



SIKS Dissertation Series No. 2017-xx

ISBN/EAN: xxx-xx-xxxxx-xx-x

© Brigit van Loggem, Overberg, The Netherlands, 2017

Cover design by xxx

Printed by xxx, The Netherlands

All rights reserved

Towards a Design Rationale for Software Documentation

A Model of Computer-Mediated Activity

Proefschrift

ter verkrijging van de graad van doctor
aan de Open Universiteit
op gezag van de rector magnificus
prof. mr. A. Oskamp
ten overstaan van een door het
College voor de promoties ingestelde commissie
in het openbaar te verdedigen

op _____ 2017 te Heerlen
om xx.00 uur precies

door

Brigitte Elisabeth van Loggem

Geboren op 11 augustus 1957 te Amsterdam

Promotor

Prof. dr. G.C. van der Veer
Open University of the Netherlands

Overige leden van de beoordelingscommissie

Prof. dr. M.C.J.D. van Eekelen
Open University of the Netherlands

Dr. J. Karreman
Universiteit Twente

Prof. dr. P.A. Kirschner
Open University of the Netherlands

Prof. dr. D.R. Olsen Jr.
Brigham Young University, UT, USA

For Matthew

Table of Contents

1. RESEARCH QUESTIONS AND SCOPE	1
INTRODUCTION	1
CHAPTER CONTENTS	4
2. THE CASE FOR USER DOCUMENTATION	9
THE PARADOX OF THE ACTIVE USER	9
USER DOCUMENTATION AS THE CINDERELLA OF USER SUPPORT	12
DOCUMENTATION AS A DESIGN DISCIPLINE	17
THE NEED FOR RESEARCH	25
3. THE DOCUMENTATION JOURNEY	33
INTERACTION WITH DOCUMENTATION: THE SECONDARY TASK	33
STAGEPOST 1: NEEDING INFORMATION	36
STAGEPOST 2: SEEKING INFORMATION	37
STAGEPOST 3: FILTERING INFORMATION	40
STAGEPOST 4: APPLYING INFORMATION	41
4. A MODEL OF COMPUTER-MEDIATED ACTIVITY	43
INTERACTION WITH SOFTWARE: THE PRIMARY TASK	43
CMA AS REPEATED DECISION-MAKING	44
LEARNING FROM PRACTICE	47
SATISFICING AND AMBITION IN CMA	50
COGNITIVE LOAD IN CMA	53
MENTAL MODELS AND THE USER VIRTUAL MACHINE	54
UNCERTAINTY AND ILL-DEFINEDNESS	58
THE DOCUMENTATION JOURNEY IN CMA	62
CMA AS A KNOWLEDGE ENGINE	64
5. DOCUMENTATION AS ARTEFACT	71
EXPERTISE AND MASTERY	71
ASSESSING USE COMPLEXITY	73
DOCUMENTATION REQUIREMENTS FOR OPERATORS, ACTORS AND ACTIVATORS	83
6. SDDPL VALUE SYSTEM AND ORGANIZING PRINCIPLE	89
DESIGN PATTERNS AND DESIGN PATTERN LANGUAGES	89
COMMON VALUE SYSTEM	92
COMMON ORGANIZING PRINCIPLE	94
A PROPOSED FORMAT	100

7. MACRO PATTERNS	103
1. DOCUMENTATION ENVIRONMENT	103
2. MEDIA MIX	109
8. MESO PATTERNS	117
3. SEPARATION OF PURPOSE	117
4. MOTIVATOR	126
5. EVERY PAGE IS PAGE ONE (EPPO)	130
6. STEP LADDER TUTORIAL	134
7. MINIMAL MANUAL	137
8. COOKBOOK	140
9. MICRO PATTERNS	143
9. JOB AIDS	143
10. SCREEN CAPTURE	147
11. CONCEPTUAL MODEL	154
12. CROSS-REFERENCE	160
13. CHAPTER SUMMARIES & CHAPTER INTRODUCTIONS	163
14. STEPWISE INSTRUCTIONS	166
10. DISCUSSION AND CONCLUSIONS	169
COMPONENT RESEARCH QUESTIONS: CONCLUSIONS	169
OVERALL RESEARCH QUESTION: DISCUSSION	174
A WORD TO PRACTITIONERS	176
11. APPENDIXES	179
APPENDIX 1. SOFTWARE DOCUMENTATION: A STANDARD FOR THE 21 ST CENTURY	179
APPENDIX 2. “NOBODY READS THE DOCUMENTATION”—TRUE OR NOT?	189
APPENDIX 3. USING THE REPERTORY GRID TECHNIQUE FOR MINING DESIGN PATTERNS	203
APPENDIX 4. TERMS LIST	211
12. REFERENCES	215
13. SAMENVATTING	237
14. CURRICULUM VITAE	239

Dedication

In 1992 or perhaps 1993, an employee of a computer shop arrived at the office to install a new PC. In those days, getting a PC to actually do any work was a non-trivial exercise, and one not to be left to the uninitiated. Computer suppliers could distinguish themselves from the competition by not simply shipping the box, but hand-delivering the new equipment and installing operating system and applications software at the customer's premises. All this tended to take time. As the man sat feeding floppy disks into the new computer, occasionally reaching out to press a key, in an attempt to kill time and make conversation he asked what line of business we were in. Software manuals, we told him. But surely, our new friend said, manuals did not need to be written? After all, didn't they always come with the software? The idea that someone had to actually write those manuals proved difficult to grasp. Did we perhaps translate them? No we didn't, we wrote them. Ah. So we re-wrote them, those that were difficult to understand? No, we wrote them from scratch. But why, if they already came with the software? The conversation became a little bit strained. When DOS and WordPerfect were ready to do their work on the new office computer and our technician left, he seemed still unconvinced that software manuals were not somehow brought down from Mount Sinai by Moses, on the same stone tablets that contained the Ten Commandments; but that they are written by people, from scratch.

The man who installed a computer for me in the early 1990s, and whose name I never knew, is not part of the documentation design community at which this work is aimed. Yet in an indirect way, it was written for him.

1. *Research Questions and Scope*

Abstract: User documentation for software-driven tools lacks a comprehensive framework within which its products can be positioned for discussion, design, research and evaluation. Practising documentation designers and academic researchers into user documentation need to know what the process is that the documentation artefacts aim to support, as well as how and under which conditions the proposed support products can be expected to intervene in this process. This PhD thesis presents such a comprehensive framework in the shape of a model of Computer-Mediated Activity (CMA). It is then explored how the CMA model can provide design rationale for documentation artefacts, by outlining a Software Documentation Design Pattern Language (SDDPL).

Introduction

The use of software by others than specialists began roughly with the advent of the personal computer in the 1970s and has been problematic from the very beginning. Software is for most users a means to an end, not an end in itself: people start out with the intention to get something done. They turn to a piece of software to help them achieve their objective; but if the tool does not immediately seem to meet requirements, then rather than find out how to apply it to the problem at hand, they lower their aims to match what they think the software can do. Sometimes the bar is lowered considerably, to such a degree even that I have seen quite a few instances of (costly) software hardly being used at all.

The answers that people give when asked why they are satisfied with unsatisfactory results of their often considerable efforts, seem to be a variation on either of two themes: “I have a job to do now, but one day I will find the time to look into this” is one; and “I don’t need all those advanced features, I only use the software to do simple things” is the other. To me, this has always sounded like the new car owner who, not knowing about the concept of gears, regards anything higher than first gear as an advanced feature that it is not necessary to explore, because the car is used only for simple tasks such as doing the weekly shopping. Personally, I have never met anyone who told me, *First gear gets me there, doesn’t it? When I have to travel out of town I take the train, so I’ve no problem with driving along at ten miles per hour. All those old biddies behind me can honk their horns as much as they want! And when I get to the parking space, I just drive around the block if I have to, so why should I take the trouble to learn about difficult things such as reverse gear?* Yet I must have met dozens, if not hundreds of people who told me something along the lines of, *I’m sure that styles and templates are very useful if you do clever things with a word processor; but I don’t need them for my ten-page articles. It’s good fun playing with all those fonts on my computer, and if my table of*

contents comes out funny then so be it — I'm an author, not a computer scientist! Sure, my editor grumbles about the files I deliver; and re-using bits of an article I wrote last year in something I'm writing today can be a real pain. But that's life, isn't it? Whereas driving instruction is in most countries enforced by law, not all company employees are trained before or even during their work with the sometimes very powerful computer software installed on their desktops. Without the documentation, how can they learn?

When I started out on the present work, I had been working as a technical writer for more than twenty years. All that time I had been seeing end users struggle with software, using it inefficiently, ineffectively or only partially. I had been hearing them complain about their chosen tools, yet sometimes point-blank refusing to actually learn how to use them what I saw as 'properly'. This has always filled me with equal measures of frustration and incomprehension. The desire to 'do something about it' initially brought me to a rather ambitious research question, as follows:

Can academic theory and research contribute to documentation designers' efforts to support users of all types of software; so that these users acquire all the knowledge that they need to make full use of their tools, improving their day-to-day work and overall satisfaction?

Gradually it became clear to me that answering this question would involve developing a complete, evidence-based approach to documentation for all types of software and all types of user. Reluctantly I was forced to conclude that I had set myself too tall an order to be met in one PhD dissertation. This thesis is therefore restricted to laying the foundation for such a complete approach, by defining a framework within which documentation and documentation designs can be discussed in terms of what a particular product should aim to do to its reader, under what conditions, and why; as well as how it may be designed so that it can be expected to do what it aims to do.

This brings me to the following, somewhat more feasible, research question that this thesis sets out to answer:

RQ: Can design rationale for a coherent approach to the design of software documentation artefacts be found in a framework that describes the system dynamics of human performers interacting with, and learning from the interaction with, software and information?

This thesis purports to offer a reference framework for discussing software documentation artefacts in terms of their application as an intervention to a particular end. Such a framework is found in an abstract model of computer-mediated activity, which describes the problem space in which documentation designers move.

Figure 6 on page 24 shows the problem space in terms of a primary and a secondary task. This is taken as a starting point. An analysis of the secondary task, describing how people interact with and learn from documentation, is given in Chapter 3. In Chapter 4, the analysis is embedded in a wider analysis of the primary task, that of repeatedly interacting with software and learning from the experience. This is what I call the CMA model, where CMA stands for Computer-Mediated Activity. It is an abstract description of the problem space, which in Chapter 5 is shown to yield concrete, prescriptive considerations in real-life design contexts.

To move on to a possible solution space and to answer the research question, an attempt is then made to validate the CMA model. Can it provide design rationale for a coherent approach to the design of documentation artefacts?

One possible “coherent approach” is a design pattern language. A design pattern language is slightly different things to different people. Sometimes, it is taken to be no more than a collection of design patterns; where a design pattern, in the words of the visionary architect who first developed the idea, “describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem” (C. Alexander, Ishikawa, & Silverstein, 1977). It has however been proposed that for a collection of design patterns to be a language, the individual patterns need to share an underlying value system and an organizing principle (see Chapter 6 for a more detailed discussion). The value system underlying the pattern language describes the problem space in which designers move. The patterns themselves describe the solution space, and the organizing principle allows practitioners to use the pattern language when analysing a specific design context and creating specific design products.

The presence of an underlying value system and organizing principle common to the separate patterns is what can make a pattern language a “coherent approach”; more so than, for example, a set of guidelines or a style guide. To validate the CMA model as providing design rationale for software documentation, a tentative design pattern language is therefore presented: the Software Documentation Design Pattern Language or SDDPL. The SDDPL finds its underlying value system in the CMA model and its organizing principle in considerations of usability for practitioners. In Chapter 6, the pattern language is formalized. Three further chapters present a number of design patterns for software documentation, and the work is rounded off with a conclusion.

Users of software are as varied as the software they use. It is a mistake to think of ‘end users’ as always non-technically oriented, just as it is a mistake to think of them as forever novices. If, for example, a particular software product is used by seismic interpreters, then much more algorithmic detail is required in the documentation than in that for a word processor. Analysis of the intended audience is always the first step in the workflow of any professional

documentation designer. A close second is the analysis of the possible forms that the documentation can take: there is no law prescribing a book, nor one prohibiting the use of video, comics, or flash cards; to name but a few of the near-endless possibilities. These are design decisions, to be based on that which the documentation sets out to do—a user manual is after all a document with a job, and its quality can be judged only by how well it does that job. This work purports to provide a language in which a documentation artefact's design rationale can be discussed: not one by which an absolute label 'good' or 'bad' can be assigned to such an artefact or its components.

Chapter Contents

The overall research question is worked out in a number of chapters, each addressing a separate sub-question; as follows:

Chapter 2—The Case for User Documentation

SRQ: What is the relevance of the current work to society and academia?

This chapter first describes the problem that all disciplines involved with the support of software users aim to solve: the frequently-observed phenomenon of users failing to achieve full mastery of software-driven tools. It then makes a case for the design of user documentation as a legitimate topic of interest and discusses the relatively low profile of documentation design in the academic (although not professional) literature, evidenced by a lack of current research studies and theory-building.

Two small-scale studies have been carried out in the context of this chapter, each described in detail in an Appendix.

Concepts and theoretical frameworks discussed in this chapter include:

- the Paradox of the Active User
- yoking
- hierarchical task analysis
- design theory

Part of this chapter has been presented in a full conference paper at the 2013 World Conference on Information Systems and Technologies (WorldCIST'13), held on 27-30 March 2013 in Olhão, Portugal (van Loggem, 2013b).

The study referenced in this chapter and fully described in *Appendix 1. Software Documentation: A Standard for the 21st Century* has been presented in a full

conference paper at the Information Systems and Design of Communication (ISDOC 2014), held on 16-17 May 2014 in Lisboa, Portugal (van Loggem, 2014b).

The study referenced in this chapter and fully described in *Appendix 2. “Nobody Reads the Documentation”—True or Not?* has been presented in a full conference paper at ISIC 2014: The Information Behaviour Conference, held on 2-5 September 2014 in Leeds, United Kingdom (van Loggem, 2014a).

Chapter 3—The Documentation Journey

SRQ: What is known about the way in which people interact with documentation?

This chapter consists of a literature review, describing the secondary task in the field of user documentation design: that of referring to documentation when using a particular piece of software to a particular end.

Concepts and theoretical frameworks discussed in this chapter include:

- information behaviour

Chapter 4—A Model of Computer-Mediated Activity

SRQ: How can the process be modelled that software documentation is designed to support?

In this chapter, an abstract model of Computer-Mediated Activity (CMA) is constructed, describing the primary task in the field of user documentation design. The CMA model is assembled from a number of pre-existing models and theories describing human behaviour in naturalistic settings. It describes how people interact with software, and how they achieve mastery (that is, expertise) while doing so over time.

Concepts and theoretical frameworks discussed in this chapter include:

- Naturalistic Decision-Making and the Recognition-Primed Decision model
- cognitive constructivism
- the SRK framework of cognitive performance
- mental models theory
- satisficing
- Cognitive Load Theory
- uncertainty and ill-definedness

A first, more limited, version of this chapter has been presented in a full conference paper at the 31st European Conference on Cognitive Ergonomics (ECCE 2013), held on 26-28 August 2013 in Toulouse, France (van Loggem, 2013a).

Chapter 5—Documentation As Artefact

SRQ: How can a *generic, theoretical* model of computer-mediated activity (CMA) be instantiated to apply to *specific, practical* documentation design problems?

In this chapter, we move away from a description of the situation ‘as is’, before any artefacts are designed, to the goal that any such artefacts aim to achieve. The twin concepts of expertise and mastery are explored. As not all software systems are equally difficult to master, rather than using generic denominations such as ‘simple’ and ‘complex’, a more rigid three-tier categorization of software is given. This is then shown to provide a map of the requirements for the software’s documentation.

Concepts and theoretical frameworks discussed in this chapter include:

- activity theory
- expertise, mastery and knowledge
- use complexity

A first version of the section on use complexity in this chapter has been presented in a short conference paper at the 4th International Conference on Human-Centred Software Engineering (HCSE 2012), held on 29-32 October 2012 in Toulouse, France (van Loggem, 2012).

Chapter 6—SDDPL Value System and Organizing Principle

SRQ: How may a Software Documentation Design Pattern Language (SDDPL) be constructed on the foundation of the CMA model?

A pattern language is presented in which design patterns can be written that discuss the design of documentation artefacts with reference to the underlying values identified in the previous chapters.

Concepts and theoretical frameworks discussed in this chapter include:

- design patterns and design pattern languages
- usability
- Cognitive Dimensions

A first version of this chapter has been presented in a writer’s workshop at the EuroPLoP 2016 conference, held on 6-10 July 2016 in Kaufbeuren, Germany. The current version will be included in the Proceedings (van Loggem, 2016).

