can be made between just reading or browsing the descriptions, and actively changing them.

The current example system for computing support is described in detail in Chapter 5. Its usability is evaluated in Section 5.3.1, where it becomes clear that ease of maintenance for all design team members is a problem, due to the implementation of the example system. This is a property of the implementation, however, and not of the WONDER descriptions. The design team members did all actively work with the descriptions, and made changes either on paper, or in joint editing/reading sessions. The changes proposed in Section 5.3.2 will alleviate most of the problems, and make it easily feasible for all design team members to edit the descriptions. Based on feedback from the design team this should not be a problem. This makes the assumption **mostly true**.

#### 6.5.4 Assumptions about the way of controlling

**Assumption 11** *The roles defined in Table 4.5 on page 121 match actual use of WONDER.*

The evaluation of the design team during the SHIPSHAPE case is described in Section 6.3.5, and the assignment of roles is summarized in Table 6.7 on page 169. Drawing more general conclusions is hard in this case because the design team is small, resulting in a lot of sharing of roles between team members.

Two observations were not apparent before. First, having several team members with the same background, in this case software engineering, does pose problems in that the design process is likely limited by this. Second, sharing roles can cause a lot of problems because parts of the communication between different roles is no longer explicit.

In summary there is not enough information in the case to draw a conclusion either way; the assumption is **undecided**.

**Assumption 12** *The domain expert shall be the team leader.*

This assumption is **true**. As discussed in Section 6.3.5 this was not strictly adhered to in the SHIPSHAPE case, but even so the observations made during the case indicate that the domain expert is in the best position to be the team leader.

### 6.5.5 Testing the way of supporting

**Assumption 13** *Computer assistance is needed to work efficiently and effectively with WONDER.*

This assumption is **true**. Computer assistance is a crucial part of working with WONDER. It makes working with WONDER more efficient. Assumption 14 indicates that working with WONDER is worth the time at this moment. If additional time would have to be taken to do things manually which are now supported by computer, this would not be the case. The computer assistance takes care of many of the small details, and having a single repository for the design representations which can also be shared over the network is quite helpful. Even so, Section 5.3.1 indicates that additional improvements can be made, making the use of WONDER even more efficient.

Computer assistance also makes working with WONDER more effective. For example, tracking inconsistencies becomes much easier with computer assistance, which is important as is argued in Assumption 4. Navigation also becomes easier. Finding the right information quickly is also easier, for instance through a search engine. Finally, browsing the information helps to develop a better mental image of the whole of the design, leading to better design decisions.

### 6.5.6 Other assumptions

**Assumption 14** *The time taken to use WONDER is worth the results it yields.*

This assumption questions the value of WONDER in terms of the investments it requires. An evaluation of this assumption is very hard, because both the cost and the benefits are difficult to quantify, and because the use of WONDER would have to be compared to methods which would yield the same general results. Notwithstanding these issues, it is at least possible to say something about the costs and benefits of using WONDER.

One indicator of the cost how the design team judged the effort of using WONDER in the SHIPSHAPE case. The comments portrayed in Section 6.3.5 seem to indicate that this is the case: WONDER is perceived to be beneficial, or at least not hindering work progress.

As for the benefits, the use of WONDER should have a tangible effect, either by helping to structure the design process, or by serving as input to the remainder of the design process. Certainly use of WONDER is a big improvement over the situation described in Chapter 2, in the sense that WONDER provides much more structure at the level that is suitable in the early design process. Also, the

design representations were directly useful in developing a prototype environment, as described in Section 6.4.1.

In summary, while it is impossible to test this assumption in objective terms within the context of this thesis, the design team did have the perception that using WONDER was worth the effort. This assumption is **true**.

**Assumption 15** *Compared to tools currently in use a WONDER workspace description provides at least the same level of support.*

This assumption is tested with a question in the questionnaire. This question asks whether the proposed workspace seems to provide a better work environment than currently is use. The scores for this question are shown in Table 6.13 on page 184. Although in general the proposed workspaces seem to perform a bit better than the current work environment, some caution is in order with these results.

The results of this question depend strongly on the tool already in use at a yard. Table 6.12 on page 183 shows that some tools already provide a high level of satisfaction, and when the SHIPSHAPE workspaces are compared to these tools the score often goes towards favoring the current tools. This might happen because, while the workspace offers something comparable to the current tools, the current tools are simply more familiar.

Overall, it can be concluded that the proposed workspaces provide at least the same level of support as the current tools, thus this assumption is **true**.

### 6.5.7 Additional observations

Several observations on using WONDER in the SHIPSHAPE case can be made apart from the assumptions. They are presented below. Further reflection on WONDER is presented in Chapter 7.

**Actions** Clearly the *actions* are not a very useful part of WONDER, as is asserted in Assumption 5. In fact, if the actions would be taken out of the evaluation of this assumption it would be ranked as mostly true. Actions are simply design details which are too low-level at this stage of the design.

**Multiple types of workspaces** One surprise is that several distinct types of workspaces surfaced, as discussed on page 6.10. The placement of the different types of workspaces in the workspace hierarchy depicted in Figure 6.2 makes it

Assumption	Result
1 The model described with workspaces conforms to the mental model of the users toward their work.	True
2 Each workspace allows the users to accomplish the goals associated with the workspace.	Mostly true
3 The description of ill-structured work matches the perception of that work by both the domain expert and participating users.	Mostly true
4 Explicitly allowing ambiguity and consistency in the design process is beneficial.	Mostly true
5 The WONDER representations consist of the right ingredients: workspace, material, action, tool.	Undecided
6 Text is a good medium for a model during the early design phase.	Mostly true
7 The evolution from free to structured information as indicated in Table 4.4 works well.	True
8 No crucial information related to the early design process is kept outside of WONDER.	False
9 The design activities described in Section 4.4 are a correct representation of the actual use of WONDER.	Mostly true
10 The different design team members can all work with WONDER.	Mostly true
11 The roles defined in Table 4.5 match actual use of WONDER.	Undecided
12 The domain expert shall be the team leader.	True
13 Computer assistance is needed to work efficiently and effectively with WONDER.	True
14 The time taken to use WONDER is worth the results it yields.	True
15 Compared to tools currently in use a WONDER workspace description provides at least the same level of support.	True

Table 6.18: Assumptions about WONDER and their results

clear why these three types of workspaces emerged. They are the unconnected workspaces, and the nodes and leaves of the hierarchy.

**One person in charge of the design representation** During the evaluation of Assumption 10 it is noted that not all design team members did edit the WONDER descriptions directly. This may actually be a good thing. When the representations are edited by one person there is a better chance that terminology will be used consistently [Joh00].

**Overall quality of the design** An effort directed specifically at producing an actual planning tool for shipyards may result in a better overall design. The specific focus of this research on the use of WONDER for conceptual design has caused some design activities to not get the attention they should have gotten. For example, better contextual design with the actual planners would have shown earlier in the design process that the views did not get the right place in the design.





## CHAPTER 7

---

### Conclusions

---

The questions that are posed in Chapter 1 will be answered in this chapter. In between much has happened. An exploratory case study and a literature study have resulted in a theory on the design of interactive systems called WONDER. WONDER has been used on the large real-world problem of shipyard planning in the SHIPSHAPE project, which allows answering the research questions asked in Section 1.3. The assumptions drafted in Section 4.8.2 and discussed in Section 6.5 provide the linking pin for this. This chapter also contains a broader reflection on the research presented in this thesis. It is concluded with a look toward the future.

### 7.1 The thesis in a nutshell

In this section the major line of argument is brought together. First, WONDER is summarized in the light of the research leading up to it. The center of the argument are the assumptions associated with WONDER, and tested in the SHIPSHAPE case. These are summarized, and then reflected back to the research questions.

### 7.1.1 WONDER and its roots

Chapters 2 and 3 provide the roots for WONDER as it is presented in Chapter 4. Together they provide a combination of theory and practice aimed at exploring the support for the design of interactive systems for ill-structured work.

Working on a practical real-world design problem like DIANA resulted in a number of important conclusions. First, support for conceptual design is certainly not mainstream, not even for structured work. This is a big bottleneck, and guidance for doing this part of the design is lacking. Second, both task analysis and grounding the design in the work of the users did help, but neither of them proved to be an ideal solution. Task analysis proved to be quite useful for the structured parts of work, thus giving a good framework for the ill-structured parts. Grounding the design in the actual work of users proved to be very helpful in understanding the real problems to be solved, but did not in itself help to set a direction for the design. Third, the use of graphical interface elements, while sometimes helpful to get across certain design aspects, has a tendency to focus on the details instead of the big picture.

Additional study of the available theory did not lead to a single solution for this problem, but it did point out a number of recurring themes. The importance of the user context, the need to deal with ill-structured work, the usefulness of design tools, and the balance between rigidity and flexibility are all important themes during the early design process. Based on these issues and on other literature picked up during the study, a set of requirements and suggestions has been formulated.

These requirements and suggestions are picked up in Chapter 4. In this chapter a theory called WONDER is presented which fulfills most of the requirements and suggestions, at least in theory. This workspace-oriented design representation introduces a way of thinking about interactive systems: that they are built up from individual places, in each of which a particular goal can be reached. By dividing the interactive system into these workspaces, the overall structure of the system can be mapped out. By describing each workspace carefully in terms of the materials and tools needed to reach the goal given the particular context, a precise representation can be given without resorting to specifying the, possibly ill-structured, tasks.

### 7.1.2 Assumptions

The presentation of WONDER ends with a set of assumptions about WONDER in Section 4.8.2. The purpose of these assumptions is to determine whether WON-

DER satisfies all the requirements and suggestions presented in Section 3.5.3. The assumptions are used in Section 6.5 to reflect on the results of the SHIP-SHAPE case study. Table 6.18 on page 198 summarizes all the assumptions and their results.

Of the fifteen assumptions, six are true and six are mostly true. This shows that for the most part WONDER does live up to the requirements on which these assumptions are based, even though in some cases additional improvements can be made.

Assumption 8 is false. Crucial design information was kept outside of WONDER during the design process. Not all the information needed during the design process is captured by WONDER's representations. This is not listed as a requirement, though. The assumption can be traced back to the discussion about design tools. There is an implicit assumption that the design tool should cope with all aspects of the early design phase, but this need not be the case, as long as it is clear what is and what is not covered by WONDER. Investigating this assumption has made that clear.

Assumption 5 is undecided. This classification is due to the *action* concept, which was not used much in the SHIPSHAPE case. Looking at the *workspace* and *material* concepts alone this assumption would be classified as *mostly true*. The *action* concept is simply not the right level of abstraction at this phase of the design process.

Assumption 11 can not be answered because the SHIPSHAPE case has not yielded enough information on this issue. The constituency of the design team, and the way the roles had to be assigned make it quite hard to determine exactly whether the roles and responsibilities as they are presented in Table 4.5 on page 121 are right. In general it seems as if in the SHIPSHAPE case these roles and responsibilities mostly matched those presented in WONDER, but because each design team member had several different roles to take care of this can not really be determined. Further study is needed to provide a clear answer, preferably in a situation where each person has only one role.

### 7.1.3 Reflecting the assumptions onto the research questions

In the previous section it has become clear that WONDER does satisfy the requirements posed for it. This gets us back to the research questions. Does WONDER also provide answers for these questions?

*How can we formulate the design activity for interactive systems which facilitate ill-structured work?*

WONDER is the answer to this research question, as it provides a formulation for the design activity with a focus on ill-structured work. This research question is answered positively in this thesis in Chapter 4.

*How can ill-structured work be described explicitly during design without reverting to interface components?*

The concept of a workspace with materials and tools is the answer to this question. It could be argued that this is not really an explicit description. In Section 3.2.4 it is made clear that an explicit description of the actual ill-structured work itself is not possible. If it were, then the work would not be ill-structured. The materials and tools in a workspace are used to describe the effects and possibilities for the ill-structured work as closely as possible, without describing the actual work itself. The materials signify the information used as input and output of the work, and the tools signify operations on these materials that are needed as part of the work.

In WONDER's representations materials and tools are still somewhat abstract entities. No decision has been made at this point how to represent them in the user interface. At this point it is only clear that they need to appear in one way or another. No interface components are used in WONDER.

*How can it be ensured that the formulated design activities and representation of ill-structured work provide a usable, workable, and fitting solution toward the design of interactive systems facilitating ill-structured work?*

*Usable* in this question means whether the design activities and the design representation can be used by the design team. Assumption 10 asserts this: all the design team members could work with WONDER. Another positive indicator is assumption 9, which asserts that for the most part the design activities as they are described in WONDER were followed during the SHIPSHAPE case.

Whether WONDER is *workable*, i.e. whether the activities and representations fit into the design process, is addressed by assumption 14, which asserts that using WONDER is worth the time. Furthermore, the WONDER representations were a good starting point for the creation of the prototype described in Section 6.4.1.

The final part of this research question asks whether WONDER is *fitting*, i.e., whether the activities and representations yield the desired results. Assumptions 1, 2, and 3 assert that WONDER yields the results it should accomplish. Also, assumption 15 indicates that WONDER yields results at least similar to other methods.

In summary, it appears that WONDER does fulfill the research questions. Additional improvements are possible, as outlined in Section 7.3, but overall WONDER provides a good answer to the research questions of this thesis. The use of WONDER on a real and complex design problem has given a lot of insight into the *actual* use of WONDER, and in this way it has provided a very valuable translation from a chapter full of theory to the practical problems experienced in the trenches of interactive systems design.

## 7.2 Reflection on the research

This research looks at a small and well-defined portion of the world so that the research can be conducted. However, the results and findings should not be seen as separated from our real world. This section attempts to define this broader context by describing the consequences of this research, by providing a broader look at the research, and by revisiting the paradigm underlying the research questions.

### 7.2.1 Consequences of the research

**WONDER** Evaluation of WONDER can focus on the principles behind it, or on the tool itself. The principles are discussed in the next paragraph. WONDER as a tool could need additional improvements before being used by design teams for interactive systems, as discussed in Section 5.3.2. This fits with the focus of the research. The whole purpose of creating WONDER as part of this research was not to create the ideal design tool, but rather to validate principles and ideas behind it, and to allow exploration of the early design process. Future research might want to extend WONDER to look at additional principles, or determine how these principles can be incorporated best into a design tool for interactive systems designers.

**The principles behind WONDER** It is much more important for the *principles and ideas* behind WONDER to have impact. These principles and ideas all relate to the degree of control we want and need to have. It is easy to not want to have any control, any responsibility, but this also leaves us without accountability and ultimately without any sense of accomplishment. Gaining full control provides the opposite, usually becoming harder as the target to be controlled comes in sight ever more closely, but never fully, under control. Both

approaches leave much to be desired, and the real challenge is not to attain either one of them, but rather to find the precarious balance in the middle.

This is done in WONDER by acknowledging that ambiguity and inconsistency are normal elements of the design process. Therefore they are given a place in the design representations, even though the goal in the evolution of these representations is to get rid of ambiguity and inconsistency as much as possible. Furthermore, WONDER shows that more structure can be used to make the early design process more explicit.

One strong idea behind WONDER is that the people who will end up using the interactive systems are valued more. This is expressed by investing more effort into finding actual solutions for ill-structured work. Too often this is not done. Instead the work is structured to death, leaving people fighting a system instead of working with it. Or the designers throw their hands in the air, decide on a way to cram all the functionality into a system which looks nice, and hope for the best, thus leaving people without many clues as to how to be productive.

### 7.2.2 A broader look

**WONDER's aim** WONDER is primarily aimed at designing large and complex interactive systems which support ill-structured work. The benefits of using WONDER really only pay off if the design becomes too large or complicated at this stage of the design process to keep in one's head all the time. This is not a design flaw, but rather a consequence of some of the decisions leading up to the design of WONDER. In light of this aim it is interesting to see how WONDER deals with recent developments in creating interactive systems.

**The Web** It certainly is clear that the WWW more and more is becoming an important interface for interactive systems. Sometimes it is a front-end to talk to legacy terminal-based systems in the back-office over the intranet, other times an easy way to communicate directly with your bank's systems for stock trading over the internet. At any rate the web provides a flexible way to unlock all kinds of systems and processes in a somewhat uniform way. Somewhat, because what is shown in the web browser's window is often anything but uniform.

This would not be a problem if the WWW would simply be used as an implementation platform along with for example back-office systems to create a single interactive system. While this is done on occasion, the trend is to cre-

ate applications and environments based on all kinds of components offered by Application Service Providers. Using WONDER as a design tool would be very hard in these cases. For instance, the formal design process often gets distributed, sometimes even down to the individual users who have to select those components they think they need to use.

At first sight this might be interpreted in a positive way, because people can select their own workspace in detail. However, two other things present in WONDER are not being dealt with in this case. First, the activities in WONDER are separated from the workspaces because the activities provide the communication between different workspaces. If these materials are not uniform between workspaces, even if used in different ways in different workspaces, use of the system will become very confusion or even impossible. Second, actions would warrant a more prominent role than in the SHIPSHAPE case. When components from different sources are being combined, it is no longer logical to assume that they will match well without any explicit effort. Standardizing the basic actions will help to create a more usable system.

### 7.2.3 WONDER and the underlying paradigm

Section 1.3 contains a paradigm which underlies the research questions posed in the same section. Some issues regarding this paradigm have already been discussed in Section 7.2.1. This paradigm is:

*An interactive system should be designed as a whole; the leading perspective during design should be that of its users.*

The paradigm hinges on two thoughts: that interactive system design is a holistic practice, and that the perspective of the users should always be leading during design. WONDER embodies these two thoughts. The actual start of the design process is carried out within WONDER's context, shaping all of the rest of the design, including the technology needed to implement it. The latter is accomplished, for instance, by the definition of the materials, which will often lead to a similar data model. The focus on workspaces takes a strong perspective on the people who will need to use the system. It does not put them in control right from the start of the design process, but that is not what the paradigm is about. It also does not mean that the user perspective should always win, because there are many more viewpoints which need to be taken into account.

### 7.3 Future research

The research presented is also another starting point for additional investigation into the complex and interesting field of interactive systems design. Certainly enough starting points for additional research can be found in this thesis.

**Extending WONDER** While WONDER's representations worked well enough, using only text is not sufficient, so WONDER's representations should be extended to also include graphics where appropriate. Perhaps other media should also be included. While ruled out because of the technical requirements at the time, in 2001 both audio and video clips have become a possibility also. Research into their usefulness to augment the design process would be useful. For instance, would short video clips of the circumstances of real planners trying to reach some goal not be a better description of the context than a text field?

Task Analysis currently has no place in WONDER, apart perhaps from the goal hierarchies. For those parts of the system that support well-structured work, task hierarchies might be useful. They would give the design a bit more stability and a bit less ambiguity, which is not a bad thing once the workspaces stabilize. The tools in particular should lend themselves to a description of tasks to be carried out with them. Whether or not this would be a useful extension would have to be investigated.

**Using WONDER** Also, a re-evaluation of WONDER based on the SHIPSHAPE case would be in order. For instance, is the *action* really superfluous at this stage in the design process as it would seem, or does it make sense when WONDER is applied to another case? The task structures also need another look. But to make good judgments for any of these possible changes, it is really important that WONDER is used in several additional cases. Only then will the real strengths and weaknesses become apparent.

Furthermore, as the first pages of this thesis already commemorate, computing devices become smaller and more ubiquitous all the time. With network developments such as Bluetooth all these devices can form a single 'workspace' in whatever constellation is useful at the time. Will WONDER's design concepts become stronger because of this, or will the way of thinking break once the interactive system is moved off of the desktop?



**Redesigning WONDER** WONDER as it is now has been put together based on analysis. The research instrument which allows working with WONDER has not been designed rigorously. One important thing which could lead to new insights is to look at the needs of real world design teams more closely, in light of the lessons learned with WONDER, and the principles underlying it.

**Comparison with Contextual Design** The development of Contextual Inquiry into Contextual Design, which happened in parallel with the development of WONDER, is also interesting [BH98]. In particular the *User Environment Design* model has a lot of things in common with WONDER. A comparison of the features and names in both models would be interesting, as would the use of both WONDER and Contextual Design on the same case.

**The design process after WONDER** Another point of investigation is what the design process should look like after initially using WONDER. Section 4.7 makes some allusions to scenario-based design, the use of prototypes, or the use of design space analysis. Which one works best, and which adaptations can be made to WONDER is not clear at this point. Scenarios have already been used in progressive design [GCR98], and it would be interesting to see how this matches with the evolution within WONDER's representations.

## *CHAPTER 7. CONCLUSIONS*

---

# APPENDIX A

---

## Structure and operation of DIANA

---

This appendix describes some of the structure and operation of DIANA, providing background to Chapter 2.

### A.1 Overview

The existing implementation of DIANA is batch-oriented. Users create a number of plain text files which describe the models, meshes, constraints, and analysis to be run on those models. These input files are fed into DIANA, which analyses the files and passes them on to the relevant internal modules. These modules carry out the actual work.

An overview of the user environment can be seen in Figure A.1 on the following page [TNO91]. Figure A.2 on page 214 shows the relations between the user, the system, and the files. The user creates two types of files: input files (.DAT) and command files (.COM). The user will use manuals and possibly a pre-processor to prepare these files. The input files describe the model, mesh and additional constraints. The command file describes the computations which need to be carried out and their parameters. Each of these files is parsed by DIANA in turn, and processed by the appropriate modules. DIANA

can use two types of intermediate files. The internal database FILOS (File Organization System) holds intermediate results, meshed models, etc. Optionally DIANA can read and write so called 'neutral files'. These neutral files are written in an open specification language based on the IGES standard, and can be used to exchange data with other software packages.