Chapters 7, 8 and 9—Reality Check

SRQ: Can proposed and existing documentation design solutions be expressed in terms of the SDDPL?

A number of proposed patterns taken from the academic literature are presented and described in terms of the SDDPL. In addition, a selection of existing documentation artefacts is mined for patterns and described in terms of the SDDPL.

Some of the patterns were mined during a ‘focus group’ held on 9 July 2015 during the EuroPLOP 2015 conference in Kaufbeuren, Germany (van Loggem, 2015). A full report of the focus group is given in *Appendix 3. Using the Repertory Grid Technique for Mining Design Patterns*.

Chapter 10—Discussion and Conclusion

The preceding chapters are discussed and conclusions are drawn. Furthermore, an agenda for further research is presented.

Notes

- In the opening chapters of this PhD thesis, some ideas are discussed that were already mentioned in my Master’s thesis (van Loggem, 2007). Where this was the case, the argument may have been phrased in similar or even identical terms in the two works.
- This thesis pulls together ideas, approaches, and theories from different disciplines. As a result, different concepts may be discussed that in the various separate literatures have similar or even identical labels. I have therefore habitually re-labelled existing terminology. Then, this thesis discusses people doing things. In such discussions it is inevitable that everyday-English words and phrases are given a meaning that is more restricted than their standard dictionary definitions. Whenever a word or phrase is re-labelled by me, or used in a more restricted meaning than its usual one, I say so explicitly and provide a definition delineating its use for the purpose of this work. Such terminology is italicised on first use in the body of the text, and the definitions are repeated in *Appendix 4. Terms List*.
- Throughout this work, hypothetical people of unknown gender (such as a *person*, *user*, *performer*) are referred to using the grammatically gender-neutral personal pronouns *he*, *him* and *his*. No value judgment is intended, nor should one be inferred.

■

2. *The Case for User Documentation*

Part of this chapter has been presented in a full conference paper at the 2013 World Conference on Information Systems and Technologies (WorldCIST'13), held on 27-30 March 2013 in Olhão, Portugal (van Loggem, 2013b).

The study referenced in this chapter and fully described in *Appendix 1. Software Documentation: A Standard for the 21st Century* has been presented in a full conference paper at the Information Systems and Design of Communication (ISDOC 2014), held on 16-17 May 2014 in Lisboa, Portugal (van Loggem, 2014b).

The study referenced in this chapter and fully described in *Appendix 2. "Nobody Reads the Documentation"—True or Not?* has been presented in a full conference paper at ISIC 2014: The Information Behaviour Conference, held on 2-5 September 2014 in Leeds, United Kingdom (van Loggem, 2014a).

Abstract: This chapter first describes the problem that all disciplines involved with the support of software users aim to solve: the frequently-observed phenomenon of users failing to achieve full mastery of software-driven tools. It then makes a case for the design of user documentation as a legitimate topic of interest and discusses the relatively low profile of documentation design in the academic (although not professional) literature, evidenced by a lack of current research studies and theory-building.

The Paradox of the Active User

Results of numerous studies (for a listing, see Bhavnani & John, 1997) suggest that expert usage of computer software does not automatically follow from either good design or any degree of experience of its users. Even the most experienced users of the best possible designs are at times seen to encounter problems, sometimes quite serious ones. Ben-Ari and Yeshno (Ben-Ari & Yeshno, 2006) make the following comment about a group of science teachers who use a particular word processor extensively in the course of their everyday work: "... almost all of them claimed that they were not fit subjects for the experiment because they were not expert users ... Considering the high levels of education and experience of the subjects, the most surprising result was the simplistic level of their interaction with this sophisticated but very familiar software tool ..." (p. 3). Fu and Gray note, in more formal terms but no less bemused: "From our analyses, people chose to use suboptimal procedures even when they apparently had knowledge of the optimal procedures, and thus have violated the normative principle of rationality" (Fu & Gray, 2004, p. 928).

Such observations tie in with the general awareness amongst technical writers, trainers and software support staff that people encounter difficulties coming to terms with the software they use to carry out their work. In an earlier work (van

Loggem, 2007), I distinguished three types of problem that are frequently encountered:

- *Inefficient use* is when the end result is what was wanted and needed, but the work took longer than necessary; without this being the result of fulfilling a particular desire. Applying multiple typefaces one after the other to the same piece of text in a word processor constitutes inefficient use when done because the user does not know how to get out of the dialog box presenting the font formatting options. On the other hand, when the typefaces are tried one after the other to judge their suitability, it does not.
- *Ineffective use* is when the end result is unsatisfactory while the software is perfectly capable of delivering perfection. Attempting to lay out a table inserting spaces between text elements resulting in columns not lining up constitutes ineffective use of a word processor—unless the program simply cannot handle tabular text.
- *Under-use*, finally, is when only part of the software's functionality is applied to the task at hand, and this is not the result of a conscious decision. There is under-use when a user enters his text without any formatting in a word processor that is capable of automatically laying out a complete publication, then saves the result as plain text and sends it off to a print shop to be set in Times New Roman with all the headings in bold. But there is no under-use when the user decides not to use any of the pre-defined layouts because they are judged to be unsatisfactory.

It has been shown (Charness, Tuffiash, Krampe, Reingold, & Vasyukova, 2005; Ericsson, 2005) that the acquisition of expertise requires many hours of deliberate practice. Practice makes perfect, as the saying goes, and expertise is regularly seen as a direct function of experience (e.g., van der Veer, Tauber, Waern, & van Muylwijk, 1985). However, it is that which we practise that we become perfect at: whichever strategies and concepts are used during performance of a task will be reinforced through repeated practice. This goes for optimal as well as suboptimal strategies and for correctly as well as incorrectly understood concepts. In the 1980s and early 1990s, Yvonne Waern and others carried out a number of learning-by-doing studies, to study the development (or not) of expertise by computer users who did not receive any form of training. In such a situation learning is a side-effect of attending to task requirements, Waern concluded. As such, it is tenuous. "People [...] may try to learn by doing, but they fail, and therefore need help from somebody else" (Waern, 1993, p. 333).

The phenomenon that people have considerable trouble acquiring all the knowledge that underlies expertise in the use of software, and that their skills tend to converge at relative mediocrity even after extensive practice, is so widespread that a phrase has been coined for it: the 'Paradox of the Active User' (John M. Carroll & Rosson, 1987). Originally referring to the paradoxical situation that while

people skip the learning stage in order to save time, they could save considerably more time later on by earlier investing some in learning, the phrase has stuck. Giving a slight twist to it, its meaning has since been neatly summarized as “the persistent use of inefficient procedures by experienced or even expert users when demonstrably more efficient procedures exist” (Fu & Gray, 2004). Software users are primarily interested in getting things done, yet all their practice fails to make perfect. After a certain degree of expertise has been acquired, no further progress is made. A beautiful example of the Paradox of the Active User was provided by one respondent in a study conducted in the course of my Master’s thesis (van Loggem, 2007), who estimated that carrying out a particular task using a complex software package took her on average one-and-a-half working days of extremely unpleasant and RSI-inducing work, spread out over one working week. The same task could have been performed in no more than twenty minutes, provided she set up the system appropriately—a one-off effort that could demonstrably be done in about half an hour. The task was one that she carried out five times per year. Yet it had never even occurred to her to see if any features might be available that would help her do what she was ultimately trying to achieve.

Carroll and Rosson, who first identified the Paradox of the Active User (1987), identified two underlying ‘biases’ working together so that even extensive practice still does not result in perfect performance: the production bias and the assimilation bias.

First, there is the *production bias*. People generally do not use computers just for fun. They have an ulterior motive, in that there is something that they want done. Their paramount goal being throughput, they have little patience with learning for learning’s sake, nor with activities that, once carried out, do not bring the desired goal noticeably closer. Although they may very well know that effort spent now will be of benefit later, they are driven by the wish to get things done. Whilst there is nothing wrong with wanting to get a job done—indeed, without such a wish there would be no motivation to attempt to use any tool in the first place—this desire may lead to required learning being skipped.

Definition: The *production bias* is the tendency to put short-term before long-term results when using a tool in order to meet an objective.

The second bias, the *assimilation bias*, follows from the first. In learning, people assimilate new knowledge into existing structures. These existing structures have been acquired at a sometimes considerable cost and are not easily replaced. People will go to great lengths to not give up on something that they think they know. When, driven by the desire to get the job done, the performer begins to compromise, the assimilation bias kicks in and seduces him to cling to actions that worked before in situations that seem similar to the current one, and to disregard dissimilarities or even plain undesired results.

Definition: The *assimilation bias* is the tendency to actively or passively ignore newly-acquired knowledge when using a tool in order to meet an objective.

The assimilation bias is not restricted to software users. Students of science, or indeed practicing natural scientists, have been shown to be just as reluctant to change the theories by which they explain the natural world. Indeed, so closely related are the two situations that well-respected studies have been carried out aiming to develop a model of the scientific reasoning process, in which the behaviour was studied of subjects trying to figure out the workings of a programmable device (Klahr & Dunbar, 1988). These researchers mention what they refer to as “a pervasive confirmation bias” (p. 3): when evaluating hypotheses (of how a system works), subjects focus on attempts at confirmation and over and over again fail to test potentially disconfirming instances. Neither do they change their hypothesis in the face of disconfirming outcomes (p. 41). Chinn and Brewer (1993) distinguish seven different ways of dealing with data that is incompatible with the established conceptions that constitute a currently held view. Only one of those is that of actually changing the incorrect theory of how the world works and this strategy seems to be not at all popular. The bodies of literature (in history of science, education, and psychology) studied by Chinn and Brewer present a rather discouraging multitude of students and learners on the one hand and practicing scientists on the other, opting time and again for one of the other six ways of coming to terms with what is perceived as anomalous data: ignoring it, rejecting it, excluding it, holding it in abeyance, reinterpreting it or, if there is really no other way out, yielding to it marginally by making only peripheral changes to non-core aspects of the current theory. These findings, and many others (e.g., Griffin & Ohlsson, 2001; Shulz, Katz, & Lepper, 2001) from the study of scientific reasoning and discovery, correspond closely to the assimilation bias that is encountered in computerised homes and workplaces.

Almost thirty years after it was first identified, the Paradox of the Active User is as predominant as ever. People still struggle with software, even when they have access to documentation and instruction. They still get stuck at a level of persistent suboptimal use or “asymptotic mediocrity” (Novick, Elizalde, & Bean, 2007). They still need help from somebody else.

User Documentation as the Cinderella of User Support

User documentation is the Cinderella of user support, regularly dismissed as irrelevant because only there to make up for defects in the software’s user interface. If the latter were better, the argument goes, the former would not be

required: so all our efforts should be geared towards improved interaction design. The dismissal is sometimes strengthened with the corollary that “nobody reads the manual anyhow”; after which the speaker turns away to attend to more pressing business.

It is my contention that the main argument and its corollary are both rooted in misconception. The ‘documentation as unnecessary evil’ as well as the ‘nobody reads the manual’ lines of reasoning can be easily disproved. User documentation is (or should be) a core deliverable of any but the most trivial of software systems (P. Wright, 1998): the need for it is not something that will quietly go away if ignored long enough.

“Well-Designed Software Needs No Documentation”

The first to suggest that good design of software user interfaces removes the need for user support was Donald Norman, who in 1988 re-defined the term affordance to refer to “...the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used. [...] When affordances are taken advantage of, the user knows what to do just by looking: no picture, label, or instruction needed.” (Norman, 1988). Although Norman went on to become one of the founding fathers of user interaction design, not least on the basis of this and similar appeals made in the same book, it cannot be ignored that the appeals were made with reference to ‘everyday things’ which were not software-based—such as water taps and, indeed, sliding doors. In 1999, Norman described the “single, general” personal computer as “a great compromise, sacrificing simplicity, ease of use, and stability for the technical goals of having one device do all” and as a “fundamentally difficult machine” (Norman, 1999). By trying to make one product do many things, Norman noted, complexity increases. Thus, complexity is built into any artefact that is not geared towards one specific, well-defined task. Many of today’s large-scale, sophisticated software packages certainly fit this description.

Norman concluded his 1999 book with a passionate cry for the computer to disappear into tools specific to tasks: “a world of information appliances”. Some ten years later this vision had become reality when Apple launched the first so-called tablet, the iPad™, which took the world by storm and was soon followed by similar devices from other manufacturers. A tablet is a flat, hand-held device consisting, as far as the user is concerned, mainly of a screen. The start-up screen is populated with so-called ‘apps’ that each perform one very specific task. Almost without exception, apps do not require documentation and indeed most come without. However, there is no evidence that multi-functional software is being replaced with app-like information appliances, presumably because there are always tasks for which a complete toolbox is required rather than one single tool. And indeed, at around the time of the spectacular rise of the iPad, Norman

dedicated a whole book to the necessity of complexity, in which he wrote: “I find it interesting that we complain when a new technology requires an hour or two of study. [...] When new items are appropriately complex, it is reasonable that they require time and effort to master” (Norman, 2010, p. 31).

It is highly unlikely that a company such as Microsoft Corporation leaves the design of the user interaction of its products to amateurs. We must assume, rather, that Microsoft’s products are carefully designed by competent professionals, striving for the highest possible usability. Yet it is Microsoft’s flagship word processor Word™ that was used in Ben-Ari and Yeshno’s study (Ben-Ari & Yeshno, 2006) and that prompted these authors’ expression of surprise at the manifestation of the Paradox of the Active User that they witnessed. Word™ is complex, not unnecessarily so because it was badly designed; but necessarily so because it is a very sophisticated tool that can be put to a multitude of uses—including some that its designers could not even dream of. To give an example: Word was almost certainly not designed with an explicit view to supporting people in the task of creating cross-stitch patterns. Still, I have personally witnessed a friend’s mother-in-law doing exactly this.

For those systems that are necessarily complex, insights from earlier years (van der Veer et al., 1985) will continue to hold: that there are types of information regarding the system that the user must know and that cannot be communicated within the user interface but for which documentation is required. For this reason alone documentation will always be needed. But there is another reason.

Figure 1 shows ad-hoc documentation for a sliding door; obviously created to meet a real-life requirement. Arguably, the door was designed badly. But it was there in the cold winter of 2010-2011 at a railway station in the Netherlands: and people were, quite literally as well as figuratively one presumes, stuck with it.



*Figure 1: Instructions on how to use a sliding door:
This is a sliding door. Therefore do not push or pull! Door does not
automatically open and close. (Photo courtesy Ester Moraal, 2010)*

Bad designs do happen, in the world of software as well as that of sliding doors. Moreover, the market mechanism almost guarantees that bad designs will continue to happen. Almost all software is developed to satisfy the needs and requirements of those who pay for it; which is not necessarily those who use it. Systems that do things such as controlling traffic lights, supporting the trade in stocks and bonds, or keeping track of the location of lab coats in hospitals is what we might call 'non-consumer software': and unless it is employed in high-risk environments such as aerospace, the additional design expenditure following from attention to usability, user satisfaction or ultimately user experience is unlikely to ever be reclaimed. Being not economically viable, these matters will remain forever neglected. The imperfect world in which we live contains many badly-designed products that people have to deal with, whether they like it or not.

So, although unnecessary complexity is the inverse of usability and would ideally be removed through good design of the user interface, this cannot be an excuse for refusing to pay attention to documentation. Unnecessary complexity will always exist and badly-designed products will always have to be documented. Then, even in a perfect world in which all software designs are of the highest possible standard, a need for documentation will arise from the necessary complexity that is a consequence of versatility. For these two reasons, the idea that the need for user documentation will eventually fade away is a “persistent myth” (Mehlenbacher, 2003).

“Nobody Reads Documentation, Anyhow”

The respondents who caused Ben-Ari and Yeshno (Ben-Ari & Yeshno, 2006) such puzzlement over their lack of expertise almost certainly worked with a version of Word™ that came equipped with online Help, and possibly with an extensive printed user manual as well. Although Microsoft stopped delivering printed user manuals when Word97 came on the market in December 1996, every version of this popular word processor thus far has come with online Help. Does this then mean that users, as has often been said (Rettig, 1991), ‘simply won’t read the instructions’? Documentation developers the world over secretly hope for permission from management to have printed on every page the catch-phrase, ‘Read The F... Manual!’ or RTFM for short. Yet a respondent in a study on the appropriation of the iPhone was quoted to say: “I don’t have the time to explore. I miss having a manual that I can lean back and read.” (Bødker & Christiansen, 2012).