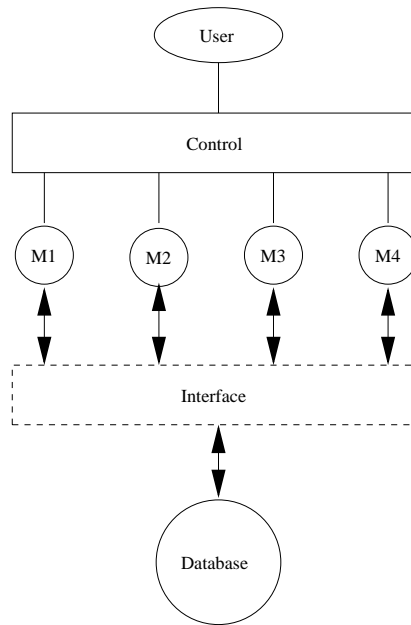


Figure A.1: DIANA's user environment

DIANA can generate two types of files: output files (.OUT) and plot files (.PIC). The output files contain the textual results of all the modules which were run during the session. This output contains error messages, warnings, tables with numerical results, etc. The plot files contain numerical results as well, but in a format which can be visualized with an appropriate program.

Pre- and post-processors are external programs which can be used with DIANA. Pre-processors are used before DIANA calculations are made. They support creating a model, mesh, and associated constraints. Some pre-processors allow the user to create such a model interactively through a graphical interface. These model definitions are then imported into DIANA through the neu-

tral file interface.

Post-processors serve a similar goal, but they operate on computational results from DIANA. Post-processors can be used to do further calculations based on DIANA's results, but most often post-processors serve to visualize the results in some graphical way.

## A.2 Working with DIANA

In this section some comments are presented on the consequences of the architecture of DIANA. This is followed by some examples of input files.

**Architecture** The control layer in Figure A.1 may suggest a strongly controlled system, but this is only true when it comes to controlling the running of modules. All of the application logic is contained in the modules. This leads to a very decentralized system, where each module has complete control over what it does. The file store is used to communicate between different modules. For instance, one module calculates a value and drops it into the file store. A while later another module picks that value up from the file store, and performs additional calculations on it, again dropping the results in the file store.

This architecture makes sense in a batch system, as explained in Chapter 2. It also makes sense when the development of DIANA is taken in consideration. DIANA also serves as a framework for fundamental research into finite element analysis calculations. Using this modular structure, individual researchers can work on their module without bothering other parts of the system. Once the module is done, it can be dropped in the working system, again without many dependencies on other parts of the system. This provides a very flexible development environment.

**Input files** An example of the effect of this modular system approach for the end user can be found in the input files in which a calculation is configured. The input files have a very specific format, which is described in an extensive set of manuals. The syntax description contains many optional, mutually exclusive, or mandatory choices for each of the commands. Together, these different options allow a wide variety of input, while still being very strict about which combinations are valid. This section contains examples of load calculation.

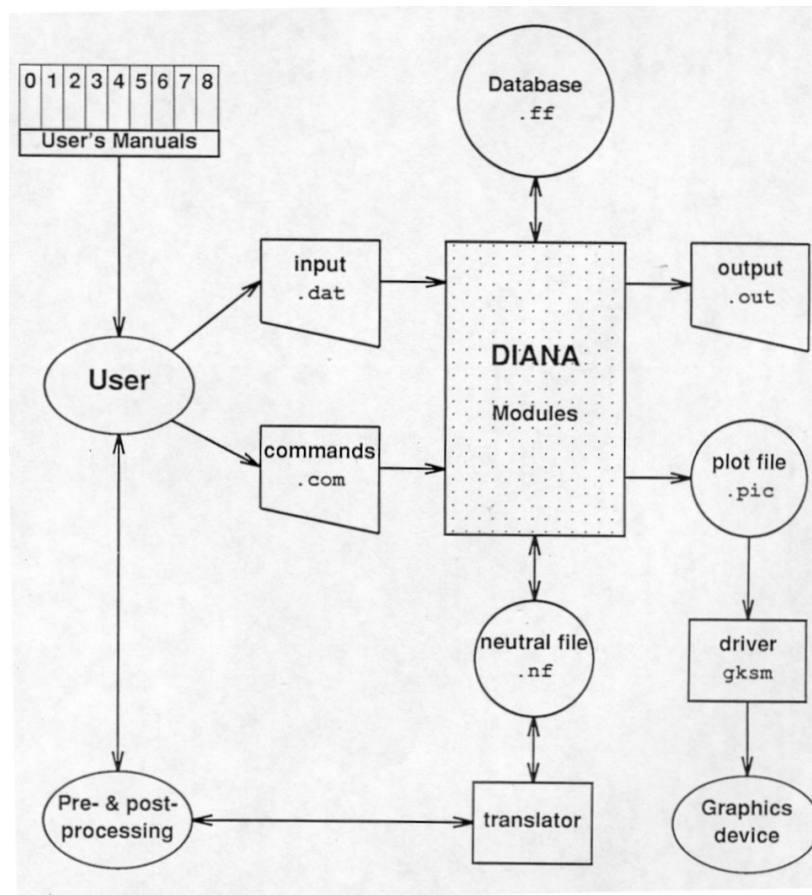


Figure A.2: DIANA program structure

**Setting up a load calculation** The first two figures show setting a load calculation. This indicates how a load will be calculated. Figure A.3 contains a syntax description for setting parameters for a load calculation. Figure A.4 contains an example of these settings.

```
*LOADS

[INITIA [MC=mlosetn] ]

[SETUP]

[SELECT _____ / ] ...

    LOADS losetsn...

    NODES nodesng...

[PRINT{.SXRD}{_____} ]

    NODES{.NCD}
```

Figure A.3: Syntax specification for load calculation

The description shows some of the complexity of the syntax used in DIANA's input files. The order in the input file is fixed, and can not be changed. Square brackets indicate optional steps. Each step represents a segment to be executed, and possibly its arguments. For instance, the `INITIA` (initialize) command can take an optional argument which denotes the maximum number of load sets. The `SELECT` segment has a mandatory argument, which can be either a `LOADS` or `NODES` parameter. The ellipsis denotes that this option can occur several times, while the slash indicates the end of the arguments for that option.

The example in Figure A.4 does not look very complicated. The calculation is initialized with 10 load sets, then it is set up, and then a number of loads are selected and calculated for a number of nodes. Finally the results for this calculation are printed. It is not hard to think of a graphical user interface for this input file. When looking back at the specification in Figure A.3, however, it becomes clear that the input file could have looked different as well. Keep in mind that this is a very simple example. Much more complex input files are needed for real-world calculations, using many more of the options. Imagining a graphical user interface for 20 options, each of which is optional, and some

```
*LOADS
INITIA MC=10
SETUP
SELECT LOADS 1 3 5 /
SELECT NODES 20-60 /
PRINT.S NODES
```

Figure A.4: Example specification for load calculation

of which are mutually exclusive, becomes much harder already.

**Defining the load** To complicate things further, certain fields are defined based on column position in the input lines. One, two, or three fields can be used on a single line, and each fields must start in the proper column for that field. As an additional constraint, each line has a maximum width of 80 characters. Figure A.5 describes the input format for a load specification, while Figure A.6 shows an actual example of it. The LOADS specification has three arguments per (range of) elements. These three arguments are placed in three fields, starting at columns 1, 6, and 13.

```
'LOADS'

ELEMEN [MA=maxelon]

%1____5 6____12 13____80

elemn lodnamw loddatt?

%1____5 6____12 13____80

/ elemsng... /

lodnamw loddatt?
```

Figure A.5: Syntax specification for load specification

The manuals contain many syntax specifications for specific input. Examples of such specifications are shown in Figure A.3 on the preceding page and



Figure A.5 on the preceding page. Dependencies between the different specifications are not detailed, however. For instance, when specifying the loads computation, either load sets or nodes can be selected. It seems obvious that those nodes need to exist, but this requirement is not listed in the manuals. While this particular example is a trivial example, more complex dependencies can also exist. The only place where they are specified reliably is in the source code for each module.

```
'LOADS'  
  
ELEMEN  
      8  EDGE    ETA1  
        FORCE    10. 15.  
  
/ 9-32 /  
      EDGE    ETA2  
        FORCE    25.
```

Figure A.6: Example of load specification

Thus, changing a module has effect on what is a valid input file. If that changed module is called in the input files, then it will influence what is valid syntax. Including all of this flexibility into a graphical user interface is possible, but it also makes it very hard to provide enough support for what the user actually tries to accomplish.

One example of this came out of the interviews with users. Users like to use the visualization loop, where they iterate between making changes to the model and visualizing the results of these changes. This can only be supported well when changes to the graphical output can be made easily and quickly. In DIANA, however, the visualization module, like all other modules, is controlled by a DIANA input file. Even if the same FILOS image can be used for each visualization, the user still needs to modify the input file, run it through DIANA, and pass the output to the visualization system for even the smallest change, such as a small change of viewpoint. This roundabout way totally destroys the iterative nature of the problem solving process, and will result in less visualizations being made, even if this degrades the problem solving process. Some interviewees in fact told us they did not make as many visualizations as they would like for exactly this reason.



## APPENDIX B

---

### PSM notation

---

Figure B.1 on the following page contains an overview of the graphical conventions of the Predicate Set Model (PSM) modeling technique. [tH93].

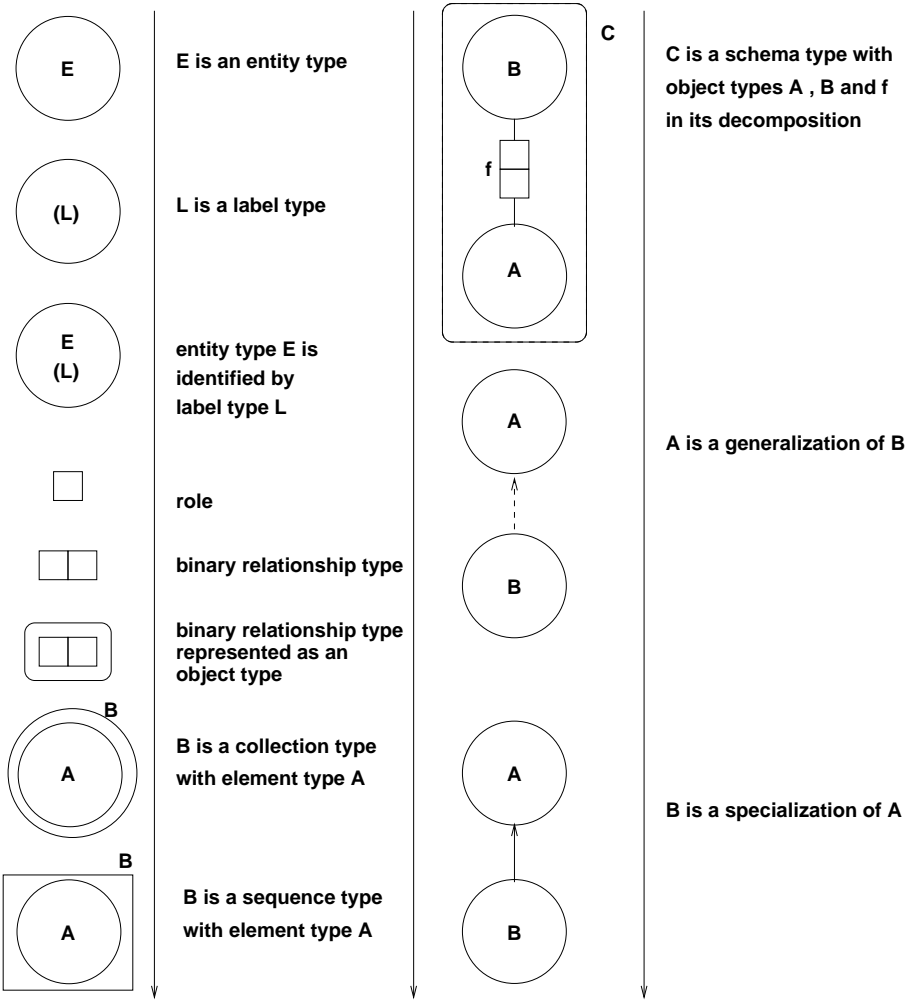


Figure B.1: PSM Graphical notation

## APPENDIX C

---

### WONDER representations for SHIPSHAPE

---

This appendix contains examples of WONDER representations taken from the design for SHIPSHAPE. A more complete and interactive version can be found online at <http://degraaff.org/thesis/shipshape/>. Chapter 6 contains an overview of the SHIPSHAPE case. Please refer to that chapter for additional information and context. The WONDER representations are discussed in Section 4.3.