A handful of research studies have been carried out to determine whether users are indeed reluctant to consult the documentation that is delivered with the product, and they are surprisingly unanimous in their findings. All cast at least reasonable doubt on the assertion that documentation is ignored. Invariably, it is shown that—at least for complex and unfamiliar products—the documentation *is* consulted; even if it is not read, marked, learned, and inwardly digested in its entirety. *Appendix 2. “Nobody Reads the Documentation”—True or Not?* gives an overview of existing research, then describes a study carried out in 2013/2014 when sixty-nine students at an institution for short tertiary education and thirty employees of a developer of scientific software for the interpretation of seismic data were questioned as to their use of documentation and other sources of information on the use of software. The results again offered strong evidence that reports of the death of user documentation have been greatly exaggerated. Depending on what exactly is counted, between 24% and 80% of information-seeking behaviours consist of recourse to documentation. Interestingly, all the numbers attempting to describe frequency of recourse to documentation in this study were higher for the (better educated, older) group labelled ‘professionals’ than for the (less educated, younger) group labelled ‘students’. Although the

sample sizes were relatively small and therefore it is not known whether they extrapolate to the general population, this can be seen as encouraging. If professional users of software are as willing to consult documentation as the findings suggest, then taking pains to design and develop documentation of the highest possible quality is a worthwhile endeavour.

In addition to the results of studies such as these, there is another strong indication that users of software do not necessarily avoid consulting documentation, or have given up altogether on the idea of getting help from an acknowledged authority: we can see them vote with their feet. A visit to any bookshop will show long shelves filled with software-related books (van der Meij, Karreman, & Steehouder, 2009; D. Wright, 2008). These books, many of them running to hundreds of pages, bear witness to a genuine demand. If there was no market for them then they would not be there. But instead, people go out and buy. This particular market is a thriving one.



Figure 2: Tag line printed on the covers of books in the 'Missing Manuals' series, published by O'Reilly, Sebastopol, CA, USA

Documentation as a Design Discipline

Whether the persistent suboptimal use of software is indeed a paradox in the dictionary meaning of the word or not, it is without doubt a highly undesirable state of affairs. If software systems in organizations are routinely used inefficiently, ineffectively or only partially, then the economic consequences cannot be other than serious. Man-hours wasted; incorrect or invalid results; reduced acceptance of the system after implementation; that what has been paid for not being fully utilized: all these are known to cost an organization dearly¹.

¹ Hard proof of this claim is difficult to find. The following quotation offers no more than anecdotal evidence: "I once commissioned a study tracing all efforts to correct a faulty invoice sent to the customer. The cost of correcting a mailed invoice was 20 to 30 times the cost of an error-free invoice. We could not find any computer errors ... All errors were human errors, which in each instance originated from inadequate training or poor management practices." (Strassman, 1990, p. 314) More anecdotal evidence is provided in the trade press (e.g. Gartenberg, 2005).

For this reason, practitioners in different disciplines have been called up to fight the Paradox of the Active User and provide the help that Yvonne Waern (Waern, 1993, p. 333) called for. They do so by *design*. Although design is sometimes understood to refer to the aesthetic aspects of artefacts, in a more general sense it refers to the intentional creation of a tangible or intangible product for application at a later time, often (but not necessarily) by somebody other than the designer.

Definition: *Design* is purposeful endeavour resulting in a particular product for the benefit of a human performer.

Targeting the tool: HCI design

One possible target for improvement through design is the software tool itself. This is the responsibility of those who design the user interface (UI), also referred to as the Man-Machine Interface (MMI) or the Human-Computer Interface (HCI); where the letter I is frequently understood to stand not for ‘interface’ but for ‘interaction’.

HCI design is concerned with the usability and usefulness of software-based artefacts. The ISO 9241-11:1998 standard defines usability as “The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.” Enhancing usability has different aspects. One approach is to incorporate meta-communication. This road is taken when meaningful names are given to commands in a command-line interface or CLI and to controls in a direct manipulation interface or DMI, or when the appearance of currently unavailable choices is made to dynamically change (‘dimming’). Another has to do with making the right way obvious and the wrong way as good as unthinkable. Unfortunately, the suggestion that problems with the use of software could be eradicated through cleverly-designed user interfaces has in later years shown to be wishful thinking for all but the most simple software systems (as discussed on p. 13).

Targeting the user before the work is undertaken: instructional design

Not only the task environment is open to improvement by design. Another strategy targets the user; either before the work is undertaken, or during its execution.

Before the work is undertaken, instruction can be delivered: either through a teacher (classroom training) or through self-study. This undoubtedly improves future performance, and instructional design is a discipline that can draw on decades of experience in as well as a vast body of literature. Instruction can however not easily move along with increasing expertise over time, and is most naturally administered to novices.

Targeting the user whilst the work is undertaken: information design

User support during execution of the work confronts the Paradox heads-on, filling the knowledge gap between that which is known and that which is required through the provision of *information*.

Definition: *Information* is knowledge that is implicitly or explicitly formulated so as to be of value to a human being.

Such user support through information can again be designed in different ways. Information on how to work with the software can be offered personally, one-on-one, delivered to users at the moment they need it and pertaining to their individual problems. In this setting a particular question or problem is formulated by the user and put to another person. Depending on whether the person so approached is another user or a representative of the software manufacturer, we speak of a user community or of a Helpdesk (for an interesting discussion of the latter, see Steehouder, 2002).

Targeting the user whilst the work is undertaken through pre-recorded information: documentation design

Learning can take place not only through instruction *before* performance, but also through practice *during* performance. If we accept that achieving mastery of much software requires learning, and also that generally speaking software users cannot be expected to expend time and energy on the act of learning *per se*, then we must devise appropriate tactics in our designs.

The information on how to work with the software can also be recorded and stored beforehand in 'information artefacts': physical or electronic objects that are created to express ideas and meaning such as books, drawings, photographs, audio recordings and websites (Marchionini, 2010). These are created not to meet one particular request for support but to meet all future requests: one-on-many. This is what I refer to as *documentation*: a design approach to counteracting the Paradox of the Active User that targets the user during the work through one-on-many information artefacts.

Definition: *Documentation* is pre-recorded information that is purposefully selected and presented to assist future users deploying a particular tool.

Documentation can be delivered as a standalone product but it can also be embedded inside the software: in the form of messages that provide feedback on interactions, or as 'tooltips' that display a brief explanation when the mouse hovers over a particular software control. With the increased availability of processing power, it has even become possible to present embedded information

in a personal-seeming manner, depending on the particular context of use but still pre-recorded and stored. Microsoft's (now defunct) 'Office Assistant', remembered by many as 'Clippy, the talking paperclip', is a well-known example of such an approach. Embedded information may be considered documentation just as much as information that is delivered separately from the software. For practical reasons, however, the focus in this thesis will be almost exclusively on the latter: standalone documentation artefacts are simply more easily available and accessible as study objects.

Figure 3 below shows the various design disciplines working to combat the Paradox of the Active User.

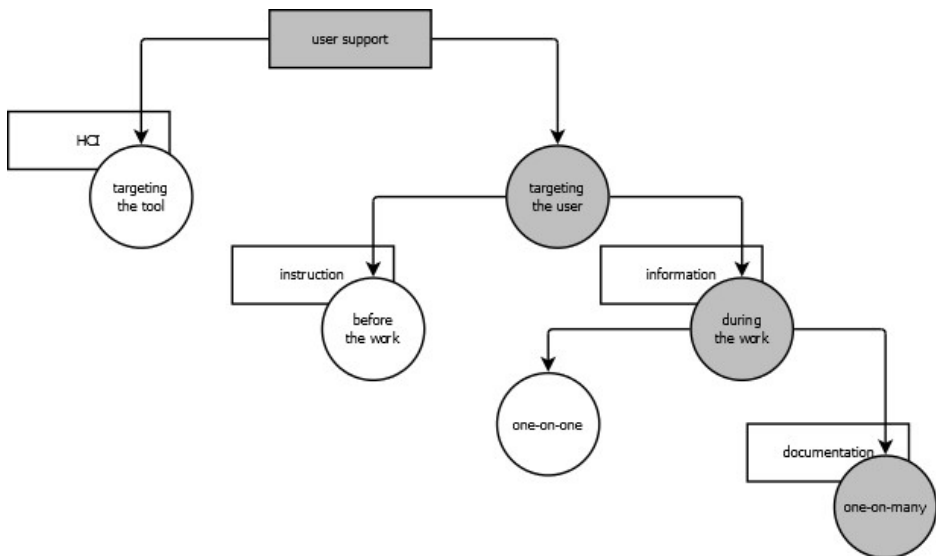


Figure 3: Positioning software documentation amongst other design disciplines for user support

Primary and secondary tasks

Regardless of the discipline in which they work, designers find it useful to distinguish between primary and secondary tasks; the primary task being that which someone sets out to achieve, and the secondary task that which follows from any design put in place to enable completion of the primary task. Architects, for example, design an environment that fulfils the desire for shelter and to move from 'inside' and 'outside' and back again at will. To meet this desire, every architectural design will contain at least one doorway. An industrial designer may then come in to design a door, at which point a secondary task is created: that of operating the door. Here is where a tension between user desires and needs is first

established. The designer now attempts, through the content of his design, to maximize support for the primary task while, through its usability, minimizing the disruption caused by the secondary task.

In line with the traditional view of design as creating a means to a pre-existing end, difficulty in learning to use computers has been attributed to there being a gap between the software world and the outside world that needs to be bridged. Software design disciplines focus on bridging the gap. This is routinely achieved by first carrying out a task analysis describing the current situation, labelled 'Task Model 1' (van der Veer, Lenting, & Bergevoet, 1996) or 'Descriptive Study I' (Blessing & Chakrabarti, 2009). Then, when designing the new task environment incorporating the new tool, the aim is to achieve a close parallel between the 'before' and 'after' situations; making the secondary task as transparent as possible. The design of the new situation is described in a 'Task Model 2' or 'Descriptive Study II', which ideally is user-tested and adjusted where required before full-scale implementation. After implementation, again in an ideal situation, the performance of users is then evaluated to determine the value of the solution. Task Models 1 and 2 are described as a hierarchical structure of main tasks and sub-tasks, and user performance is judged against some 'ideal' task performance. This is an extremely popular approach, extensively described in the literature as "hierarchical task analysis". Various methods have been developed, usually known by an acronym and often over the years elaborated and refined: MAD (Sebillotte, 1988), GOMS (Card, Moran, & Newell, 1983), TAG (Payne & Green, 1986), ACTA (Militello & Hutton, 1998) and many others. This focus on separation of task and tool, of primary and secondary task, is reflected in measures of learnability and complexity of tools. The user is then, for example, seen as 'yoking' the problem space (the real-life situation) and the device space (the software world), by mapping tasks in the problem space to manipulations in the device space. As the gap to be bridged becomes wider, yoking becomes more problematic (Payne, 1992; Payne, Squibb, & Howes, 1990). In line with the idea of the primary task being if not known then at least knowable, the concept of yoking pre-supposes the feasibility of ultimately establishing a mapping between tasks in the problem space to manipulations in the device space; as does an early measure of the complexity of software from a user's perspective (D. Kieras & Polson, 1985).

But tool and task can become indistinguishable. Even when they are not, any tool will still qualitatively change the nature of the task to which it is applied (Bødker, 1991; John M. Carroll, 2000; Kaptelinin & Nardi, 2006). Over the years, software has become ever more sophisticated. In the first decades of computer use by the general public, people used computers mostly for reproducing their work, for automating tasks that they could at least theoretically have done without a computer. Nowadays more and more software enhances people's work in such a way that it enables them to do non-routine things that were unthinkable before

(Mirel, 1998a). Bødker & Bertelsen (2003) distinguish three scenarios (see Figure 4).

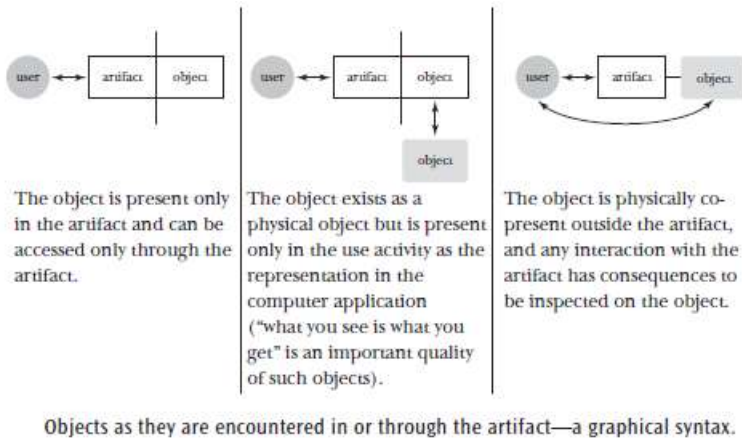


Figure 4: The positioning of the object with regard to the software (taken from Bødker & Bertelsen, 2003, p. 307)

In the first, the object (that is: the final outcome of the work done) is contained within the software world and has no presence in the outside world, while engagements with the software have consequences in the software world only. The other extreme is the scenario in which the object exists in the real world only and has no presence in the software world, while any engagement with the software has consequences to be inspected on the object. In between the two extremes Bødker & Bertelsen envisage the scenario in which the object exists in the real world while a representation of it is held in the software world. It is this representation that is continually being changed through engagement with the software.

When the object exists in the software world only, as shown in the first of Figure 4's three descriptions, the work is undertaken with the sole objective of creating a particular software state, stored in a file or a set of files. People create spreadsheets to help plan their annual budgets, their vegetable gardens or their cross-stitch projects; they create websites, page-by-page or through a Content Management System; they send and receive electronic mail; they create long, camera-ready manuscripts; they write smaller or larger programs; and they set up software environments for themselves or others to work in. For this type of activity there no longer exists a distinction between the task and the tool: rather, the task can be defined as working with the tool. A software tool's new capabilities may well lead to people coming up with new uses to put it to, in which case a task

analysis undertaken before the tool is created will by definition fail to capture the full range of activities that is eventually carried out with the software. And just like tasks, sub-tasks may originate from the software: for example, when a word processor introduces the concept of templates or an image processor that of layers.

In all these examples, the idea of a task analysis undertaken *before* the software is designed becomes untenable, and traditional measures of performance—which also pre-suppose yoking—no longer suffice (Frøkjær, Hertzum, & Hornbæk, 2000). How can yoking describe the difficulties encountered when writing a program, for example, or creating camera-ready copy? Suboptimal performance can now be evidenced not only by inefficiency or ineffectiveness, as before, but also by an overly narrow *intention horizon*; which encompasses that which the user sees as possible applications of a tool's capabilities, and beyond which he does not perceive any information or situation as relevant to the work at hand. When the intention horizon lies within the total functionality of the software, under-use of the software will be the result.

Definition: The *intention horizon* is the perimeter of the solution space that a user perceives to be offered by a particular tool.

Using documentation is not a primary task, as is, for example, moving between outside and inside, or writing a report. It is not even a secondary task, as is operating a door to go inside, or using a word processor to write a report. Using documentation is in fact a tertiary task: documentation must sometimes be endured in order to use a tool (such as a sliding door or a word processor) in order to fulfil a desire (such as going inside or writing a report). And it does not always end here. Some software artefacts are so complex that in order to use them to their full potential, users must find their own solutions to problems they encounter; and this they can do only after learning the ins and outs of the system. The learning process then becomes a task in its own right, turning the interaction with documentation into a quaternary (!) task. Now as many as three steps removed from the original motivation, as far as the user is concerned the documentation might easily become nothing but yet another obstacle between his desires and their fulfilment (see Figure 5).

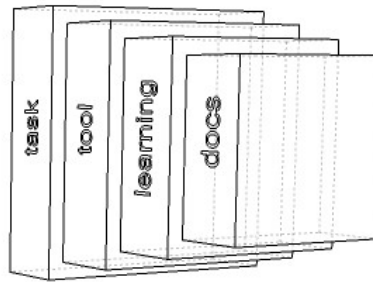


Figure 5: Documentation as a quaternary task

Put like this, the whole idea of designing documentation that works sounds rather hopeless. But when we collapse this task hierarchy so that only a (re-defined) primary and a secondary task remain, we are back in familiar design territory (see Figure 6). I propose to describe the primary task of software users as computer-mediated activity or CMA; legitimized by the fact that task and tool are always interwoven. Their secondary task is then the ‘documentation journey’: the engagement with documentation as and when required. Learning is understood as being part and parcel of both primary and secondary task and not as a separate activity.

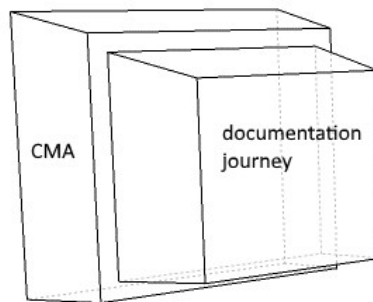


Figure 6: Learning through documentation as a secondary task

A key characteristic of documentation is that the communication between the creator of the documentation—in the context of a particular documentation artefact more commonly referred to as the product’s author—and the user in his role of reader is asynchronous. Sender and receiver of the message are separated in time and place. As a result, individual reader differences and preferences cannot be catered for. Although the information contained in software documentation

may be presented to the user interactively, its scope is always predetermined. Also because of its asynchronous nature, documentation is used at the reader's discretion. The author cannot enforce the quality or quantity of use and must depend on the readers' willingness to use it, and to do so in the way the author envisioned (Novick & Ward, 2006). Referring to Figure 5, we must acknowledge that no user can be forced to work his way through the whole pyramid. He may easily go round part or all of it, dispensing with the documentation and perhaps with the learning process as well. This is the challenge facing designers of documentation.

Although we can never 'force-feed' information, we can certainly attempt to 'drip-feed' learning where required. It seems worthwhile to try and support unsupervised practice in ways that contribute not only to the user's immediate goal but—where appropriate—equally well to the development of true expertise, of being able to tackle situations of ever-increasing complexity successfully and satisfactorily; so that over time, practice makes perfect.

The Need for Research

Throughout the centuries, people managed to become expert users of complex tools such as, for example, the astrolabe, the loom and the combine harvester. They read the instructions where available, then practised and became perfect. The Paradox of the Active User entered our lives only when software-driven tools became commonplace. War stories about struggles with personal computers, VCRs and mobile telephones have become part of our popular culture (see, for example, Figure 7). This suggests a fundamental difference between physical tools and software-driven tools that may not always have been fully appreciated. We shall come back to this many times: this current work takes the existence of such a fundamental difference as pivotal to all attempts to defeat the Paradox of the Active User.

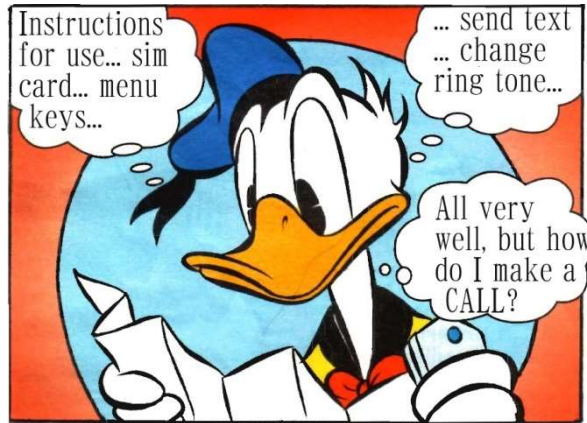


Figure 7: Donald Duck fighting a mobile telephone and its instruction manual (translated into English from the Dutch original publication)²

If the problem is not that users are unwilling to ‘RTFM’ then it must be that their reading the manual just does not help them enough. It will be necessary to challenge established beliefs and to re-think the design of documentation artefacts on a fundamental level. In the words of Mark Baker, a professional documentation designer who developed the ‘EPP0’ approach to software documentation (see page 117):

Technical documentation is a decision support system. Insofar as it fails to support the user’s decision making, it fails its purpose, even if all the physical procedural steps are documented correctly. (Baker, 2013, p. 95)

The presence of user documentation in itself is, as we have seen, not enough to defeat the Paradox of the Active User. The way in which a documentation artefact is designed can qualitatively change the way it is read (P. Wright, 1998). There are strong indications that the single most relevant predictor of success in learning new technology through documentation is not a person’s demographic background, experience or education but rather the documentation’s quality (Schriver, 1997, p. 458). ‘Quality’ of documentation, like that of other products, is achieved by following a design path rather than making ad hoc decisions as and when the need arises (P. Wright, 1994). Unfortunately, documentation designers’ knowledge and skills have not developed at the same pace as the software that they document (P. Wright, 1998). Software in the 2010s is a far cry from that in the

² story H28339, © Disney, 2009, reproduced with permission

1980s, when guidelines for documentation were first formulated. Yet user documentation in the 2010s looks very much as it has looked for a long time.

A typical user manual or Help system is hierarchically structured with the chapter as its main unit. Sometimes the chapters are grouped into sections, or groups of chapters are split off into separate documentation artefacts. In this manner, the ‘installation manual’ and the ‘advanced user’s guide’ may come about. Each chapter contains an operational description of and (usually stepwise) instructions for working with a particular program concept or user task. Conceptual descriptions are added to place that which follows in context; in addition, a number of introductory chapters or appendixes provide program-wide meta-information by listing hardware requirements, providing installation and troubleshooting instructions, and informing the reader of terminology and typographical conventions used in the manual.

This prototypical approach to software documentation has remained unchanged for a number of decades. It is led by guidebooks and standards originally dating from the 1980s. Although regularly updated and added to, such guidelines tend to change only slowly and superficially and by their nature cannot reflect the latest developments. Moreover, they largely ignore the particular problems posed by software-driven as opposed to physical tools.

In the last decade of the 14th century, a 10-year old boy addressed as “little Lewis” who was the son of Geoffrey Chaucer (or perhaps of one of Chaucer’s friends) wanted to master the astrolabe. To help him, Chaucer started to write an instruction manual, describing the use of this highly complex tool (Chaucer, 1391). The work was never finished, but enough of it was written for us to see that most if not all of the recommendations laid down in the above-mentioned standards were already faithfully followed by Chaucer. Structure and tone of voice of the “Treatise on the use of the astrolabe” do not differ significantly from those applied in present-day instruction manuals, be they for physical tools or for software-driven artefacts. Similarly, it has, for example, proven quite possible to validate an instruction manual for a table loom, published in 1925, to ISO/IEC Standard 26514:2008 (*Systems and software engineering — Requirements for designers and developers of user documentation*) and establish a high level of conformance (see *Appendix 1. Software Documentation: A Standard for the 21st Century*). The Standard’s focus turns out to be on form rather than function.

In this, the Standard is not alone. The quality of software development documentation, for example, is often expressed in terms of numerical values of measurable properties (Abran, Desharnais, & Cuadrado-Gallego, 2012) which can be quantified automatically (Dautovic, Plosch, & Saft, 2011). Extension of such ideas into the domain of user documentation then results in attempts at automatic assessment or even generation (Olaverri-Monreal, Dlugosch, & Bengler, 2013) of user instructions. But development documentation has very little in common with

user documentation. Development documentation is created for developers, who have technical knowledge of the underlying implementation and need to know the details of how the software was built. User documentation, on the other hand, is created for end users. This is a completely different audience which does not need to know how the software is constructed, but rather how to use it. Development documentation is really about the software, which is measurable and quantifiable. User documentation is really about the end user and his learning requirements—which are not.

There is no reason why the traditional user manual or Help file described earlier should remain the only genres of user documentation. The authors of commercially available computer books certainly do not restrict themselves to it. On the contrary: they develop new genres in abundance. They experiment with placing screen shots central to the whole of the discourse³, come up with so-called ‘cook books’ revolving around re-usable solutions that readers/users can copy and modify to match their own projects⁴, and generally speaking allow their imagination to be limited only by perceived usefulness to their readership. Although users hold such works in much higher esteem than they do documentation that is delivered with the software (M. D. T. de Jong & Karreman, 2017), whether their new designs are indeed useful, and if so, under which conditions, is not known. The need for theory-based and evidence-based guidance on how to document different types of software in an appropriate manner is felt both by the documentation design community and by the beneficiaries of its efforts: the users of software (Schrivier, 1997, p. 227). This need has not sufficiently been met by the academic community.

Like a true Cinderella, documentation is regularly overlooked as ‘not sexy’. Theories of learning tend to be swept up to form a starting point for the design of classroom instruction and self-study materials, while theory and experimental results from cognitive psychology have been taken up by the applied field of interaction design; which is where the interest of computer science in user support ends. Library and information science is highly system-oriented and has an established tradition of focusing on formal information systems for document retrieval from repositories rather than on information retrieval from documents (Vakkari, 1999). Communication science, finally, covers a broad area of which technical communication form but a small part and the design of documentation, be it for software tools or other artefacts, an even smaller part.

Even when research into documentation is carried out, this tends to embrace new technology-enabled media of communication without questioning. Almost as soon as it was recognized that written instructions are vital for optimal performance,

³ as exemplified by the “Visual Technology Book” series (published by Wiley, Hoboken, NJ, USA)

⁴ as exemplified by the “Cookbooks” series (published by O'Reilly, Sebastopol, CA, USA)

the research focus shifted from paper-based documentation to hypertext and soft copy (P. Wright, 1998). Since then, it has shifted further: to the World Wide Web, then to CSCW or ‘computer-supported cooperative work’, to social media, and at the time of my writing this to mobile appliances and ubiquitous computing channels for the delivery of information. Experimental research (such as Byrd & Caldwell, 2011; Spannagel, Girwidz, Löthe, Zendler, & Schroeder, 2008) and theory-building (e.g. de Souza, Marconi, & Dyson, 2008; Selber, 2010) tend to focus on the deployment of a particular medium or modality. Lured away by the beckoning of glamorous new media in cyberspace, researchers seem sometimes to overlook fundamental yet less exciting matters such as the content of the documentation or the application of printed publications, and “do not always make clear the boundary conditions within which their findings will apply” (P. Wright, 1998). Research questions ask *how* a particular medium of communication can most fruitfully be deployed; glossing over the question *when* (that is, under which conditions) or *why* any effects that are found may hold. The subject of Karen Schriver’s seminal work (Schriver, 1997), which on publication almost immediately became the thinking technical writer’s one-book-library, is *document* design rather than *documentation* design: a subtle but important difference as it presumes that certain choices have been a priori made.

A documentation artefact is a form of non-fictional, instructional writing: the fields of communication science and library and information science are heavily involved, as are those of instructional design and cognitive psychology. Finally, the creation of software documentation requires technical and business knowledge of computer use and information processing. This multi-disciplinary nature is something that documentation design has in common with the other user-support disciplines shown in Figure 3. But unlike HCI, instruction, and information, documentation is not an established academic discipline. There exists no recognized ever-growing body of knowledge, constructed by an established academic community that is served by multiple dedicated conferences and journals.

Documentation has remained a niche interest: as a consequence, the literature in the field is fragmented. There are very few obvious places to look for publications reporting on research studies and theory-building. To ‘keep up with the literature’, one will need to diligently perform searches in search engines covering a large number of disciplines. Since every imaginable keyword has other dominant meanings, weeding out irrelevant search results is a non-trivial exercise. Searching for publications using the search term ‘instructions’, ‘manuals’, ‘guides’, ‘documentation’ or ‘information’ yields publications relating to almost every topic under the sun other than the one intended (P. Wright, 1998); see also Figure 8.

- ★ 1. [Russian Orthodoxy: A User's Manual. Church Advice Literature as a Reflection of Present-Day Reality](#)
 Bettina S. Weichert
The Modern Language Review, Vol. 102, No. 2 (Apr., 2007), pp. 451-465

Figure 8: The problem with database searches for 'user manual'.

The fragmented nature of the literature in the field makes for an incoherent body of knowledge. Even a researcher who has gone through the laborious search process will find it difficult to relate the different publications, which may appear in places such as the *International Journal of Audiology* (Caposecco, Hickson, & Meyer, 2014) or *Computers and education* (Nistor, Schworm, & Werner, 2012), to each other. Indeed, the field as a whole does not seem to make much progress. Only two methodologies for the creation of documentation have been proposed in the past decades that constitute a clear deviation from the time-honoured format described earlier. The first of these is the minimalist approach, which applies specifically to software documentation (see p 137). Unfortunately, research as to the effectiveness of this approach has been sparse and limited to documentation specifically geared towards novice users (van der Meij et al., 2009). The second attempt to improve the traditional user manual is Information Mapping™, for the documentation of software and non-software products alike. Although it was claimed that the effectiveness of this commercially marketed and jealously guarded methodology has been corroborated by academic research, closer inspection makes short shrift with any such claims and the method has been shown to perform no better than any other (Jansen, 2002; Jansen, Korzilius, le Pair, & Roest, 2003).

Before we can think about designing interventions to guide a process towards a particular outcome, we need to understand the mechanism of the process when it runs its natural course. To know the conditions under which a particular design solution is useful, such a solution must be built upon a naturalistic and descriptive model on an abstract level (Kirlik, 2006, p. 7). The informed design of software documentation demands that the choice for medium and format of the communication, as well as its content, be based on an understanding of the underlying processes of people interacting with software and its documentation (Schrivver, 1997, p. 389; P. Wright, 1998). Only then can we begin to understand the mechanisms underlying the Paradox of the Active User; to attack the Paradox purposefully through documentation designs; and to express design rationale (MacLean, Young, Bellotti, & Moran, 1991a, 1991b; Moran & Carroll, 1996) underlying documentation design decisions.

When user interaction design was still a fledgling field, dreams were dreamt of finding "a set of basic design rules of the following kind: IF user(*i, j...*) AND task (*k, l...*) THEN apply design principles (*m, n...*) where *i, j* stand for variables that

characterize particular (groups of) users, k, l for task variables, and m, n for design principles [...]” (van der Veer et al., 1985). In a similar vein dreams should at least be dreamt, even if turning them into reality will prove as elusive as it has been for interaction design, of a set of basic design rules expressed in the form IF user_behaviour($i, j \dots$) AND software_characteristics($k, l \dots$) THEN apply design_principles ($m, n \dots$); where i, j now stand for variables that characterize the way people interact with software and its documentation; k, l for variables characterizing the inherent complexity, from the user’s point of view, of a particular piece of software; and m, n for design principles in user documentation. What is needed is a framework within which the dreamt-of complete set could be expressed: insight is needed into the nature of the parameters $i, j \dots, k, l \dots$ and $m, n \dots$.

In this thesis I propose such a framework, in the shape of an abstract description of people interacting with software and documentation. Such a model will not magically dispel the Paradox of the Active User. But it will provide a vocabulary with which documentation artefacts can be described, discussed, and designed. This will give Cinderella the support she needs to put on her slippers; so that she can go to the ball.

3. *The Documentation Journey*

Abstract: This chapter consists of a literature review, describing the secondary task in the field of user documentation design: that of referring to documentation when using a particular piece of software to a particular end.

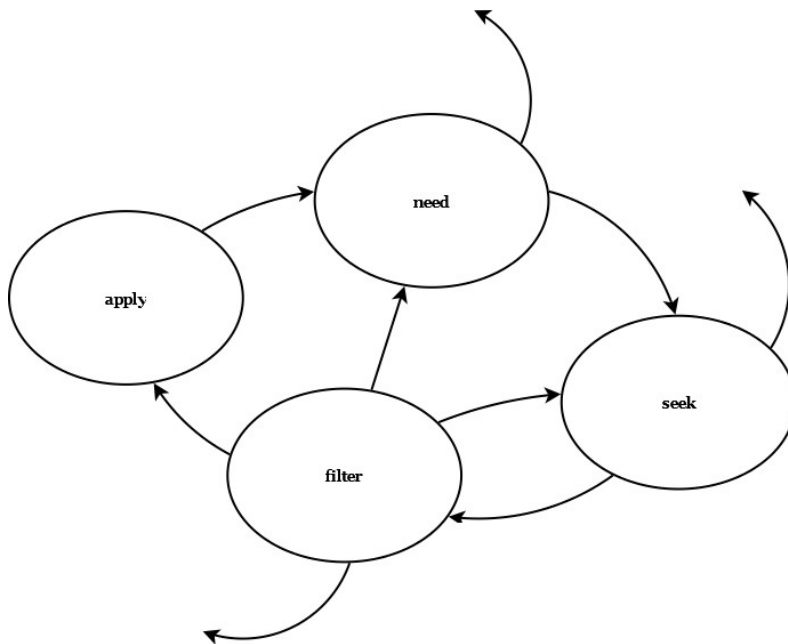
Interaction with Documentation: the Secondary Task

The way in which people interact with information became a focus of academic interest in its own right in the early 1900s, under the generic names of ‘information interaction’ or ‘information behaviour’. Initially, in the first two-thirds of the century, most if not all research into information behaviour was conducted from a viewpoint of not the users of the information but the artefacts in which it resides and the venues in which it is sought (Case, 2007). Information and Library Science (ILS; also known as Library and Information Science or LIS or simply as Library Science or LS, or as Information Science or IS) was concerned with information in the shape of books: their description, indexing, archiving, cataloguing and disclosure. Information Retrieval (IR) then became interested particularly in formal querying of mainly electronic information systems, such as databases and electronic library catalogues. Both ILS and IR are highly system-oriented and their focus is on formal information systems (Vakkari, 1999). Exemplary of IR has long been the series of experiments known as the ‘Cranfield experiments’ (Cleverdon, 1970; Cleverdon, Mills, & Keen, 1966), which aimed to increase the effectiveness of electronic catalogue systems by employing better indexing languages and methods. These experiments were conducted without any involvement of actual people carrying out the queries and judging the relevance of the documents found.