#### C.1 SHIPSHAPE example workspace

##### *Maintain scopes*

<b>Goal</b>	Many planning operations need only to work on a particular subset of the complete plan. Such a subset is called a plan scope.
<b>Context</b>	A scope is a small subset of a particular plan domain.
<b>Enclosed in</b>	Maintain plan
<b>Contains</b>	(none)
<b>Works with</b>	Maintain plan versions

### *Materials*

<b>Plan scope</b>	The available scopes within a plan domain
<b>Plan elements</b>	The plan elements which make up a particular scope: activities, resources, skills, projects, time window
<b>Interaction</b>	The interaction takes place with two different points of focus: the collection of current scopes (at which point their contents is not of paramount importance), and the region of a plan which will become a new scope. In this case the contents of the scope is very important, and should be visible.

### *Tools*

<b>Select scope elements</b>	select ?: all elements which should be part of the new scope, such as Activity, Resource, Skill, Project, Time. Elements can be selected using several selection criteria. Exact criteria currently not specified yet.
<b>Create scope</b>	create ?: a new Plan scope, given the selected elements. When creating a scope elements or constraints (such as a time window) can be added at will. The following default should be possible: Empty scope; elements limited by time-window from higher plan domain (e.g. project); elements with available capacity at higher plan domain level; elements with available capacity at particular resource level; elements with unallocated capacity.
<b>Delete scope</b>	delete ?: selected scope (but, obviously, not the elements within it).
<b>Show scope elements</b>	Show the scope elements in the plan domain. (This option depends on a standard plan domain view, which currently is not being designed. Note that views will also be able to show scope elements.)
<b>Join scopes</b>	Join two scopes, and create a new scope with all elements from both original scopes.
<b>Intersect scopes</b>	Create a new scope, which contains the common elements of both original scopes.

<b>Add/delete elements</b>	Add elements to scope, or delete them from the scope.
<b>Update scope contents based on specification criteria</b>	Re-apply the specification criteria to update the elements in the scope.
<b>Browse scope elements</b>	Browse through elements of a scope (e.g. with a list). Also show context of elements (e.g. in view)
<b>Scope status</b>	Show the status of the activities in the scope, using the activity panel.
<b>Remarks</b>	We need some concept to denote subsets of the complete plan. These subsets are used throughout planning, in a variety of ways. The plan scope will be this concept, but it needs further definition before it can be fully applied in the design. A better name for a plan scope might be a region.

## C.2 SHIPSHAPE example material

### *Plan scope*

<b>Description</b>	A plan scope is a selected part of a plan. It remains within a single plan domain, but selects a number of activities, resources and skills within a time window in this plan domain. A plan scope is used to limit the working area for other planning tools within a plan domain.
<b>Current Example</b>	No plan scope is being used. A plan scope which contains all welding activities for the Hall 1 resource.

### *Associations*

<b>Project</b>	A project is implicitly available through its activities, but it might also be used directly as a shortcut.
----------------	---

## APPENDIX C. WONDER REPRESENTATIONS FOR SHIPSHAPE

---

### *Attributes*

<b>name</b>	Identification only
<b>date created</b>	Set by system; informational only.
<b>time window</b>	Time window between two dates. Only elements which fall within this time window are shown.
<b>selection criteria</b>	Specific selection criteria, such as projects matching X, activities matching Y, etc.
<b>follow changes in selection</b>	When changes occur in the plan, a mismatch between the selection criteria and the actual elements may occur. This option indicates whether the elements should always be according to the selection criteria, or only initially. Note that this option might be very expensive, computing-wise!
<b>Activity (1-600)</b>	The activities which are part of this list. One or more sets of connected activities.
<b>Resource (??)</b>	One or more resources.
<b>Skill</b>	One or more skills.
<b>Phase 2 Remarks</b>	<i>(none)</i>  The plan scope is one of the central concepts in the planning system. Its representation and interaction are very important accross the different workspace. Because the scope occurs in many workspaces, it needs to be consistent throughout them.



---

## Bibliography

---

- [ADO94] B. Beth Adelson, Susan Dumais, and Judith Olson, editors. *CHI'94 Human Factors in Computing Systems*, Boston, 1994. ACM, ACM Press.
- [AIS77] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language*. Oxford University Press, 1977.
- [Ale64] Christopher Alexander. *Notes on the synthesis of form*. Harvard university Press, 1964.
- [Ale79] Christopher Alexander. *The timeless way of building*. Oxford University Press, 1979.
- [ALMN99] David Avison, Francis Lau, Michael Myers, and Peter Axel Nielsen. Action research. *Communications of the ACM*, 42(1):94–97, January 1999.
- [Ano] Anonymous. Goldylocks and the three bears.
- [APS90] Chris Argyris, Robert Putnam, and Diana McLain Smith. *Action Science*. Jossey-Bass Publishers, San Francisco, 1990.
- [Arg80] Chris Argyris. *Inner contradictions of rigorous research*. Academic Press, 1980.

## BIBLIOGRAPHY

---

- [Ban95] Liam Bannon. The politics of design. *Communications of the ACM*, 38(9):66–68, September 1995.
- [Bas93] Len Bass. Architectures for interactive software systems: Rationale and design. In Len Bass and Prasun Dewan, editors, *User Interface Software*, number 1 in Trends in Software, chapter 2, pages 31–44. John Wiley & Sons Ltd., 1993.
- [BB91] Liam J. Bannon and Susanne Bødker. Beyond the interface: encountering artifacts in use. In John M. Carroll, editor, *Designing interaction: psychology of Human-Computer Interaction*, chapter 6, pages 227–253. Cambridge University Press, 1991.
- [BB92] Sunny Baker and Kim Baker. *On time/On budget: A step-by-step guide for managing any project*. Prentice Hall, 1992.
- [BB99] Jacob Buur and Kirsten Bagger. Replacing usability testing with user dialogue. *Communications of the ACM*, 42(5):63–66, May 1999.
- [BC85] Lorriane Borman and Bill Curtis, editors. *CHI'85 Human Factors in Computing Systems*, San Francisco, 1985. ACM, ACM Press.
- [BCNS92] Reinhard Budde, Marie-Luise Christ-Neumann, and Karl-Heinz Sylla. Tools and materials: an analysis and design metaphor. In G. Heeg, B. Magnusson, and B. Meyer, editors, *Tools-7, Technology of Object-Oriented Languages and Systems*, pages 135–146. Prentice Hall, 1992.
- [BD96] Graham Button and Paul Dourish. Technomethodology: Paradoxes and possibilities. In Michael J. Tauber, editor, *Proceedings of the CHI '96 conference on Human Factors in Computing Systems*, Human Factors in Computing Systems, pages 19–26. ACM SIGCHI, ACM Press, 1996.
- [BH98] Hugh Beyer and Karen Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. Interactive Technologies. Morgan Kaufmann, 1998.
- [BHvdMS93] P.A.J. Brijs, T.S. Hoek, C.A.P.G. van der Mast, and G.J.F. Smets. Designing in virtual reality: Modelling objects in a virtual environment (move). Technical Report DUT-TWI-93-91, Delft University of Technology, 1993.

- [Blo95] Jeanette L. Blomberg. Ethnography: Aligning field studies of work and system design. In Andrew F. Monk and Nigel Gilbert, editors, *Perspectives on HCI: Diverse Approaches*, pages 174–197. Academic Press, 1995.
- [Bot89] Pieter W.G. Bots. *An environment to support problem solving*. PhD thesis, Delft University of Technology, 1989.
- [Bry93] Erik Brynjolfsson. The productivity paradox of information technology. *Communications of the ACM*, 36(12), 1993.
- [BSMH95] Victoria Bellotti, Simon Buckingham Shum, Allan MacLean, and Nick Hammond. Multidisciplinary modelling in hci design... in theory and in practice. In *Proceedings of the CHI'95 conference on human factors in computing systems*, pages 146–153, Denver, CO, 1995. ACM SIGCHI, ACM Press.
- [But96] Keith A. Butler. Usability engineering turns 10. *interactions*, 3(1):58–75, 1996.
- [Bux94] William Buxton. The three mirrors of interaction: a holistic approach to user interfaces. In L.W. MacDonald and J. Vince, editors, *Interacting with virtual environments*. Wiley, New York, 1994.
- [cac96] Communications of the acm, 1996. The issue on technology transfer.
- [CAH87] Stuart K. Card and Jr. Austin Henderson. A multiple, virtual-workspace interface to support user task switching. In *Proceedings of CHI+GI 1987*, pages 53–59, Toronto, April 1987. ACM.
- [Car94] John M. Carroll. Making use a design representation. *Communications of the ACM*, 37(12):29–35, 1994.
- [Car95] John M. Carroll, editor. *Scenario-based Design*. John Wiley & Sons, Inc., 1995.
- [Car96] John M. Carroll. Artifacts and scenarios: an engineering approach. In Andrew F. Monk and Nigel Gilbert, editors, *Perspectives on HCI: Diverse Approaches*, pages 121–144. Academic Press, 1996.
- [Che81] P. Checkland. *Systems thinking, systems practice*. Wiley, 1981.