More recently the focus has shifted and widened to become more naturalistic and person-oriented, placing the search for information in the particular context in which the information is sought. Blandford and Attfield (2010) speak in this respect of an ‘information journey’. Figure 9 shows the information journey generalized and narrowed down to the case of interaction with not any type of information but specifically with that relating to the use of a (software) tool: the *documentation journey*.

Definition: The *documentation journey* is the search for information on the deployment of a particular tool.

Figure 9 identifies four separate stageposts along the journey, but these are not necessarily visited sequentially. Return loops in Figure 9 reflect the fact that the information journey is not mapped out completely beforehand. Instead, it is iterative and self-shaping (Marchionini, 1995). Pieces of information that are found change the direction of subsequent behaviours. Kuhlthau (Carol Kuhlthau, 1991; Carol Kuhlthau, 1993), for example, sees the information journey as cyclic and aimed at a gradual refinement of the information need. In Bates' 'berry-picking' model of information behaviour (Bates, 1989), an information seeker is likened to a hunter-gatherer looking for food, in this case, berries. Once a bush carrying berries is encountered, its berries are picked; further berries are looked for depending on the quality and quantity of what the first bush has yielded. For the information gatherer, this means that an often poorly-defined need for information leads him to a first titbit, which then is applied not only to the situation from which the original need arose, but equally well to the need itself. The next piece of information may then be picked from the same location within the same source; from a different location within the same source; or from a completely different source.



*Figure 9: The documentation journey (based on:
Blandford & Attfield, 2010)*

The berry-picking metaphor, in which information gathering is likened to 'browsing' in the original meaning of gathering food, is part of a persistent

tradition. Miller (Miller, 1983) described information gathering as an instinctive behaviour not just in humans but in all higher organisms and coined the term 'informavore', describing man as an 'information eater'. Ever since, seeking information has been likened to food gathering. After berry-picking, browsing, and 'grazing' (Blandford & Attfield, 2010) entered the literature, a recent elaboration of the metaphor is the Information Foraging Theory (Pirulli & Card, 1999). This sees information seekers as moving from one 'patch' of information to the next driven by 'scent', provided by the description of a remote information 'patch' that is present in the current one. The available information patches become apparent to the forager only after he has embarked on the hunt for information. Again, in this model the requirements are re-shaped and re-directed as the process unfolds.

An information seeker is under no compulsion to visit all the stageposts but can decide at any moment to abandon the journey altogether. Based on a certain amount of empirical evidence, Atkin (Atkin, 1973) describes a model in which "an individual will select a [mass media] message when his estimate of its reward value exceeds his expectation of expenditures involved in seeking or avoiding it." (p. 237) Taking a slightly different approach, Case (Case, 2007) even sees *all* searches for information ending in the subject deciding to give up, as the potential amount of information on any topic is infinite so that there is always more to be known. A more mundane consideration is that the information may be simply unfindable even when present, or not presented in such a manner that the user can understand it or apply it in context. Arrows pointing 'nowhere' in Figure 9 show that just as we can lead a horse to water but we cannot make it drink, we can offer a person information but we cannot make him do anything with it. All that we can do is, depending on how the information journey is ideally shaped, make the preferred 'route' easy whilst others are downplayed.

Returning to the literature on Information Behaviour, it is possible to describe the information journey in more detail. Using its own nomenclature in accordance with the objectives of this work, Figure 10 shows for each stagepost one or more aspects that form a part of its description. Whereas the stageposts are visited one after the other in a sequence indicated in the diagram by arrows, the aspects come into consideration in no particular order.

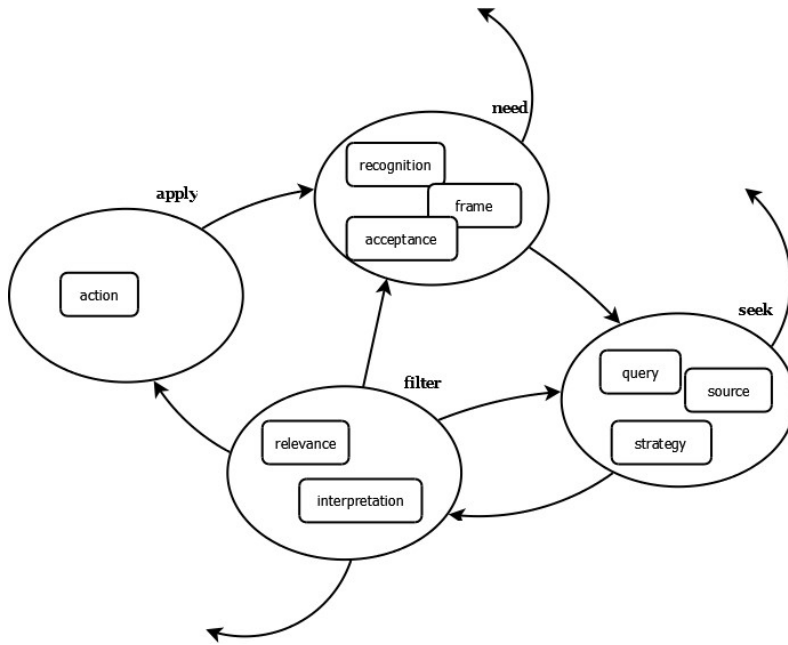


Figure 10: The information journey in greater detail

Stagepost 1: Needing Information

People interact with information in the service of some broader activity, as a consequence of wishing to satisfy a goal (Wilson, 1999). Recognition of an information need has been described as “a recognition that your knowledge is inadequate to satisfy a goal that you have” (Case, 2007) or “recognition and acceptance of an information requirement” (Marchionini, 1995). In a food-gathering metaphor, the need for information would correspond to a person’s appetite. An information need may be the expression of a recognized anomaly in the state of one’s current knowledge (Belkin, 1982) or a sense of uncertainty regarding (Atkin, 1973; Carol Kuhlthau, 1993) or dissatisfaction with (Taylor, 1962) the situation as it presents itself to the individual. Atkin (1973) pinpoints the experienced sensation further, by distinguishing extrinsic uncertainty, that is, related to the situation, and intrinsic uncertainty, or relating to the individual’s current state of knowledge.

recognition

The descriptions vary and so does the specificity with which the various authors regard the nature of the information need (Case, 2007). Almost constant however is the mention of recognition of the need for information. Existence of need is a necessary condition for embarking on an information journey but it is not a sufficient one: inadequate as an individual's current state of knowledge on a particular topic in a particular context may be, if he is not aware of the fact he may experience the need to satisfy a goal but he will not experience the need for information. Without recognition of an information need, no information is sought.

Taylor (1962) distinguishes the visceral information need from the conscious need. The former is the actual, as yet unexpressed, need for information, whereas the latter is the "conscious within-brain description of the need". Case (2007) points out that, precisely because of their subjectivity once experienced, needs are open to discussion: It is quite possible to be mistaken as to the exact nature of one's own needs, or even to be unaware of one's 'true' needs. Indeed, Taylor notes that the conscious need may lead to discussion with others, in order to refine it further before attempts are made to meet it.

frame

Once an information need is recognized, it begins to take shape in the subject's mind. The ambiguity that characterizes the conscious need is phased out by further refinement of the question until, still according to Taylor, a formalized need has been reached at. This is framed so as to be rational and properly qualified. A conscious or subconscious selection is made as to which aspects of the conscious need are included and which are discarded. Inevitably, nuance will be lost, as the subject is now in Taylor's terminology three steps removed from the 'real' need.

acceptance

Finally, before a subject can begin seeking the information required to alleviate his felt need, he must make a decision to do so. At any time, a subject may simply decide not to bother; to ignore and live with the information need, without attempting to do anything about it.

Stagepost 2: Seeking Information

After an information need has presented itself, information is sought. This has been described as "a conscious effort to acquire information in response to a need

or gap in your knowledge” (Case, 2007), or “a subset of information behaviour that includes the purposive seeking of information in relation to a goal” (Spink & Cole, 2006). Information seeking, then, is undertaken in order to alleviate an information need: in a food-gathering metaphor, this stagepost would correspond to obtaining the food.

Note that information may also be accessed without a prior awareness of need and therefore in a non goal-directed manner, simply ‘for the fun of it’ or even for lack of anything better to do. Aimless browsing of a magazine or undirected clicking to follow one hyperlink after another on the Internet is common human behaviour, not driven by a felt need nor a requirement for information. Although such drifting through an information landscape is not information seeking, it can develop into it: when not given up in favour of a totally unrelated activity, it may become more focused at the moment a salient piece of information is encountered. The first inkling of an information need is then beginning to make itself felt and this may grow, so that an information journey is begun.

strategy

Very different behaviours may be displayed at the seeking stage. A general distinction can be made between on the one hand directed search (White & Roth, 2008), information search (Atkin, 1973), or simply searching (Carol Kuhlthau, 1993) as opposed to on the other hand non-directed search (White & Roth, 2008), information receptivity (Atkin, 1973) or browsing (Carol Kuhlthau, 1993). Sometimes, non-directed browsing is seen as a stage prior to directed searching (e.g. Carol Kuhlthau, 1991), although all authors stress the cyclic nature of the information behaviour and agree that non-directed browsing and directed searching may alternate more than once during one information journey.

The two strategies can both be described in a more fine-grained manner. For example, Ellis (Ellis, 1989, 1993) describes a number of information behaviours, three of which take place during seeking: chaining (following cross-references), browsing (scanning a document to see what’s in it) and differentiating (organising the sources). In this thesis no distinction is made further than that between ‘searching’ and ‘browsing’.

Definitions: *Searching* is directed information seeking, undertaken to find the answer to a well-formulated question. *Browsing* is interaction with information sources that is driven by a general desire for as-yet unspecified knowledge related to a particular topic.

As Belkin points out (1982, 1993), missing knowledge is difficult to formulate. Because people can’t easily express what they don’t know or what is missing, questions submitted to information systems based on an individual’s request often don’t adequately represent what is needed. It is for this reason that if an

information source needs to provide knowledge that its readers do not know is missing, it should support, facilitate and encourage browsing as well as searching.

query

The term ‘query’ is often associated with electronic information systems, and linked exclusively to searching (which, according to one definition (Spink, 2010), is the act of “entering queries often created from multiple words and making judgements about the retrieved documents or websites”). However, a query or question put to the information system is just as well formulated during browsing, and/or when accessing printed materials. While searching is driven by a more-or-less strictly delineated query, browsing comes from a more general desire for ‘finding out about’ or FOA (Belew, 2000). Still, even then there is an awareness of what it is that needs finding out about and an attempt to formulate a question. On consciously turning to sources of information, there is always a moment where “the question is recast in anticipation of what the files can deliver” (Taylor, 1962). This results, after the visceral, the conscious and the formalized stages, in what Taylor describes as the fourth stage of a need and which he refers to as the compromised need. ‘Compromised’, because practical considerations shape the need into a query matching the expected capabilities of the information system. The query is phrased to match what the inquirer thinks he will get out of the information sources.

This recasting of the question is visible in experienced search engine users who add to the query words or phrases that they expect to be present in the type of result they are after, for example by including a non-English word when looking to buy a product with an English name in a non-English-speaking country. It is equally manifest in people asking a librarian for information in terms that are much more generic than those in which their own formalized need is expressed. However, when it is not immediately obvious that a query is formulated to match expected capabilities of the information system, this does not mean that no adaptation has taken place. Without such adaptation, very few attempts to locate information in a printed manual through its index or even Table of Contents (TOC) would ever be successful.

source

Refining and enhancing a number of previous studies, Agarwal, Xu and Poo (2011) conducted an investigation into the information seeking behaviour of 352 professionals in a wide range of disciplines, which confirmed that information seekers demonstrate a strong and stable ranking of preference for information sources of different types. Whether it was the order of use that was measured or the amount or frequency of use, most popular was online information gathering, then face-to-face, then via phone or chat, then via e-mail or forums, and only as a

last resort through books or manuals. Similar results were obtained in the (much smaller) study described in *Appendix 2. "Nobody Reads the Documentation"—True or Not?*. Unfortunately, as is the case in many sourcing studies (Novick et al., 2007), the categorization used by Agarwal, Xu and Poo was rather coarse and not strictly defined. Examples given by the authors to characterize 'online information' included nothing more specific than "Google, company digital library, intranet" (p. 1092) so that it is not clear whether an online Help system would in this categorization qualify as 'online' or as 'manual'—which is all the more unfortunate as these two types of information source proved the most and the least popular, respectively. In line with results found almost forty years earlier (Gerstberger & Allen, 1968), source quality and access difficulty proved important antecedents of source use, with seekers placing more weight on source quality as the importance of the task increased. Note that 'access difficulty' in the 2011 study was defined as "the time and effort required and the difficulty encountered in reaching a particular information source" (p. 1089); not the time, effort and difficulty encountered in reaching a particular piece of information *within* a source. Likewise, Gerstberger and Allen in 1968 considered the 'accessibility' of a number of 'information channels' rather than the accessibility of the information contained in a document. In the context of documentation, the latter concept is of more interest, as the documentation artefact as a whole is a given: it is either readily accessible, or not present at all.

Stagepost 3: Filtering Information

Filtering the information is not simply selecting a subset of that which is available: it is a transformative process. In a food-gathering metaphor, filtering the information would correspond to preparing the food that has been obtained. Once found, information is matched against existing knowledge and against the situation that made the information need felt. This is a 'sense-making' process: is the information relevant to me and the situation I am in, here and now? Can I internalize it, fit it in, apply it? Should I keep it in the back of my mind, or use it now? Or is this perhaps something to be ignored altogether? By the time the information is acted upon, it may be not just quantitatively but also qualitatively changed. Two of Ellis' (Ellis, 1989) information behaviours take place in the filtering stage: extracting (identifying fragments for further use) and verifying (checking accuracy and reliability).

relevance

Judgments of relevance of the information that is found shift during the course of the information journey (Carol Kuhlthau, 1991). Moreover, such judgments are, as

are most elements of the information journey, subjective. What one information seeker judges to be relevant may not be so to another one, even if both were to experience exactly the same information need. For this reason a distinction is sometimes made between relevance, which is then taken to be an objective judgment ('does the information apply?') and the more subjective concept of pertinence ('is the information useful to me?') (Kemp, 1974).

interpretation

Newly-found information is organized and lined up so as to fit in with the information seeker's current knowledge base. Details may be lost or created during this part of the process, when information is interpreted to align with rather than contradict earlier-held views (Spink, 2010) (Spink & Cole, 2006).

Stagepost 4: Applying Information

The result of filtering the information in the previous stage is translated into a decision as to what to do next. In a food-gathering metaphor, applying the information would correspond to eating. Implementing this decision changes the environment, which in the situation under consideration is a software-supported task environment. The next chapter will look at this environment in more detail.

4. *A Model of Computer-Mediated Activity*

A first, more restricted version of this chapter has been presented in a full conference paper at the 31st European Conference on Cognitive Ergonomics (ECCE 2013), held on 26-28 August 2013 in Toulouse, France (van Loggem, 2013a).

Abstract: In this chapter, an abstract model of Computer-Mediated Activity (CMA) is constructed, describing the primary task in the field of user documentation design. The CMA model is assembled from a number of pre-existing models and theories describing human behaviour in naturalistic settings. It describes how people interact with software, and how they achieve mastery (that is, expertise) while doing so over time.

Interaction with Software: the Primary Task

So far, we have considered the documentation journey: how people interact with information sources. This is the secondary task depicted in Figure 6. For every secondary task, there exists a primary task. In this chapter, we will turn our attention to that primary task, that is, the context in which the documentation journey is undertaken. This primary task is what documentation aims to support: and in order to design documentation that does so, its characteristics must be known

No comprehensive model of human interaction with software is readily available, but separate pieces of the larger picture are. Combining insights from various disciplines that study human behaviour in naturalistic settings (psychology, instructional science, cognitive science, risk analysis and management, information behaviour, and economics) like so many pieces of a jigsaw puzzle, a complete model can be assembled of Computer-Mediated Activity or CMA. The admittedly eclectic nature of the resulting model allows it to cover the entire field of interest. All its constituent parts have been tried and tested in naturalistic settings, and their validity as ways of looking at various aspects of human cognition 'in the wild' is widely accepted: all are regularly applied in disciplines other than those they originated in.

CMA being human behaviour, variables in the CMA model cannot plausibly be quantified. The model is therefore a qualitative one. It is built up step by step, taking into account one or two pre-existing insights at a time. Starting point is a generic positioning of CMA as repeated decision-making. The model is then in subsequent sections gradually refined and expanded.

Each section contains the following components:

1. A grounding in the literature of the part of the model under discussion.
2. An influence diagram showing system dynamics. Parts that have been previously developed are dimmed. The direction of influences is marked with a plus or minus sign; if an arrow indicates flow or sequence rather than influence, it is not marked. Influences manifesting themselves after a time delay are indicated with dashed connectors.
3. A summary of the main implications of the part of the model under discussion in words and, insofar as possible, through one or more monotonic functions. (A monotonic function shows how increase or decrease in one entity causes another entity to increase or decrease. For example, the monotonic function $X = M^+(Y)$ says that if Y goes up, then so does X; and $X = M^-(Y)$ says that if Y goes up, X goes down.)