## BIBLIOGRAPHY

---

- [CK89] John M. Carroll and Wendy A. Kellogg. Artifact as theory-nexus: Hermeneutics meet theory-based design. In *Proceedings of ACM CHI'89 Conference on Human Factors in Computing Systems*, pages 7–14, 1989.
- [CMN83] S.K. Card, T.P. Moran, and A. Newell. *The psychology of human-computer interaction*. Erlbaum, Hillsdale, 1983.
- [CO88] John M. Carroll and Judith Reitman Olson. Mental models in human-computer interaction. In Martin Helander, editor, *Handbook of Human-Computer Interaction*, pages 45–65. North-Holland, New York, NY, 1988.
- [Col84] H.M. Collins. Researching spoonbending: Concepts and practice of participatory fieldwork. In C. Bell and H. Roberts, editors, *Social Researching: Politics, Problems, Practice*, pages 54–69. Routledge & Kegan Paul, London, 1984.
- [Cro89] Nigel Cross. *Engineering design methods*. John Wiley & Sons, Ltd., 1989.
- [CY79] Y.K. Cheung and M.F. Yeo. *A practical introduction to finite element analysis*. Pitman, 1979.
- [Dan81] George Daniels. *Watchmaking*. Sotheby Publications, 1981.
- [dB95] Leven de Braal. Perfuse: a medical expert system user interface prototype. Master's thesis, Delft University of Technology, 1995.
- [DFAB93] Alan Dix, Janet Finlay, Gregory Abowd, and Russel Beale. *Human-Computer Interaction*, chapter Task analysis, pages 221–250. Prentice Hall, 1993.
- [dG92] Johannes J. de Graaff. Context-sensitive help as an integral part of a user interface design environment. Master's thesis, Delft University of Technology, October 1992.
- [dG93] J.J. de Graaff. Tno-project: Uitwerking vragenlijst interactieve interface voor diana. Technical Report 93-NM-R0934, TNO, 1993. In Dutch.
- [dG94] J.J. de Graaff. Results from a user study for diana's interactive interface. Technical Report 94-NM-D047, TNO, 1994.

- [dGGP97] Johannes J. de Graaff, Abraham Guyt, and Martin Postma. Een presentatie en demonstratie van een nieuwe planningsstrategie voor scheepswerven. CD-Rom, 1997. Unpublished, In Dutch, Working title subject to change.
- [DH86] Jr. D.A. Henderson. The Trillium user interface design environment. In *Proceedings of SIGCHI '86, Human Factors in Computing Systems*, pages 221–227, Boston, 1986. ACM.
- [Dia89] Dan Diaper. *Task analysis for human-computer interaction*. Ellis Horwood, 1989.
- [Dia92] Diana world. DIANA Analysis BV, 1992.
- [Dia00] Diana analysis bv website, 2000. <http://www.diana.nl/>.
- [Dic93] B. Dick. A beginner's guide to action research. *Arcs Newsletter*, 1(1):5–9, 1993.
- [DIN87] Teil 8: Bildschirmarbeitsplätze. grundsätze der dialoggestaltung. DIN 66234 standard, 1987.
- [Dix90] Alan Dix. Non determinism as a paradigm for understanding the user interface. In M. Harrison and H. Thimbleby, editors, *Formal methods in human-computer interaction*, chapter 4, pages 97–127. Cambridge university press, 1990.
- [Dix95] Alan J. Dix. Formal methods. In Andrew F. Monk and Nigel Gilbert, editors, *Perspectives on HCI: Diverse Approaches*. Academic Press, 1995.
- [Dra93] Stephen W. Draper. The notion of task in hci. In *InterCHI'93 Adjunct proceedings*, 1993.
- [dV95] G.J. de Vreede. *Facilitating organizational change: the participative application of dynamic modelling*. PhD thesis, Delft University of Technology, 1995.
- [ea83] W. Buxton et al. Towards a comprehensive user interface management system. In *SIGGRAPH '83: Computer Graphics*, pages 34–42, 1983.

## BIBLIOGRAPHY

---

- [Eri] Thomas Erickson. Supporting interdisciplinary design: Towards a workplace pattern language. To appear in "Workplace Studies", pre-print from the Web at [http://www.research.apple.com/people/Tom/\\_Erickson/Patterns.html](http://www.research.apple.com/people/Tom/_Erickson/Patterns.html).
- [Fit54] P.M. Fitts. The information capacity of the human motor system in controlling amplitude of movement. *Journal of Experimental Psychology*, 47:381–391, 1954.
- [FKKM91] James D. Foley, W.C. Kim, S. Kovacevic, and K. Murray. Uide - an intelligent user interface design environment. In J. Sullivan and S. Tyler, editors, *Architectures for intelligent interfaces: Elements and Prototypes*, chapter 15, pages 339–384. Addison-Wesley, 1991.
- [FLMM90] Gerhard Fischer, Andreas C. Lemke, Thomas Mastaglio, and Anders I. Morch. Using critics to empower users. In *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, pages 337–347, 1990.
- [FNO95] Gerhard Fischer, Kumiyo Nakakoji, and Jonathan Ostwald. Supporting the evolution of design artifacts with representations of context and intent. In Gary M. Olson and Sue Schuon, editors, *Symposium on Designing Interactive Systems: Processes, Practices, Methods & Techniques*, pages 7–15. ACM Press, 1995. DIS '95 Conference Proceedings.
- [FPB95] Jr. Frederick P. Brooks. *The mythical man-month*. Addison-Wesley, 20th anniversary edition edition, 1995.
- [FRW<sup>+</sup>95] G. Fischer, D. Redmiles, L. Williams, G.I. Puhr, A. Aoki, and K. Nakakoji. Beyond object-oriented technology: where current approaches fall short. *Human-Computer Interaction*, 10(1):79–119, 1995.
- [FT89] George A. Ferguson and Yoshio Takane. *Statistical Analysis in Psychology and Education*. Psychology Series. McGraw-Hill, international edition edition, 1989.
- [FvDea82] James D. Foley and A. van Dam et al. *Fundamentals of Computer Graphics*. Addison-Wesley, 1982.

- [FW74] James D. Foley and Victor L. Wallace. The art of natural graphic man-machine conversation. *Proceedings of the IEEE*, 64(4), April 1974.
- [GCR98] Jr. George Chin and Mary Beth Rosson. Progressive design: Staged evolution of scenarios in the design of a collaborative science learning environment. In Steven Pemberton, editor, *Human Factors in Computing Systems, CHI 98 Conference Proceedings*, pages 611–618. ACM SIGCHI, ACM Press, 1998.
- [GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [Gib79] J.J. Gibson. *The ecological approach to visual perception*. Houghton-Mifflin, 1979.
- [GK91] Joan Greenbaum and Morton Kyng. *Design at work: cooperative design of computer systems*. Lawrence Erlbaum, 1991.
- [Gla95] Robert L. Glass. *Software creativity*. Prentice Hall, 1995.
- [Gor95] Peter Gorny. Expose: Hci-counseling for user interface design. In *Proc. Interact '95. IFIP*, 1995.
- [Gou88] John D. Gould. How to design usable systems. In Martin HeLANDER, editor, *Handbook of Human-Computer Interaction*, chapter 35, pages 757–789. North-Holland, New York, NY, 1988.
- [Gre86] Mark Green. A survey of three dialogue models. *ACM Transactions on Graphics*, 5(3), 1986.
- [Gre91] Siegfried Greif. The role of German work psychology in the design of artifacts. In John M. Carroll, editor, *Designing interaction: psychology of Human-Computer Interaction*, chapter 6, pages 203–226. Cambridge University Press, 1991.
- [Guy92] A. Guyt. Data input/data carrying methoden en middelen op scheepswerven. Masters thesis, Delft University of Technology, Delft, 1992.
- [Guy95a] Abraham Guyt. Besturingsontwerp. In Dutch, unpublished, 1995.

## BIBLIOGRAPHY

---

- [Guy95b] Abraham Guyt. A production management support tool: functional specification. Unpublished, 1995.
- [Guy95c] Abraham Guyt. Takenoverzicht besturingsontwerp. Project documentation, In Dutch, 1995.
- [Guy01] Abraham Guyt. *Shipyard planning*. PhD thesis, Delft University of Technology, 2001. Forthcoming.
- [GVQD93] P. Gorny, A. Viereck, L. Qin, and U. Daldrup. Slow and principled prototyping of usage surfaces: a method for user interface engineering. In *Proceedings of RE '93 — Prototyping*, pages 125–133, April 1993.
- [Ham90] Michael Hammer. Reengineering work: don't automate, obliterate. *Harvard business review*, pages 104–112, July-August 1990.
- [Har96] Morten Borup Harning. An approach to structured display design: Coping with conceptual complexity. In J. Vanderdonckt, editor, *Computer-Aided Design of User Interfaces*, pages 121–138, Namur, Belgium, June 1996. Presses Universitaires de Namur. Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI '96.
- [HB95] Lorin Hitt and Erik Brynjolfsson. Productivity without profit? three measures of information technology's value. *MIS Quarterly*, 1995.
- [HB96] Karen Holtzblatt and Hugh Beyer. *Field methods casebook for software design*, chapter Contextual design: principles and practice. John Wiley & Sons, New York, 1996.
- [HD81] D.R. Hofstadter and D.C. Dennett, editors. *The mind's I*. Basic Books, Inc., 1981.
- [Hec91] P. Heckel. *The elements of friendly software design*. SYBEX, 1991.
- [HH89] H. Rex Hartson and Deborah Hix. Toward empirically derived methodologies and tools for human-computer interface development. *Int. J. Man-Machine Studies*, pages 477–494, 1989.



- [HH93] Deborah Hix and H. Rex Hartson. *Developing user interfaces: ensuring usability through product and process*. Wiley professional computing, 1993. Textbook for a326 course.
- [HKM<sup>+</sup>95] Larry F. Hodges, Rob Kooper, Thomas C. Meyer, Barbara O. Rothbaum, Dan Opdyke, Johannes J. de Graaff, James S. Williford, and Max M. North. Virtual environments for treating the fear of heights. *IEEE Computer*, 28(7):27–34, July 1995.
- [HKRA95] John Hughes, Val King, Tom Rodden, and Hans Andersen. The role of ethnography in interactive systems design. *interactions*, 2(2):57–65, April 1995.
- [Hug90] J. Hughes. *The philosophy of social research*. Longman, London, second edition edition, 1990.
- [Jan83] Ann Janda, editor. *CHI'83 Human Factors in Computing Systems*, Boston, 1983. ACM, ACM Press.
- [Joh92] Peter Johnson. *Human computer interaction: psychology, task analysis and software engineering*. McGraw-Hill, 1992.
- [Joh00] Jeff Johnson. Textual bloopers. *interactions*, 7(5):28–48, September + October 2000.
- [Jon92] John Chris Jones. *Design methods*. Van nostrand reinhold, 1992.
- [KA92] B. Kirwan and L.K. Ainsworth, editors. *A guide to task analysis*. Taylor & Francis, Londen, 1992.
- [Kap92] Victor Kaptelinin. Human computer interaction in context: The activity theory perspective. In *East-West International Conference on Human-Computer Interaction: Proceedings of the EWHCI'92*, pages 7–13, 1992.
- [KB91] John Karat and John L. Bennett. Working within the design process. In John M. Carroll, editor, *Designing Interaction: Psychology at the Human-Computer Interface*, pages 269–285. Cambridge University Press, 1991.
- [KC90] J. Kirakowski and M. Corbett. *Effective methodology for the study of HCI*, volume 5 of *Human factors in information technology*. North-Holland, 1990.