CMA as Repeated Decision-Making

When working with software, a potentially large number of interactions is carried out one after the other; where the decision to choose one particular *interaction* and not any other takes into account the feedback from the system as a result of the previous interaction. Computer-mediated activity can be viewed as repeated decision-making as to which interaction to carry out next.

Definitions: An *interaction* is an observable behaviour, carried out by a user to direct the functioning of a software-driven system.

Theories and models of naturalistic decision-making have been developed that describe how human performers choose a course of action in naturalistic settings. Because of their origins and applications in high-stake environments such as aviation and the military, such models are typically associated with high risk and time pressure. However, there is evidence for them to apply in all situations where decisions are made that are not presented as choice between alternatives and where the decision problem has a significant perceptual component (Lipshitz & Pras, 2005). It is thus with confidence that I take a representative model, known as the Recognition-Primed Decision model, as the basis of this section.

Theories of naturalistic decision-making assert that in naturalistic settings, decisions are not made based on detailed reasoning. Rather, a process often known as ‘situation assessment’ but which I prefer to label *situation processing* is carried out, during which a *situation model* is dynamically constructed, driven by recognition of perceived cues from the environment against the backdrop of pre-existing structures in the *knowledge base*.

Definition: The *knowledge base* is the total of a performer's current knowledge in a particular domain.

Situation processing is characterised by a focus on sensemaking, on coming to terms with the situation. External cues are perceived, comprehended and finally projected (through the predominant interaction that is part of the situation model) into the near future (Klein, 2006a). Re-framing takes place until the situation model is felt to be satisfactory. The driving force in perception as well as comprehension is recognition —hence the name 'Recognition-Primed Decision' model. Furthermore, selection of an interaction is incorporated in situation processing from the very beginning.

The situation model is a representation of a human's knowledge and understanding of the present state of the system (Endsley, 2000a, p. 2), or, the operator's internal model of the state of the environment (Endsley, 2000b, p. 4). The situation model includes not just objective features and subjective judgements but also possible actions that can be undertaken. This is different from other, 'rational' models of decision-making, that ascertain a more or less conscious lining up of the possibilities, then evaluating them one after the other and finally selecting the most promising one. In the Recognition-Primed Decision model, one or more possible interactions form an intrinsic part of every situation model that is constructed. The final situation model may then be subjected to a mental 'trial run', a thought experiment in which the current model is evaluated and the consequences of the predominant interaction are explored. However, such a trial run takes place only if the situation is perceived as worthy of further deliberation. When recognition is strong, the most 'obvious' interaction that is part of the situation model, the one most strongly recognized as being appropriate, will be carried out immediately (Klein, 1989, 1993; Lipshitz & Pras, 2005).

Definitions: A *situation model* is an internal model of the current state of the environment as perceived, comprehended, and projected into the near future. *Situation processing* is the process by which a situation model is constructed.

(In the literature, the situation model is also referred to as Situation Awareness, with initial capitals. Situation Awareness or SA for short is then the product of the process referred to as situation assessment or sa, in all lowercase. There are a few problems with this convention. The two phrases are very similar and when abbreviated, the distinction depends exclusively on capitalization. Furthermore, it is not immediately obvious which of the two refers to the process and which to the product. Finally, and perhaps most importantly, both are also used to describe sometimes very different concepts.)

It is interesting to note that naturalistic decision-making is not the only field in which recognition rather than rationality is seen as the driving force behind decision-making. Theories of embedded cognition maintain that perception of and

interaction with the world underlie all human cognition (Prinz, 1997). This assertion forms the cornerstone of much work carried out in robotics and cognitive science, where a mainstream tendency is to model behaviour based on recognition of the environment, without the intermediate layer of a separate control structure. Such models, in which a straightforward response to the environment is the main drive for action, often provide a close data fit, lending them ecological validity (see, for example, Taatgen, Huss, Dickison, & Anderson, 2008).

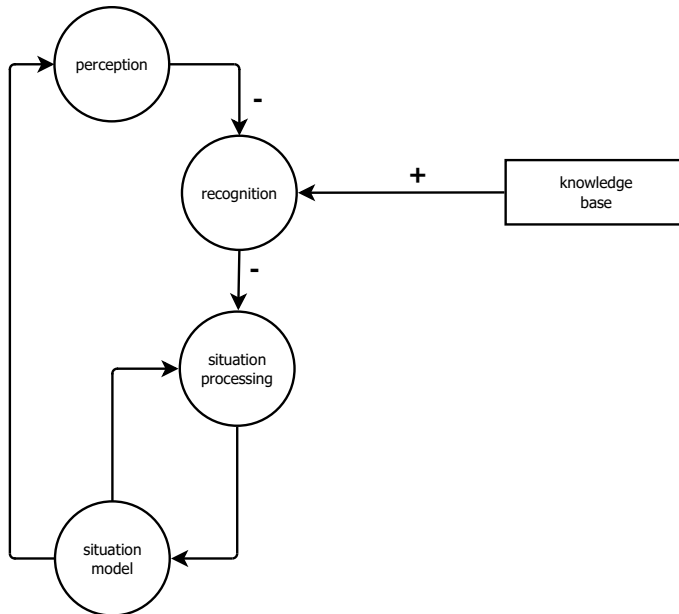


Figure 11: CMA as repeated decision-making

To summarize: Computer-mediated activity can be seen as repeated processing of the current situation as it is perceived, resulting in a mental situation model which includes an interaction. Recognition, fed from the knowledge base containing all the knowledge that the performer currently possesses, dampens situation processing.

$recognition = M^*(perception)$
 $recognition = M^*(knowledge)$
 $situation\ processing = M^*(recognition)$

Learning From Practice

Until relatively recently, instructional practice was built on the assumption that, through the simple provision of information, knowledge can be transferred from one who possesses it (the trainer or teacher) to one who as yet does not (the trainee or student). Over the past half century instructional theory has come to accept what is commonly referred to as the 'cognitive constructivist' view, which is that meaningful, applicable knowledge is not passively received either through the senses or by way of communication but is actively constructed by the learner, based on their interpretations of experiences: that is, practice (von Glasersfeld & Massachusetts Univ, 1989).

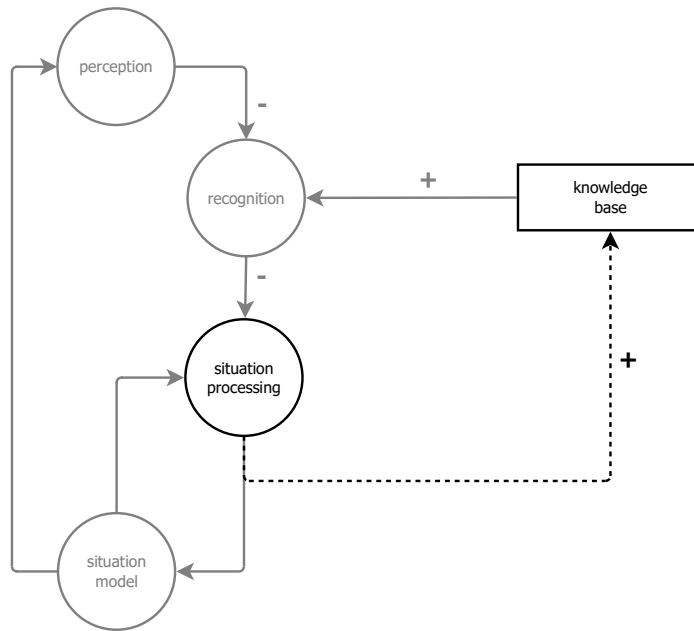


Figure 12: Constructivist learning during CMA

Cognitive constructivism asserts that information is not the same as knowledge. Rather, information is the input to a process of which the output is knowledge. Note that constructivism does not express an opinion as to the quality of the knowledge that is constructed. When experiencing, people *will* learn: even if it is only that the sun revolves around the earth and will continue to do so until the end of time. We live and learn, but the learning will be neither necessarily 'good' nor 'bad'.

Cognitive constructivism is incorporated into the CMA model by allowing for the knowledge base to expand as a consequence of the cognitive effort embodied in the situation processing.

Just like every other process, situation processing has a quantitative and a qualitative aspect. Quantity of situation processing is the amount of effort that goes into it. Were it this alone that results in additional knowledge (i.e., learning), then any struggle during CMA would cause the performer to learn. In reality, hard work does not guarantee successful completion of a task, let alone learning from the experience. To further understand the learning mechanism in the CMA model, it proves fruitful to examine not quantity but quality of the situation processing.

The SRK (Skill, Rules, Knowledge) framework developed by Jens Rasmussen builds on a number of previously developed models of human reasoning and decision-making. It describes three levels of cognitive performance, from very specific, inflexible and effortless to ever more general, flexible and effortful (Vicente & Rasmussen, 1992).

1. The most elementary level of performance in the SRK framework is skill-based (SB) performance. Routine interactions in a familiar situation are carried out without much thought: the situation model is constructed immediately by matching the external cues to a structure in the knowledge base on the basis of recognition. It will contain only one interaction, which is carried out without further situation processing. Conscious attention is applied only when an error becomes apparent, at which time performance will move up to the next, the rules-based or RB level. A person performing at SB level will find it difficult, if not impossible, to formulate a rationale for his interactions. The interaction to be undertaken seems so obvious that no alternatives present themselves (Reason, 1990; van der Veer et al., 1996, p. 316). The reply to the question, "Why did you do that?" will run along the lines of, "there is nothing else I can do", or, "this is how it's done".
2. When an error is detected that cannot be quickly and easily corrected, the performance level shifts one up to rules-based (RB) performance. Some more effort now goes into situation processing: the knowledge base is actively searched for a structure that seems to match the situation. This level of performance is called 'rules-based' because no matter what shape or form knowledge structures in the knowledge base may assume, what the performer is interested in is applying a 'rule' that runs along the lines of, "*if the situation is like [XXX] then do [YYY]*". Justification for selecting a particular interaction can be given when asked and will take the form of straightforward reference to something stated as fact (Reason, 1990).
3. The third and highest, most effortful, level is knowledge-based (KB) performance. When there seems to be really no alternative, the performer will move up yet another gear into this level, where situation processing is

thorough. Rather than trying to avoid error situations, the performer now actively seeks a route to success, if necessary from first principles. He analyses abstract relations between structure and function in order to gain new insights. Again (Reason, 1990), justification for selecting a particular interaction can be given when asked and will take the form of creative thinking and conjecture.

The SRK level at which people need to operate is directly related to the complexity of a particular domain and inversely related to their expertise in that domain. A salient feature of expertise is that relatively much of its performance takes place at SB and possibly RB level. As expertise increases, performance can and will take place on lower levels. At the same time, the knowledge involved in the situation model that is being constructed will be qualitatively different (Goodstein, Andersen, & Olsen, 1988); see also Chapter 5.

Expertise being that which follows from learning, if SRK level of performance is inversely related to expertise and performers strive to keep performance at the lowest possible level, then moving up to performance at higher levels must be a precursor to learning. I shall use for this the term *upgrading*.

Definition: *Upgrading* is undertaking situation processing at a higher, more effortful SRK level of cognitive performance.

Upgrading to situation processing at KB level can lead to the construction of truly new knowledge structures that are accessible to future performance at RB level. Upgrading to situation processing at RB level can lead to existing knowledge structures being embellished with corollaries or contingencies. When existing knowledge structures are so modified and fine-tuned, they may eventually allow for future performance at SB level.

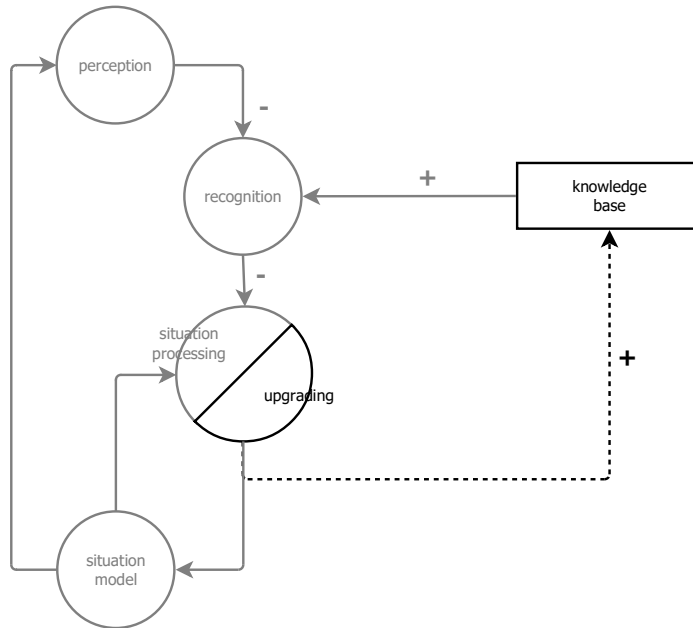


Figure 13: Learning through upgrading in CMA

To summarize: Upgrading to higher levels of cognitive performance during situation processing results in learning.

$$\text{knowledge} = M^+(\text{upgrading})$$

Satisficing and Ambition in CMA

How much situation processing and upgrading is a performer willing to do? More than a half century ago, social scientist and later Nobel laureate (economics, 1978) Herbert Simon realized that ‘economic man’, who is completely rational in his decisions and choices and makes them after fully evaluating all the possible alternatives together with their pay-offs and results, lives in a world far removed from ours (Simon, 1955). In the real world inhabited by real people, many decisions and choices are made on a good-enough basis. Human performers tend not to continue reflecting on a real-life problem until the best possible solution has been found. People do not usually optimize. Rather, they make do with the first adequate solution that presents itself: they *satisfice*, a word adopted by Simon to denote striving for good-enough rather than optimal results.

Satisficing works by the deliberate introduction of simplifications into the model of the current situation, in order to bring that model within the range of the performer's capability. One such simplification concerns the pay-off of interactions. Pay-off, asserted Simon, is rated not on a continuous scale, but as one of two (unsatisfactory or satisfactory) or three (bad, acceptable or good) values. A performer's so-called aspiration level determines at which point a choice is deemed to yield sufficient (satisficing) pay-off. The satisficing choice is then acted upon and no further effort is expended on finding an even better solution.

Definitions: *Satisficing* is an approach to decision-making in which no more cognitive effort is expended than that which is required to arrive at a decision that meets the current aspiration level.

In a situation that is dynamic in that the pay-off of a particular interaction in a sequence depends not only on the immediate result but also on the ultimate goal of the sequence, the aspiration level takes into account long-term as well as the immediate pay-off. Thus, quality and quantity of the situation processing are related to the performer's short-term and long-term *ambition*: that which he initially set out to do.

Definition: The *ambition* is the intention with which a performer embarks on a course of action. It has a long-term and a short-term component.

Aspiration level and ambition may vary from performance to performance. In general, they tend to rise when the individual finds it (subjectively) easy to come up with satisficing interactions. When on the other hand finding a satisficing interaction is felt to be difficult, aspiration level and ambition may drop. This means that if the end result of the CMA (the *value*) is felt by the performer to outweigh the situation processing involved in achieving it (the *load*), then the next time round the initial ambition may aim higher, so that further situation processing can still take place. Conversely, if the value is not felt worth the load, the next time around the initial ambition will be set lower, so that less situation processing will be required to come up with a situation model that meets it.

Definitions: The *value* of an activity is the worth that the performer attaches to the situation that results from it. The *load* of an activity is the effort that the performer experiences in carrying out the activity. Value and load are both subjective.

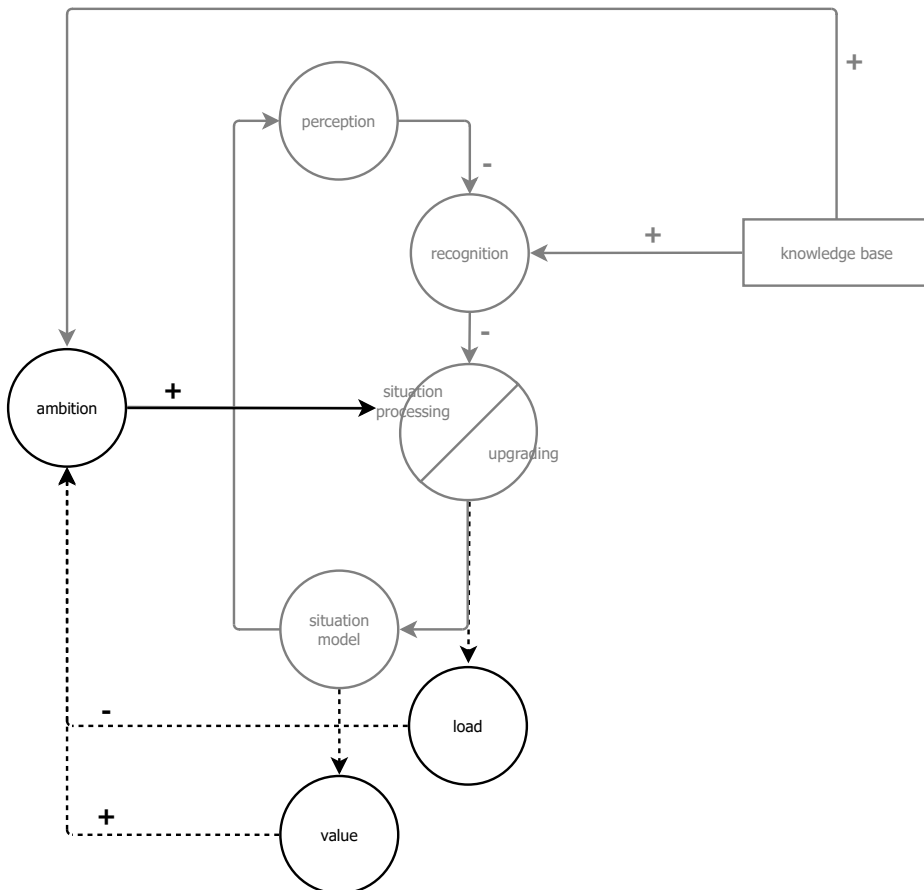


Figure 14: Satisficing during CMA

To summarize: Situation processing is driven by an ambition. Over time, the ambition will go up when experienced value exceeds experienced load, and down when it does not.

$$\text{ambition} = M^+(\text{value})$$

$$\text{ambition} = M^-(\text{load})$$

$$\text{situation processing} = M^+(\text{ambition})$$

Cognitive Load in CMA

The fact that subjective load has a negative effect on subsequent ambitions is problematic, as learning results from upgrading to more effortful processing, which is driven by ambition; but which also, by definition, constitutes cognitive load. At first sight this seems a Catch-22: learning inevitably involves cognitive load but this lowers future ambition, which in turn reduces subsequent learning.

Fortunately, things are not quite so clear-cut. Cognitive Load Theory (Sweller, van Merriënboer, & Paas, 1998; van Gog, 2006) is based on the assumption that human cognitive architecture is organized around (1) a virtually unlimited long-term memory where knowledge is stored, and (2) an extremely limited working memory where information processing takes place. No (further) learning can occur when working memory is overloaded. Cognitive Load Theory distinguishes three types of (objective) cognitive load to be taken into account. First of all there is the *intrinsic* cognitive load that is part and parcel of the subject to be learned and cannot be manipulated. Then there is the cognitive load that is a function of the design of the learning environment. Some of it is *extraneous*, in that it springs from the design without serving any purpose. The 'good' cognitive load, finally, is called *germane* and this consists of all the processing in working memory that is necessary for learning to take place. Germane cognitive load contributes towards learning whereas extraneous load detracts from it (Paas, Renkl, & Sweller, 2004). Therefore, instruction must be designed so as to make much room as possible in working memory for germane cognitive load, by reducing extraneous and (where possible) intrinsic cognitive load (van Gog, 2006).

Within the framework of the CMA model, non-intrinsic cognitive load is germane when it is part of upgrading in a situation in which such upgrading is required, but it is extraneous when not related to upgrading, or when the upgrading is not necessary for (further) learning. Being a theory of formal instruction, Cognitive Load Theory is concerned mainly with reducing non-germane cognitive load. This is different from a context in which learning is a by-product of practice, yielding real-life results. In the latter case, not all cognitive load is subjectively felt as a burden. Work done that leads somewhere, to better results or greater insight, can feel like an interesting challenge rather than 'load'. The value then not only counterbalances the load but even negates it.

To summarize: 'Load' being subjective, it is reduced when the cognitive effort involved in the situation processing is pleasant and worthwhile. Removing as much extraneous and intrinsic cognitive load as possible further reduces the subjectively experienced load.

Mental Models and the User Virtual Machine

Let us now turn to the user's 'knowledge base', which in the CMA model is the total of a performer's current knowledge, especially that about the particular computer-mediated activity in which the user engages. It is a generally accepted idea that knowledge is not stored in long-term memory as disparate indexed facts, but in non-tangible constructs relating the various 'pieces' of knowledge to one another. There is less agreement as to the nature of these "higher-order, generic, cognitive structures that underlie all aspects of human knowledge and skill" (Reason, 1990, p. 26). Many different constructs (e.g. scripts, threads, propositions, schemata, networks) have been proposed, each with their attendant features, presuppositions and applications. One that is specifically thought to support reasoning about complex systems is the *mental model*.

There is more than one theory of mental models, and no general consensus on what the term means exactly (Al-Diban, 2012; see also O'Malley & Draper, 1992, p.73; and Payne, 1992, p.109-111)—so much so that the tongue-in-cheek phrase 'mental muddles' has been deployed (Rips, 1986). A core argument can, however, be distilled. This runs as follows.

A mental model is a simplified abstraction in the mind that is used not to describe but to understand and to reason about relevant aspects of a (natural or man-made) complex system (Seel, 2006). It evolves naturally (Payne, 1991) and its construction takes place continuously during interaction with the referent (the target system). Mental models are highly individual. Development of the mental model is strongly influenced by individual factors as diverse as past experience, external information sources, interaction with the target system, the current state of the mental model, and original thinking (Payne, 1991). As a result, "a mental model is not one thing, but a combination of several interacting component representational structures" (Bibby, 1992). Mental models have been observed to be both parsimonious (in that they contain as little detail as the user can 'get away with') and sticky (in that they are maintained for as long as the user can 'get away with it') (Norman, 1987). Interestingly, Norman's observed parsimony runs parallel to the production bias component of the Paradox of the Active User, while what he labels the stickiness of mental models reflects the assimilation bias. Norman also notes that mental models are typically incomplete, unstable, vaguely defined, unscientific and not at all the "precise, elegant models" described in the literature; rather, they can "include knowledge or beliefs that are thought to be of doubtful validity" and can sometimes even be characterized as "superstition".

Many authors (Craik, 1943; de Kleer & Brown, 1981; Johnson-Laird, 1983; Kahneman & Tversky, 1981) hold that in order to predict what the target system will do under certain conditions, those conditions are mentally applied to the

model which is then ‘run’, after which the envisaged outcome is (still mentally) observed. The first author to propose the term ‘mental model’ wrote:

“My hypothesis is that thought models, or parallels, reality... If the organism carries a “small-scale model” of external reality and its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge about the past events in dealing with the past and future...” (Craig, 1943)

Others maintain that to postulate the possibility or even the necessity of a trial run is to take the analogy with physical small-scale models one step too far, and that what distinguishes mental models from other knowledge structures is simply their being representational in one way or another of an underlying reality (Rips, 1986).

At first sight there is much similarity between a mental model, especially a runnable one, and the situation model that we have seen earlier. However, there are crucial differences. A mental model 1) mirrors a well-delineated target system that has an independent existence whereas the situation model consists of disparate elements brought together only by the perception of the individual. Then, a mental model 2) remains available in the knowledge base for subsequent use whereas the situation model is discarded immediately after its predominant interaction has been carried out. As a consequence, a mental model 3) evolves over time whereas the situation model is re-constructed from scratch every time. Finally, a mental model 4) is part of the knowledge that is input to the situation processing, whereas the situation model is output of the situation processing.

For a wide-ranging and multi-disciplinary overview of mental model theories and their various applications, see (Rogers, Rutherford, & Bibby, 1992). An older seminal work is (Gentner & Stevens, 1983). Finally, (Schwamb, 1990), although unpublished, provides a thorough overview of mental models theories. What they all have in common is their tremendous intuitive appeal. Mental models just ‘sound right’. But are mental models really ‘runnable’ or does the intuitive correctness refer to a more holistic idea of ‘some sort of internal representation of an external system’? Fundamental as this question is, from a pragmatic point of view it does not really matter whether the mental model is runnable or no more than representational. In both cases, an adequate mental model of the software tool being used will help the user’s performance (Bayman & Mayer, 1988; Bibby, 1992; Santhanam & Sein, 1994).

Definition: A *mental model* is a simplified representation in the mind of a complex system, that remains available in the knowledge base and evolves over time.

To describe what makes a mental model of a software tool ‘adequate’, we can imagine a ‘User Virtual Machine’ or UVM, which is defined as “not only everything that a user can perceive or experience (as far as it has a meaning), but also aspects

of internal structure and processes as far as the user should be aware of them” (van der Veer & van Vliet, 2001). The visible part of the UVM is what these authors refer to as the ‘perceptual interface’ and what is more commonly referred to as the user interface; but the UVM as a whole is a much larger conceptual mechanism. A software system, as we have discussed before, is a self-contained ‘world’ with its own objects (think of the Clipboard in many operating systems; of templates, style sheets and fields in a word processing environment; or of layers in an image editor). When visible to the user, such software-specific objects, their mutual dependencies, and the rules governing their behaviour are as much part of the UVM as is the user interface through which they are accessed.

Definition: The *User Virtual Machine* or *UVM* is a conceptual mechanism, containing all meaningful elements of a particular software system that the user can perceive or experience, as well as those that are not directly visible but influence others which are.

The UVM is man-made and finite, and therefore in principle knowable. Although conceptual rather than tangible, the UVM can be said to ‘exist’ in the same way that the laws of nature exist. Adequacy of the user’s mental model now is the degree to which it is correct and complete with reference to the UVM.

‘Correct’ means that the user’s mental model indeed matches the UVM. ‘Complete’ means that it contains all the different types of knowledge that are required for masterful performance. Knowledge can be categorized as to its nature. In the literature the notion of procedural (‘knowing how’) versus declarative or conceptual (‘knowing that’) knowledge plays an important role (see Marshall, 1995; Ohlsson, 2005; Ummelen, 1994). Sometimes, a third knowledge type is distinguished: conditional knowledge or ‘knowing when’. One problem with such a simple distinction is its lack of specificity. The terms cover a multitude of sins. In fact, they are collective terms for a number of subtypes (Ummelen, 1994). Various authors have attempted to establish more fine-grained classification schemes and indeed many have been proposed, all with their associated terminology (see P. A. Alexander, Schallert, & Hare, 1991). One such scheme (T. de Jong & Ferguson-Hessler, 1996, pp. 106-107) is of particular interest, as it offers a matrix in which four types of knowledge are set off against a number of qualities⁵. Thus it has the virtue of being fine-grained enough, in the qualities, to do justice to the many nuances of knowledge encountered in the reality of human philosophy, while still meeting the demands of Occam’s razor through the limited number of types. De Jong and Ferguson distinguish the following knowledge types:

⁵ deep versus surface; structure; automated versus nonautomatic; modality; general versus domain-specific

- *Situational knowledge* is knowledge about situations as they typically appear in a particular domain.
- *Conceptual knowledge* is static knowledge about facts, concepts, and principles that apply within a particular domain.
- *Procedural knowledge* contains actions or manipulations that are valid within a particular domain.
- *Strategic knowledge* helps organise the problem-solving process by directing which stages should be gone through to reach a solution.

All these types of knowledge are required for masterful performance (van Loggem, 2007) and therefore, for the user's mental model of the UVM to be complete, it must contain all four types. Situational knowledge is required for apposite perception and recognition. Conceptual and procedural knowledge are required, together, for constructing a valid situation model. Strategic knowledge, finally, is required for formulating an appropriate ambition.

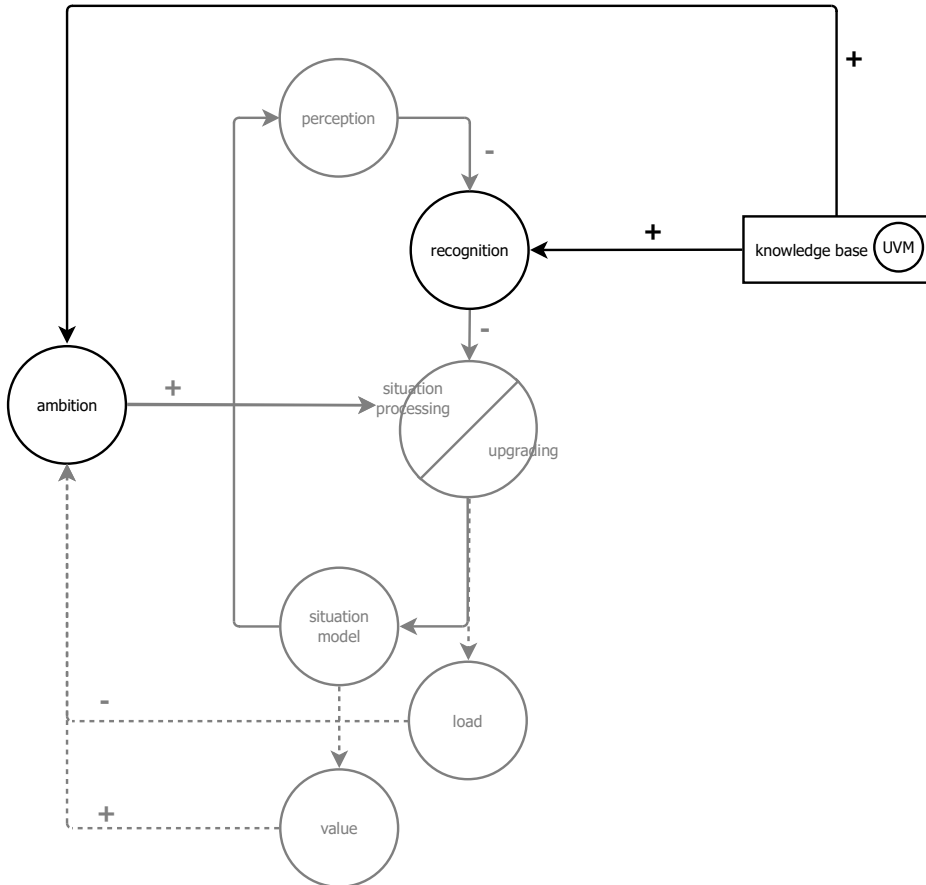


Figure 15: A mental model of the UVM as part of the knowledge base

To summarize: The knowledge base contains a mental model of the User Virtual Machine, which is constructed over time as the user interacts with the software. Eventually, the user's mental model of the UVM should be correct (that is, match the UVM) and complete (that is, contain all four types of knowledge required for masterful performance).

Uncertainty and Ill-Definedness

The model developed thus far, in the previous sections, describes the prolonged use of a reasonably complex tool, where 1) a potentially large number of decisions

as to intermediate steps must be taken and their outcomes evaluated before the initial intention for using the tool is fulfilled; 2) the intention itself can become more or less ambitious both during the activity and from one instance of an activity to the next; and 3) information on the tool and its use may be sought from sources external to the activity. Although many non-software, physical, tools fail to meet these criteria, there is nothing in the model that is necessarily software-specific. This final section refines the model to specifically describe the development of mastery when repeatedly using software-based tools.

Physical tools act directly on the material world. Operating a hammer causes (hopefully) a nail to be driven into a piece of wood. Operating a table loom causes strands of textile to be interwoven as warp and weft. Operating a typewriter causes ink to be rubbed onto paper in character-shaped patches. When operating a physical tool, the result of every operation is a state of the outside world that is—leaving fundamental philosophical argument out of the discussion—in principle open to inspection, either directly or through the intermediary of instruments. What you see is what you get: a change in an object that can be touched, heard, smelled, and/or viewed from different angles. When using a physical tool, the evidence before one's very eyes is indisputable. Incorrect previously held beliefs as well as errors having been committed show up immediately so that they can be corrected. Activities can be undertaken step by step and a user can in principle determine whether things have been done right. Also, the intermediate results as well as the end result of working with a physical tool would have been fully imaginable even before the tool came into existence, and a user can in principle determine whether he is doing the right things.

Software-driven tools, on the other hand, act at least in the first instance only on the software environment itself. Operating a PC running a word processing application causes a re-configuration of the software's definition of how a document is constructed; not directly related to character-shaped ink stains on paper. When operating a software-driven tool, the result of every operation is a state of an autonomous, self-contained 'world' with its own concepts, rules and dependencies which is never open to inspection. What you see in this case is usually no more than a change in the text and images on a relatively small screen that has no meaning in itself but needs interpretation. What you get is a change in the arrangement of bits in the computer's memory that may or may not have been defined by the creator of the software to cause a change to the outside world. Thus, when interacting with a software-driven tool, the moment when success or failure becomes apparent is delayed, sometimes indefinitely; and it is in principle not possible for a user to determine whether things have been done right.