## BIBLIOGRAPHY

---

- [Kim90] Scott Kim. Interdisciplinary cooperation. In Brenda Laurel, editor, *The art of human-computer interface design*, pages 31–44. Addison-Wesley, 1990.
- [Kir95] Alex Kirlik. Requirements for psychological models to support design: Toward ecological task analysis. In J.M. Flach, P. Hancock, J. Caird, and K.J. Vicente, editors, *Global Perspectives on the Ecology of Human-Machine Systems*, volume 1 of *Resources for Ecological Psychology*, chapter 5, pages 68–120. Lawrence Erlbaum Associates, Hillsdale, NJ, 1995.
- [KMM95] Irvin R. Katz, Robert Mack, and Linn Marks, editors. *CHI'95 Human Factors in Computing Systems*, Denver, 1995. ACM, ACM Press.
- [Kol] Raghu Kolli. An analysis of interaction design representations: Case studies of three interactive media projects. Unpublished.
- [Kyn95] Morton Kyng. Making representations work. *Communications of the ACM*, 38(9):46–55, September 1995.
- [Lan88] Thomas K. Landauer. Research methods in human-computer interaction. In Martin Helander, editor, *Handbook of Human-Computer Interaction*, pages 905–928. North-Holland, New York, NY, 1988.
- [Lan95] Thomas K. Landauer. *The trouble with computers*. The MIT Press, 1995.
- [Lau93] Brenda Laurel. *Computers as theatre*. Addison-Wesley, 1993.
- [Law90] Bryan Lawson. *How designers think: the design process demystified*. Butterworth Architecture, 1990.
- [LFNZ92] J.K. Liker, M. Fleischer, M. Nagamachi, and M.S. Zonneville. Designers and their machines: Cad use and support in the usa and japan. *Communications of the ACM*, 35(2):77–95, February 1992.
- [LVC89] M.A. Linton, J.M. Vlissides, and P.R. Calder. Composing user interfaces with interviews. *IEEE Computer*, 22(2):8–22, 1989.

- [MAR96a] MARC Analysis Research Corporation. *MARC/Designer Tutorial*, 1996. <http://www.marc.com/demo/>.
- [Mar96b] P. Marti. Task-centred design: turning task modelling into design. *SIGCHI Bulletin*, 28(3):65–70, July 1996.
- [MBYM91] Allan MacLean, Victoria Bellotti, Richard Young, and Thomas Moran. Reaching through analogy: a design rationale perspective on roles of analogy. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 167–172, 1991.
- [McG95] Joseph E. McGrath. Methodology matters: doing research in the behavioral and social sciences. In *Reading in human-computer interaction: towards the year 2000*, pages 152–169. Unknown, second edition, 1995.
- [MG95] Andrew F. Monk and Nigel Gilbert, editors. *Perspectives on HCI - Diverse approaches*. Computers and people. Academic Press, 1995.
- [MGD<sup>+</sup>90] Brad A. Myers, Dario Giuse, Roger B. Dannenberg, Brad Vander Zanden, David Kosbie, Ed Pervin, Andrew Mickish, and Philippe Marchal. Garnet: Comprehensive support for graphical, highly-interactive user interfaces. *IEEE Computer*, 23(11), November 1990.
- [Mit87] C.M. Mitchell. Gt-msocc: a research domain for modeling human-computer interaction and aiding decision making in supervisory control systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17:553–570, 1987.
- [Mor81] T.P. Moran. The command language grammar: a representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies*, pages 3–51, 1981.
- [MPH<sup>+</sup>97] Thomas P. Moran, Leysia Palen, Steve Harrison, Patrick Chiu, Don Kimber, Scott Minneman, William van Melle, and Polle Zellweger. “i’ll get that off the audio”: A case study of salvaging multimedia meeting records. In Steven Pemberton, editor, *Proceedings of the CHI 97 conference on Human Factors in Computing Systems*, Human Factors in Computing Systems, pages 202–209. ACM SIGCHI, ACM Press, 1997.

## BIBLIOGRAPHY

---

- [MPWJ92] P. Markopoulos, J. Pycock, S. Wilson, and P. Johnson. Adept: a task based design environment. In *Proceedings of the 25th Hawaii international conference on systems sciences*, pages 587–596. IEEE Computer Society, 1992.
- [MS95] Kevin Mullet and Darrell Sano. *Designing visual interfaces: communication oriented techniques*. Prentice-Hall, 1995.
- [MSN94] Roberto Moriyon, Pedro Szekely, and Robert Neches. Automatic generation of help from interface design models. In *Proceedings of SIGCHI '94, Human Factors in Computing Systems*, pages 225–231, Boston, 1994. ACM.
- [MYBM91] Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6(3,4):201–250, 1991.
- [Mye96] Brad A. Myers. A brief history of human-computer technology. Technical Report CMU-HCII-96-103, Carnegie Mellon University, 1996.
- [Nar96a] Bonnie A. Nardi, editor. *Context and Consiousness: Activity Theory and Human-Computer Interaction*. The MIT press, Cambridge, MA, 1996.
- [Nar96b] Bonnie A. Nardi. Studying context: a comparison of activity theory, situated action models, and distributed cognition. In *Context and Consiousness: Activity Theory and Human-Computer Interaction*, pages 69–102. The MIT Press, 1996.
- [ND86] Donald. A. Norman and Stephen W. Draper, editors. *User centered system design*. Lawrence erlbaum associates, 1986.
- [Neu95] Peter G. Neuman. *Computer-related risks*. Addison-Wesley, 1995.
- [NF] Jakob Nielsen and Jan Maurits Faber. Parallel user interface design. Submitted to IEEE Computing.
- [Nie93] Jakob Nielsen. *Usability Engineering*. Academic Press, 1993.
- [NL95] William M. Newman and Michael G. Lamming. *Interactive system design*. Addison-Wesley, 1995.

- [Nor88] Donald A. Norman. *The psychology of everyday things*. Basic books, inc., 1988.
- [Nor93] Donald A. Norman. *Things that make us smart*. Addison-Wesley, 1993.
- [Oka75] Hideyuki Oka. *How to wrap five more eggs*. Weatherhill, 1975.
- [Ols92] D.R. Olson. *User Interface Management Systems*. Morgan Kaufman, San Mateo, Ca., 1992.
- [Ope91] Open Software Foundation. *OSF/Motif Style Guide*, revision 1.1 edition, 1991.
- [PBACMF88] Mark D. Phillips, Howard S. Bashinski, Harry L. Ammerman, and Jr. Claude M. Fligg. A task analytic approach to dialogue design. In Martin Helander, editor, *Handbook of Human-Computer Interaction*, chapter 38, pages 835–857. North-Holland, New York, NY, 1988.
- [PC86] David Lorge Parnas and Paul C. Clements. A rational design process: how and why to fake it. *IEEE Transactions on software engineering*, SE-12(2), February 1986.
- [Pen90] Roger Penrose. *The emperor's new mind: concerning computers, minds, and the laws of physics*. Vintage, London, 1990.
- [Pet85] Henry Petroski. *To engineer is human*. St. Martin's Press, New York, 1985.
- [Pet94] Henry Petroski. *Design Paradigms: Case Histories of Error and Judgement in Engineering*. Cambridge University Press, 1994.
- [PG86] S.J. Payne and T.E. Green. Task-action grammars: a model of the mental representation of task languages. *Human-Computer Interaction*, pages 98–133, 1986.
- [Pri99] Primavera corporate website, 1999. <http://www.primavera.com/>.
- [PRS84] R.R. Panko and Jr. R.H. Sprague. Implementing office systems requires a new dp look. *Data Management*, 22(11), November 1984.

## BIBLIOGRAPHY

---

- [PRS<sup>+</sup>94] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. *Human-Computer Interaction*. Addison-Wesley, 1994. Written for the Open University.
- [Rae91] Arne Raeithel. Activity theory as a foundation for design. In C. Floyd, H. Züllighoven, R. Budde, and R. Keil-Slawik, editors, *Software development and reality construction*, chapter 8.4, pages 391–415. Springer-Verlag, 1991.
- [Rag97] Dave Raggett. *HTML 3.2 Reference Specification*. World Wide Web Consortium, 1997. <http://www.w3c.org/pub/WWW/TR/REC-html32.html>.
- [RE91] N.F.M. Roozenburg and J. Eekels. *Produktontwerpen, structuur en methoden*. Lemma B.V., 1991. (In Dutch).
- [Rei89] R. Reichman. *Integrated interfaces based on a theory of context and goal tracking*, chapter 15, pages 209–227. Number 4 in Human factors in information technology. North-Holland, 1989.
- [Rhe95a] John Rheinfank. A conversation with don norman. *interactions*, 2(2):47–55, April 1995.
- [Rhe95b] John Rheinfank. A conversation with john seely brown. *interactions*, 2(1):47–55, January 1995.
- [Ris] Risks forum. Online. todo: more info needed.
- [RMK87] Mary Beth Rosson, Susanne Maass, and Wendy A. Kellog. Designing for designers: an analysis of design practice in the real world. In John M. Carroll and Peter P. Tanner, editors, *CHI + GI 1987 conference proceedings*, pages 137–142. SIGCHI, ACM, 1987.
- [RN96] Byron Reeves and Clifford Nass. *The Media Equation: How People Treat Computers, Television, and Media Like Real People and Places*. Cambridge University Press, Cambridge, 1996.
- [RZ95] Dick Riehle and Heinz Züllighoven. A pattern language for tool construction and integration based on the tools&materials metaphor. In James O. Coplien and Douglas C. Schmidt, editors, *PLoP-94, Pattern Languages of Program Design*, chapter 2. Addison-Wesley, 1995.

- [Sac95] Patricia Sachs. Transforming work: collaboration, learning and design. *Communications of the ACM*, 38(9):36–44, September 1995.
- [Sea96] Andrew Sears. Hci education: some progress and some new questions. *SIGCHI Bulletin*, 28(4), 1996.
- [SF90] P. Sukaviriya and James D. Foley. Coupling a ui framework with automatic generation of context-sensitive animated help. In *Proceedings of the ACM SIGGRAPH symposium on User Interface Software and Technology*. ACM SIGGRAPH, oct 1990.
- [SG90] Franz Schiele and Thomas Green. Hci formalisms and cognitive psychology: the case of task-action grammar. In M. Harrison and H. Thimbleby, editors, *Formal methods in human-computer interaction*, chapter 2, pages 9–62. Cambridge university press, 1990.
- [SLN93] Pedro Szekely, Ping Luo, and Robert Neches. Beyond interface builders: Model-based interface tools. In *Proceedings of InterCHI '93, Human Factors in Computing Systems*, pages 383–390, Amsterdam, 1993. ACM.
- [Sol82] H.G. Sol. *Simulation in information systems development*. PhD thesis, University of Groningen, 1982.
- [Sol87a] H.G. Sol. *Decision Support Systems: Theory and Applications*, chapter Paradoxes around DSS, pages 3–18. Springer-Verlag, Berlin, 1987.
- [Sol87b] H.G. Sol. Informatiesystemen. Delft University of Technology course material, 1987. In Dutch.
- [SPS97] SPSS. *SPSS for Windows Manual*, 8.0.0 edition, 1997.
- [SRAD97] Michael Scaife, Yvonne Rogers, Frances Aldrich, and Matt Davies. Designing for or designing with? informant design for interactive learning environments. In Steven Pemberton, editor, *Proceedings of the CHI 97 conference on Human Factors in Computing Systems*, Human Factors in Computing Systems, pages 343–350. ACM SIGCHI, ACM Press, 1997.

## BIBLIOGRAPHY

---

- [SS92] Paul Seaton and Tom Stewart. Evolving task oriented systems. In *CHI'92 Conference Proceedings*, pages 463–469. ACM SIGCHI, ACM, 1992.
- [Sta99] Inc. Statsoft. *Electronic Statistics Textbook*. Statsoft, Tulsa, OK, USA, 1999. <http://www.statsoft.com/textbook/stathome.html>.
- [Suc87] Lucy A. Suchman. *Plans and situated actions*. Cambridge University Press, 1987.
- [Sun89] Sun microsystems, Inc., Mountain View. *OPENLOOK graphical user interface functional specification*, 1989.
- [Tay93] James R. Taylor. *Rethinking the theory of organizational communication*. Ablex, Norwood, New Jersey, 1993.
- [TCKO00] Stephanie Teasley, Lisa Covi, M.S. Krishnan, and Judith Olson. How does radical collocation help a team succeed? In *CSCW 2000 Conference Proceedings*. ACM Press, 2000.
- [tH88] L.A. ten Horn. Wenken voor vrije interviewers. Collegedictaat, Faculteit der Wijsbegeerte en Technische Maatschappijwetenschappen, Delft University of Technology, 1988. In Dutch.
- [tH93] Arthur H.M. ter Hofstede. *Information modelling in data intensive domains*. PhD thesis, Katholieke Universiteit Nijmegen, 1993.
- [Tha01] John Thackara. The design challenge of pervasive computing. *interactions*, 8(3):46–51, May + June 2001.
- [Thi90] Harold Thimbleby. *User Interface Design*. Frontier series. ACM Press, 1990.
- [Tic85] Walter F. Tichy. Rcs — a system for version control. *Software — Practice & Experience*, 15(7):637–654, July 1985.
- [Tim99] Time line solutions corporate website, 1999. <http://www.tlsolutions.com>.
- [TM91] Linda Tetzlaff and Robert L. Mack. Discussion: perspectives on methodology in HCI research and practice. In John M. Carroll,



- editor, *Designing interaction: psychology of Human-Computer Interaction*, chapter 6, pages 286–314. Cambridge University Press, 1991.
- [TNO91] TNO Building and Construction Research. *Introduction and Linear Static Analysis: User Environment and Manuals*, 1991. DIANA Course Notes.
- [TNO94] Tno corporate website, 1994. <http://www.tno.nl/>.
- [TNO00] Tno corporate website, 2000. <http://www.tno.nl/>.
- [Tog92] Bruce Tognazzini. *Tog on Interface*. Addison-Wesley, 1992.
- [vACM95] J.W. van Aalst, T.T. Carey, and D.L. McKerlie. Design space analysis as ‘training wheels’ in a framework for learning user interface design. In *CHI’95 Conference Proceedings*, pages 154–161. ACM SIGCHI, ACM Press, 1995.
- [vdM95a] C.A.P.G. van der Mast. *Developing educational software: integrating disciplines and media*. PhD thesis, Delft University of Technology, 1995.
- [vdM95b] Charles van der Mast. Professional development of multimedia courseware. In *Machine-Mediated Learning*, pages 269–292. Lawrence Erlbaum Associates, Inc., 1995.
- [vdMV92] Charles van der Mast and Johan Versendaal. Separation of user interface and application with the delft direct manipulation manager (d2m2). In *Proceedings of IFAC Man-Machine Symposium*, 1992.
- [vM94] Jeroen W. van Meel. *The dynamics of business engineering*. PhD thesis, Delft University of Technology, 1994.
- [VSG91] Axel Viereck, Egbert Schlunbaum, and Peter Gorny. Structured design of user-interfaces and knowledge-based design. In H.-J. Bullinger, editor, *Human aspects in computing*, Advances in Human Factors/Ergonomics. Elsevier, 1991.
- [WCS96] Larry Wall, Tom Christiansen, and Randal I. Schwartz. *Programming Perl*. O’Reilly & Associates, 2nd edition, 1996.

## BIBLIOGRAPHY

---

- [Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, May 1997.
- [WHK90] Dennis Wixon, Karen Holtzblatt, and Stephen Knox. Contextual design: an emerging view of system design. In *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, Transcending perspectives, pages 329–336, 1990.
- [Wij91] Gerard M. Wijers. *Modelling support in information systems development*. PhD thesis, Delft University of Technology, 1991.
- [Win96] Terry Winograd, editor. *Bringing design to software*. Addison-Wesley, 1996.
- [WJ95] Stephanie Wilson and Peter Johnson. Empowering users in a task-based approach to design. In Gary M. Olson and Sue Schuon, editors, *Symposium on Designing Interactive Systems: Processes, Practices, Methods & Techniques*, pages 25–31. ACM Press, 1995. DIS '95 Conference Proceedings.
- [WJK<sup>+</sup>93] S. Wilson, P. Johnson, C. Kelly, J. Cunningham, and P. Markopoulos. Beyond hacking: a model based approach to user interface design. In *People and computers VIII, proceedings of the HCI'93 conference*. Cambridge university press, 1993.
- [Won93] Yin Yin Wong. Layer tool: Support for progressive design. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems – Adjunct Proceedings*, pages 127–128, 1993.
- [WZ96] Larry E. Wood and Ron Zeno. Transforming user-centered analysis into concrete design. *SIGCHI Bulletin*, 28(4):35–38, October 1996.
- [Yin89] R.K. Yin. *Case study research: design and methods*, volume 2 of *Applied social research methods*. SAGE publications, revised edition edition, 1989.
- [You84] L.F. Young. A corporate strategy for decision support systems. *Journal of Information Systems Management*, 1(1), 1984.

---

## Summary

---

Starting the desing of a complex interactive system is almost magical, especially when the work these systems need to describe can not be described easily. WONDER, a design tool described in this thesis, aims to support this early part of the design process. Work that can not be described easily is also called ill-structured work; the lack of structure is precisely what makes it hard to describe. Examples of this type of work are planning of activities on a shipyard and making complex computations to optimize structural designs.

To focus the research three research questions are formulated. The first question asks which design activities are needed during the early part of the design process for interactive systems which need to support ill-structured work. What needs to be done? What are the right steps? Which activities are important, and which are not?

The second question looks for a way to describe the results of the design activities without using actual user interface components. The idea behind this question is that at this stage in the design process it is too early to make the user interface design explicit. Instead, the structure and the overall integrity of the design should focused on. User interface elements such as windows and menus can be defined at a later stage. This question asks what kind of representation is suitable for supporting the activities from the first question.

The third question is posed to ensure that the activities from the first question and the representation from the second question form a whole that is actually usable and workable. A design tool only makes sense when it can be used

during the design process.

These questions are built on a foundation: the belief that interactive systems should be designed as a whole, and that the leading perspective during design should be that of its users. This paradigm influences the research questions.

The research has been carried out according to an inductive-hypothetic strategy. In this strategy four main activities are carried out. First, a description of a real world situation is made. Often this is done based on some kind of initial theory. Second, lessons learned from describing this real world situation are abstracted and generalized. In particular, the perceived problems are made clear. This leads to step three, where a theory is developed based on the generalized observations. Finally, in step four, this theory is put into practice, and the results are evaluated. This evaluation can again be input for the first step of the process.

Thus, as a first step, a case study has been carried out. Basic theory about the design of interactive systems was used. This basic theory was taken from a small number of textbooks widely used to teach interaction design. The case study itself was the design of an interactive system for finite element analysis, called DIANA. DIANA as a system already existed, but this version only had a command line based interface and was not interactive.

The main conclusion from this case study was that the HCI textbooks did not offer much concrete support for the early part of the design process. Useful activities were interviews with users and task analysis. The interviews with users provided a clear image of the work context. The task analysis provided structure, but dealing with the ill-structured parts of the work proved to be hard. Two types of representations were used. Textual representations turned out to be hard to communicate because so much was open for interpretation. Graphical representations caused the discussion to shift to the specifics of the graphics instead of more high-level issues.

Armed with the experience from the DIANA case in general and the conclusions drawn from it in particular, a literature study has been carried out as the second activity of the research strategy. This study focused on support for analysis and early design activities, and on the nature of multi-disciplinary teams. A number of issues were not resolved by this study. No suitable design tools were found which would answer the research questions. In particular, they are often too rigid or too flexible; finding a useful balance seems to be hard. Context was deemed very important by a number of sources, but specifics were hard to come by. Not much information could be uncovered about the support for ill-structured work. In addition to these open issues a number of recommendations and suggestions have been formulated.

The third activity in the research strategy is to formulate a theory. This resulted in a new design tool called WONDER. WONDER stands for *workspace oriented design representation*. WONDER consists of a number of representations which change over time and a number of activities which describe how to work with the representations. As the design progresses, the representations become more and more structured. The key representation of WONDER is that of a *workspace*. A workspace is part of the interactive system where a particular goal can be accomplished, e.g. optimizing the floor plan of a shipyard. Workspaces can contain other workspaces or refer to them, thus building a loose hierarchy of goals. This hierarchy needs to confirm to the hierarchy of goals uncovered during the analysis carried out earlier, essentially mapping the work to be supported to these workspaces. Each workspace also refers to *materials*. These materials exist outside of the workspaces, thus forming the connection between workspaces. In the context of interactive systems these materials are really the data to be worked on, e.g. the activities in the plan of the shipyard planner.

Support for ill-structured work is accomplished by not trying to describe it in details. Instead a workspace collects all elements needed to carry out the ill-structured work. This includes all needed materials and tools to make changes or inspect these materials. The user can then use these tools and materials to solve the problems presented by the work.

The fourth activity in the research strategy is to put the theory of WONDER into practice and evaluate it. The SHIPSHAPE case, designing a planning system for a shipyard, provided a challenging environment to test WONDER in. Before starting the case a number of assumptions were formulated, each addressing a particular aspect of WONDER, e.g. that text is a good medium for a model during the early design phase.

The SHIPSHAPE case was tested in two ways. Action research was used to get an in depth feel for using WONDER, and a prototype implementation based on the WONDER design was evaluated with shipyard planners. Results from both of these activities were used to verify the assumptions.

Upon evaluation almost all the assumptions turned out to be true. This indicates that WONDER does indeed live up to expectations. Only one assumption was false: some design information was kept outside of WONDER representations, even though it was assumed that WONDER could accommodate all design artifacts at this early stage of design. Two assumptions were undecided. One questioned whether the representations contained the right ingredients. While the *workspace* and *material* representations worked well, the initially planned *action* was hardly used during design. In hindsight this representation required too much detail to be useful at this stage in the design process.

## *BIBLIOGRAPHY*

---

Another assumption on the roles in the design team was undecided because each design team member served several roles, making it hard to separate each one out.

Does WONDER also provide an answer for the research questions? The fact that it exists provides an answer to the first two questions. The activities which are part of WONDER describe the initial design activities. The representations, and in particular the combination of workspaces and materials provides a way to describe ill-structured work explicitly without using user interface components. WONDER also provides a usable and workable solution, as is asserted by some of the assumptions and by testimony of the design team.

There is something more important than WONDER or the fact that it answers the research questions positively: the main principle emerging from the theory. This main principle is finding the ever changing balance between structure and freedom. In WONDER itself this is witnessed in the changing balance from freedom towards structure in the use of the representations. In the results of WONDER this is shown by trying to find the balance in offering just the right materials and tools in a workspace. This principle can be applied more broadly than WONDER itself.

---

## Samenvatting

---

Het begin van het ontwerpen van een complex interactief systeem is bijna magie, vooral wanneer het werk dat dit systeem moet beschrijven niet goed te beschrijven is. WONDER, een ontwerpgereedschap dat beschreven is in dit proefschrift, heeft als doel het ondersteunen van het begin van het ontwerpproces. Werk dat niet goed beschreven kan worden wordt ook wel slecht gestructureerd genoemd; het gebrek aan structuur is precies wat het moeilijk maakt het te beschrijven. Voorbeelden van dit type werk zijn het plannen van activiteiten op een scheepswerf en het maken van complexe berekeningen om ontwerpen van fysieke structuren te optimaliseren.

Om het onderzoek focus te geven zijn drie onderzoeksvragen opgesteld. De eerste vraagt welke ontwerpactiviteiten nodig zijn tijdens het eerste deel van het ontwerpproces voor interactieve systemen die slecht gestructureerd werk moeten ondersteunen. Wat moet er gedaan worden? Wat zijn de juiste stappen? Welke activiteiten zijn belangrijk, en welke niet?

De tweede vraag zoekt een manier om de resultaten van de ontwerpactiviteiten te beschrijven zonder concrete elementen van de gebruikersinterface te gebruiken. Het idee achter deze vraag is dat het in deze fase van het ontwerpen te vroeg is om de gebruikersinterface al expliciet te maken. In plaats daarvan is focus nodig op de structuur en de integriteit van het ontwerp. Elementen van de gebruikersinterface zoals windows en menus kunnen later gedefinieerd worden. Deze vraag zoekt een representatie die de activiteiten van de eerste vraag goed kan ondersteunen.

De derde vraag wordt gesteld om te zorgen dat de activiteiten van de eerste vraag en de representatie van de tweede vraag een geheel vormen dat bruikbaar en werkbaar is. Een ontwerpgereedschap heeft alleen zit als het ook gebruikt kan worden tijdens het ontwerpproces.

Deze vragen bouwen voort op een fundament: het geloof dat interactieve systemen moeten worden geïmplementeerd als een geheel, en dat het leidende perspectief tijdens het ontwerpen dat van de gebruikers moet zijn. Dit paradigma beïnvloedt de onderzoeksvragen.

Het onderzoek is uitgevoerd volgens een inductief-hypothetische strategie. In deze strategie worden vier hoofdactiviteiten uitgevoerd. Als eerste wordt een beschrijving gemaakt van een bestaande situatie. Vaak is deze beschrijving gebaseerd op een eerste aanzet tot een theorie. Als tweede stap worden de lessen die geleerd zijn tijdens het bestuderen van de bestaande situatie meer algemeen bekeken. Met name de waargenomen problemen worden duidelijk gemaakt. Dit leidt tot stap drie waarbij een theorie ontwikkeld wordt op basis van de algemeen gemaakte observaties. Ten slotte wordt in stap vier de theorie in de praktijk beproefd en geëvalueerd. Deze evaluatie kan weer dienen als bron voor de eerste stap van het proces.

Zodoende is als eerste stap een case bekeken. Standaard theorie over het ontwerpen van interactieve systemen is hiervoor gebruikt. Deze standaard theorie is gebaseerd op een klein aantal boeken die veel gebruikt worden om het ontwerpen van interactieve systemen te leren. De case zelf betrof het ontwerpen van een interactief systeem, DIANA, voor het maken van eindige elementen methode berekeningen. DIANA bestond al, maar alleen met een niet-interactieve command-gestuurde interface.

De belangrijkste conclusie die uit deze case getrokken kan worden is dat de HCI boeken niet veel concrete ondersteuning bieden voor het begin van het ontwerpproces. Waardevolle activiteiten waren het interviewen van gebruikers en taak analyse. De interviews met gebruikers gaven een duidelijk beeld van het werk en de omgeving waarin dat gebeurt. Taak analyse zorgde voor structuur, maar werkte niet goed bij de slecht gestructureerde onderdelen van het werk. Twee typen representaties zijn gebruikt. Representaties gebaseerd op tekst bleken moeilijk communiceerbaar omdat zo veel nog open was voor interpretatie. Bij grafische representaties dwaalde de discussie snel naar de specifiek grafische elementen in plaats van de issues op hoger niveau.

Gewapend met de ervaring van de DIANA case in het algemeen, en de conclusies daarvan in het bijzonder, is als tweede activiteit van de onderzoeksstrategie een literatuurstudie uitgevoerd. Deze studie richtte zich op de ondersteuning voor analyse en ontwerp-activiteiten, en op multi-disciplinaire teams. Een



aantal zaken werden door de literatuurstudie niet opgehelderd. Zo werd er geen bruikbaar ontwerpgerederschap gevonden dat aan de onderzoeksvragen beantwoord. Ze waren in het algemeen te rigide of te flexibel; de juist balans is niet eenvoudig gevonden. De werkomgeving werd door een aantal bronnen als belangrijk bestempeld, maar dit werd niet meer specifiek gemaakt. Ook kon er weinig informatie gevonden worden over slecht gestructureerd werk. Naast deze zaken werden een aantal aanbevelingen en suggesties gedaan.

De derde activiteit van de onderzoeksstrategie is het formuleren van een theorie. Dit heeft geresulteerd in een nieuw ontwerpgerederschap genaamd WONDER. WONDER is een afkorting van *workspace oriented design representation*. WONDER bevat een aantal representaties die gedurende de tijd veranderen en een aantal activiteiten die beschrijven hoe met de representaties gewerkt moet worden. Naarmate het ontwerp zich ontwikkelt worden de representaties steeds meer gestructureerd. De centrale representatie van WONDER is die van de *workspace*. Dit is een onderdeel van het interactieve systeem waarin een bepaald doel bereikt kan worden, bijvoorbeeld het optimaliseren van het vloerplan van een scheepswerf. Workspaces kunnen andere workspaces omvatten of er naar verwijzen. Op deze manier wordt een losse hiërarchie van doelen opgebouwd. Deze hiërarchie moet aansluiten bij de doelen zoals die tijdens de analyse fase gevonden zijn. Zodoende vormt de hiërarchie de blauwdruk van het werk wat ondersteund moet worden. Elke workspace refereert ook naar *materials*. Deze bestaan los van de workspaces en vormen op die manier de verbinding tussen de workspaces. In de context van een interactief systeem zijn deze materials de data waarmee gewerkt worden, bijvoorbeeld de activiteiten in het plan van de planner op een scheepswerf.

Ondersteuning voor slecht gestructureerd werk wordt bereikt door het niet in detail te willen beschrijven. In plaats daarvan bevat een workspace alle elementen die nodig zijn om het slecht gestructureerde werk te beschrijven: alle materialen en gereedschappen waarmee deze materialen bekeken of aangepast kunnen worden. De gebruiker kan deze materialen en gereedschappen dan zelf inzetten om de problemen van het werk op te lossen.

De vierde activiteit in de onderzoeksstrategie is het in de praktijk brengen van de theorie van WONDER. De SHIPSHAPE case, het ontwerpen van een planning systeem voor een scheepswerf, is een uitdagende omgeving om WONDER in te testen. Voor de start van de case zijn een aantal veronderstellingen opgesteld die elk een specifiek aspect van WONDER toetsbaar maken, bijvoorbeeld het feit dat tekst een goed medium is om te gebruiken voor representaties tijdens de eerste ontwerpactiviteiten.

De SHIPSHAPE case is op twee manieren getest. Met behulp van action

research is duidelijk geworden hoe het is om WONDER in de praktijk te gebruiken, en met behulp van een prototype op basis van het WONDER ontwerp is geevalueerd met planners van scheepswerven. De resultaten van beide activiteiten zijn gebruikt om de veronderstellingen te verifiëren.

Na de evaluatie blijken vrijwel alle veronderstellingen waar. Dit geeft aan de WONDER inderdaad aan de verwachtingen voldoet. Slechts één veronderstelling was onwaar: niet alle ontwerp informatie was vastgelegd met WONDER representaties, ook al was het de veronderstelling dat het gehele ontwerp binnen WONDER zou passen in deze fase van het ontwerpproces. Twee veronderstellingen konden niet worden getoetst. Een ervan ging erover of de representaties de juiste ingrediënten bevatten. De *workspace* en *material* representaties werken goed, maar de in eerste instantie geplande *action* is bijna niet gebruikt. Achteraf bezien was deze representatie te gedetailleerd om bruikbaar te zijn tijdens deze fase van het ontwerpproces. Een andere veronderstelling had betrekking op het ontwerpteam. Deze kon niet worden getoetst omdat elk teamlid meerdere rollen had, die moeilijk los te zien waren van elkaar.

Beantwoord WONDER nu ook de onderzoeksvragen? Het feit dat WONDER bestaat geeft al een antwoord op de eerste twee vragen. De activiteiten die onderdeel zijn van WONDER beschrijven de eerste ontwerpactiviteiten. De representaties en in het bijzonder de combinatie van workspaces en materials levert een manier om slecht gestructureerd werk zo goed mogelijk te beschrijven zonder dat teruggevallen moet worden op user interface elementen. WONDER is ook een bruikbare en werkbare oplossing, zoals aangetoond wordt door enkele van de veronderstellingen en door de ervaringen van het ontwerpteam.

Eén ding is belangrijker dan WONDER of het feit dat WONDER een positief antwoord levert op de onderzoeksvragen: het principe dat voortkomt uit de theorie. Dit principe is het zoeken naar de altijd veranderende balans tussen structuur en vrijheid. In WONDER zelf is dit te vinden als de balans die verandert van vrijheid naar structuur bij het gebruik van de representaties. In de resultaten van WONDER is het terug te vinden als de balans in het juiste aanbod van materials en tools in een workspace. Dit principe kan breder toegepast worden dan alleen in WONDER.

---

## Curriculum Vitae

---

Hans de Graaff was born on July 18<sup>th</sup>, 1968 in 's-Gravenzande, The Netherlands. In 1986 he graduated from high school, the Zandeveld College in 's-Gravenzande, concluding his basic education at pre-university level. The same year he began to study computer science at Delft University of Technology in Delft, The Netherlands. As part of this study he visited the Georgia Institute of Technology in 1991. There, he worked on his thesis in the Graphics, Visualization, and Usability Center for nine months. After returning to The Netherlands, he finished his thesis on "Context-sensitive help as an integral part of a use interface design environment". On October 1<sup>st</sup>, 1992 he started working at Delft University of Technology as a Ph.D. researcher. During the course of his research he has assisted a number of final year students with their master's projects. While at the university he has helped to set up and teach courses on the design of interactive systems. Since 1997 he works at KPN Research, where he conducts research into information and communication technology. Currently he investigates the use and need of high-bandwidth mobile technology such as i-mode and UMTS from the perspective of the end user.