For these reasons, there is always a degree of *uncertainty* as to the results of one's actions in the interaction with the present generation of software. In CMA, perception is by definition incomplete. The previous interaction has changed the arrangement of the computer's memory, which cannot be directly inspected—and

even if it could, inspection would not yield any information that is comprehensible to the user. The cues from the environment consist of no more than a change in any output devices (very often a visual display). Almost inevitably the number of possible arrangements of the output device is smaller than the number of possible arrangements of the internal computer memory. One combination of perceived cues matches multiple, possibly very different situations 'inside' the computer. As a consequence, recognition is impaired. A situation may be wrongly recognised, causing errors may go unnoticed and no upgrading taking place to correct them. Figure 16 shows this uncertainty by adding a 'crossed roads' symbol to the recognition.

Furthermore, neither the intermediate results nor the end result of working with a software-driven tool would have been fully imaginable before the tool came into existence. The software 'world' on which the user operates will always have concepts that do not directly correspond to anything that the user is a priori familiar with and he may not even know whether he's doing the right things. The desired outcome of the interaction might not be fully known to a user who does not yet know the ins and outs of the software tool he is using. In this respect, the interaction with software tools has a degree of *ill-definedness* built in that is not present when interacting with a physical tool. Pressing the space bar on a typewriter moves the carriage and leaves a particular location on a tangible sheet of paper as it was, creating a blank space. But pressing the space bar when using a word processor does just as much a pressing any other key: it inserts a character (albeit an invisible one) into a non-tangible document. Understanding the difference, unimportant as it may seem at a first glance, can well be vital to the quality of the final result of the work.

In the CMA model, a user who does not understand a particular concept, or even does not know of its existence, cannot formulate an ambition in which that concept is gainfully utilised. Figure 16 shows this by adding a 'crossed roads' symbol to the ambition.

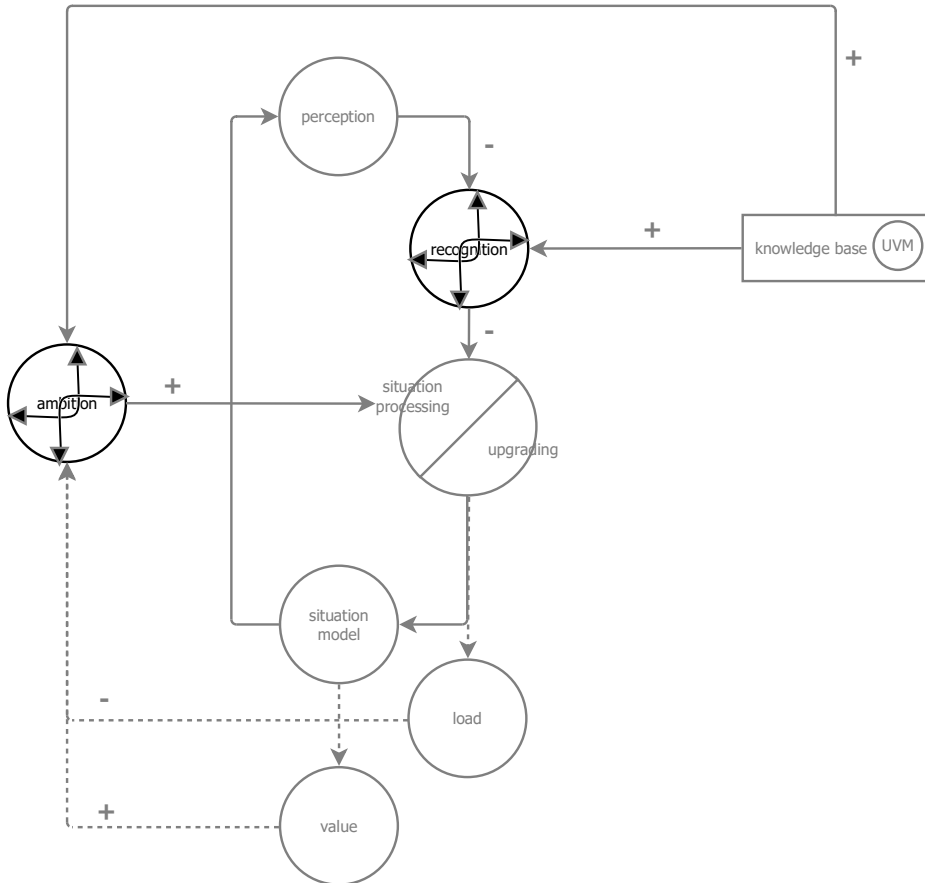


Figure 16: Uncertainty and ill-definedness in CMA

To summarize: The uncertainty built into computer-mediated activity compromises recognition. The ill-definedness built into computer-mediated activity compromises the ambition.

Difficulty in performing a task is proportional to its uncertainty and the degree to which it is ill-defined (Byström & Järvelin, 1995; D. J. Campbell, 1988). If uncertainty and ill-definedness are fundamental aspects of the use of software, then this will always be a 'difficult' activity and the Paradox of the Active User will not disappear of its own accord.

The Documentation Journey in CMA

With repeated performance involving upgrading, the knowledge base expands over time. Paradoxically, however, knowledge is not just a consequence of upgrading, but equally well a prerequisite for upgrading. Without rules no RB performance is possible. Nor can KB performance take place without first principles to be combined and elaborated. When the required knowledge to upgrade during situation processing is not available in the knowledge base, a performer may act more or less randomly, hoping for things to sort themselves out. The other option is for him to consult external information sources, e.g. by asking another person, by referring to the Internet, or by consulting a documentation artefact.

Until mastery has been achieved, external sources of information will be needed to enable adequate refinement of the current situation model. When no sufficient knowledge is (yet) available in the knowledge base, external information sources kick-start the initial situation processing and turn explorative, non-directed performance that is carried out really for lack of something better to do into directed performance. When there is already a certain amount of knowledge present, the information sources can promote upgrading and enable further situation processing until eventually, the additional knowledge allows for downgrading again and learning stops.

The documentation designer's task is to design documentation artefacts in accordance with the requirements for a particular design case. The CMA model describes the problem space in which documentation designers move. The context in which the Paradox of the Active User manifests itself is broken up into distinct cognitive constructs (perception, recognition, situation processing, upgrading, documentation journey, situation model, value, load, intention and goal) that interact with each other in a closed loop. Each of the constructs embodies a possible locus of breakdown in the process of learning from practice through repeated CMA; and each offers a target for interventions in the form of elements in the documentation that is designed.

As we have seen in Chapter 3, roughly speaking, the same stageposts on this 'information journey' are identified by different authors, albeit under sometimes very different names. Disregarding findings and frameworks that are not applicable to the area of software documentation, consensus is found on a cyclic pattern in what can now be called the *documentation journey* of 1) needing; 2) seeking; 3) filtering and 4) applying information. In the CMA model:

- *needing* information (N), that is, realizing and accepting that the knowledge base is not sufficient to meet the current ambition, follows after a lack of recognition, when situation processing begins;

- *seeking* information (S), that is, attempting to fill the gap between that which is known and that which is as yet unknown, is part of the upgrading component of situation processing;
- *filtering* information (F), that is, sorting out the information that is found to match the current need, is part of the situation processing resulting in the situation model;
- *applying* information (A), that is, acting on the information retained and understood after filtering, is part of the final situation model that is decided upon, which contains the interaction to be taken.

Definition: The *documentation journey* is the sequence of stages that a performer (mostly unconscious) completes when interacting with documentation.

The different aspects of the various stageposts in the documentation journey are all places where documentation can ‘do something’. Since documentation will be used more than once, an author can create his documentation designs to influence (future) behaviour at every one of the stageposts; not just the one currently being visited.

However, this process is seriously compromised by the inevitable presence of uncertainty and ill-definedness, leading to an inappropriate information need. Another problem is that the documentation journey itself will constitute load. Finally, the author of an information product cannot force a reader to handle the product in any particular way. What he can do, however, is steer the reader’s attention in a particular direction, strengthening or weakening the ease with which a reader can access different units of information from a particular location within the product.

In Figure 17 below, information sources are included in the CMA model, overlaid with a ‘crossed roads’ symbol to emphasize the fact that their influence is dependent on a large number of factors that lie outside the model.

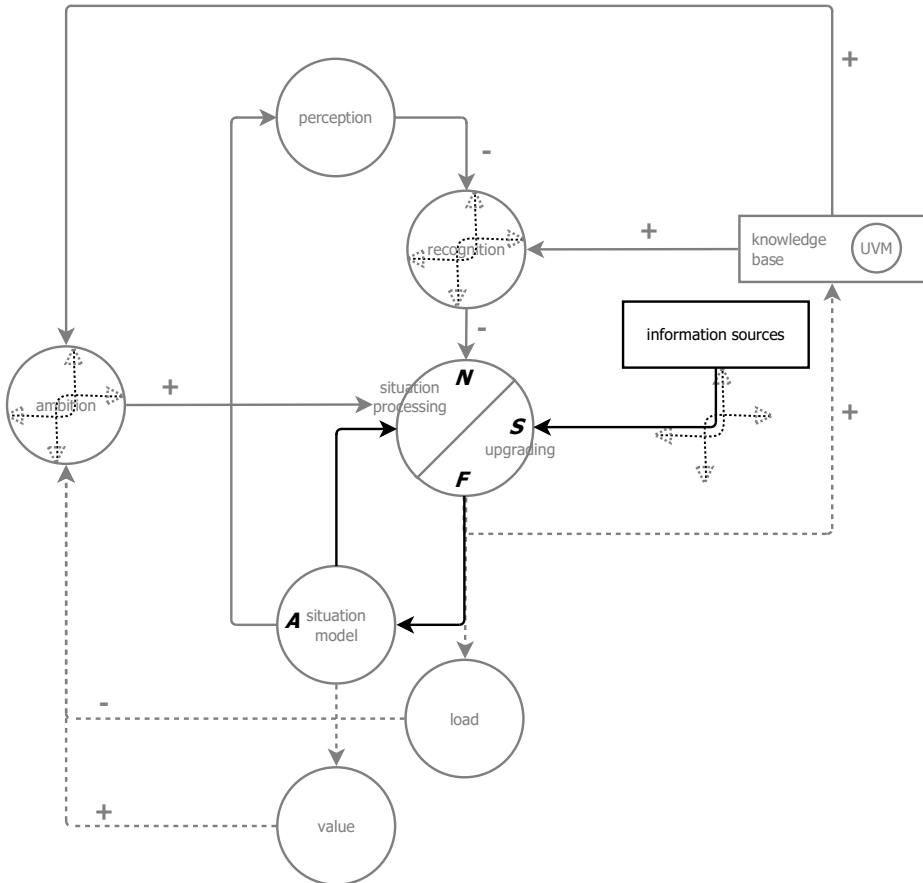


Figure 17: The documentation journey during CMA

To summarize: If the information required for upgrading is not already present in the knowledge base, it must be externally available, appropriately accessed, and correctly applied for situation processing to result in an adequate situation model.

CMA as a Knowledge Engine

The CMA model developed in this chapter is repeated in full in Figure 18. It is a model of unsupervised practice in which upgrading transforms information into value, with knowledge as a by-product; ideally the process continues until mastery of the software has been achieved.

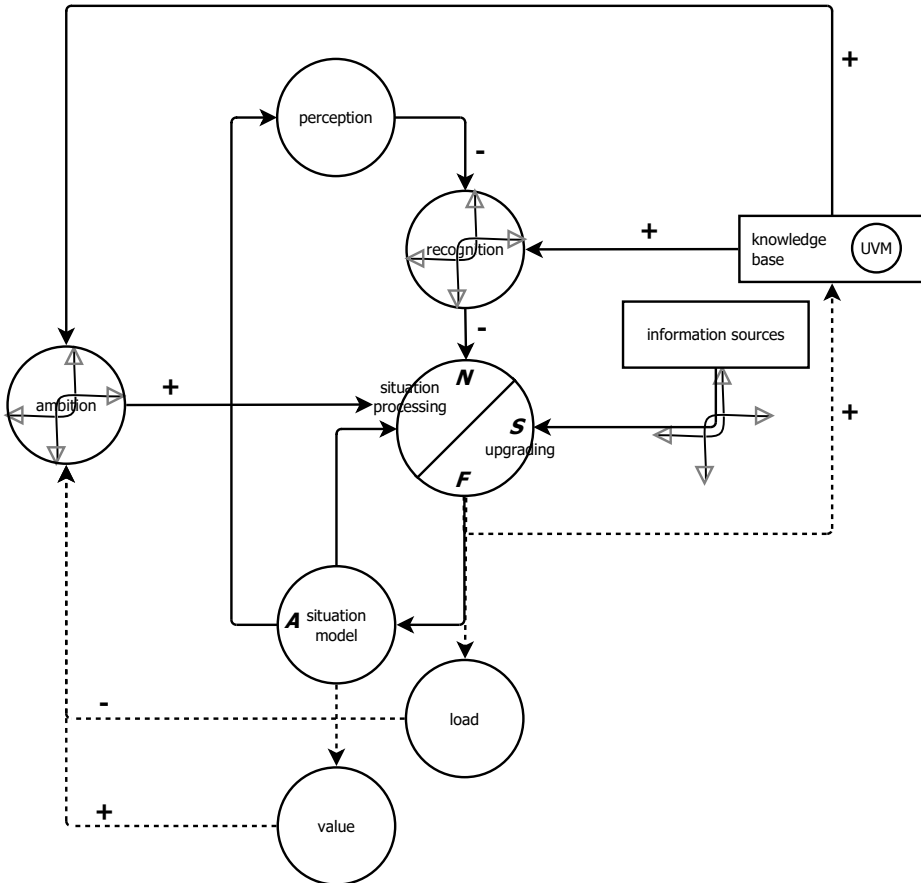


Figure 18: The complete CMA model

The process can be summarized using a metaphor. Imagine upgrading as an engine, fuelled by goal-setting. As long as the engine runs, information is fed into it and processed and as a result, the knowledge base expands. As the knowledge base expands, it will offer increasing resistance against further expansion so that the process slows down. Ideally the engine continues running until the knowledge base is full and mastery has been achieved.

Such idealized development is illustrated in the plot below (Figure 19, where the X axis represents time). As long as nothing untoward happens a novice starting out with mastery close to zero and an ambition that corresponds to 100% of the software's capabilities, will eventually—provided he keeps putting in all the cognitive effort that is needed—end up with mastery close to 100%. Practice will have made perfect.

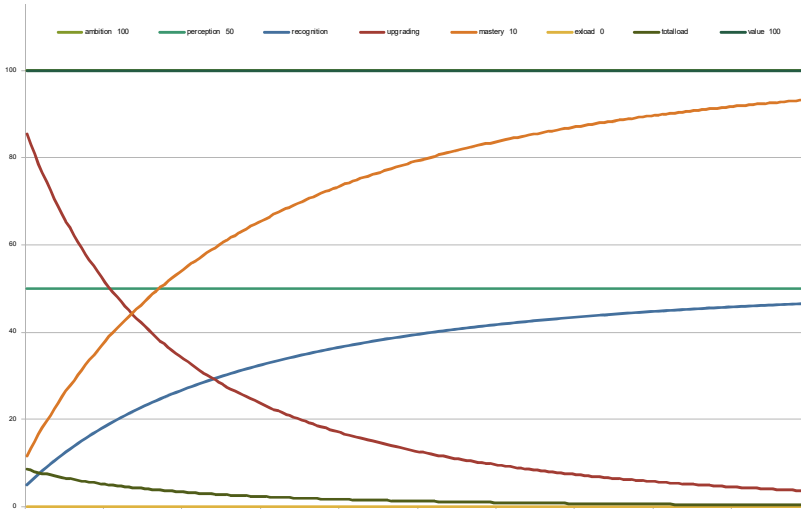


Figure 19: The achievement of mastery with practice over time, in an ideal situation

However, the process cannot be expected to always run quite so smoothly. First of all, it is highly unlikely that even the most driven individual would always be willing and able to put in all the effort required to bridge the gap between a current mastery level of close to zero percent and an ambition of close to one hundred percent.

Then, the CMA model shows two inputs to upgrading. The first is the ambition. The need for upgrading arises from the difference between that which a performer wants to do and that which he is currently capable of doing. When the gap between an performer's current mastery and his ambition is narrow, the performer never learns to perform far beyond his current competence level. When the gap is too wide, he may simply opt out and give up. At the very least he will satisfy. The ambition itself is driven by the experienced value-load balance. If the perceived value of the end result may not be what he had been hoping for, the next time he encounters a similar context, he may approach it with a (much) lower ambition. When the value does not at least outbalance the load, the fuel supply will dwindle so that the engine runs down, leaving the software user stranded at a suboptimal level of performance. Rather than the competence moving up in the direction of the ambition, the ambition moves down to match the competence and full mastery will then never be achieved (see Figure 20).

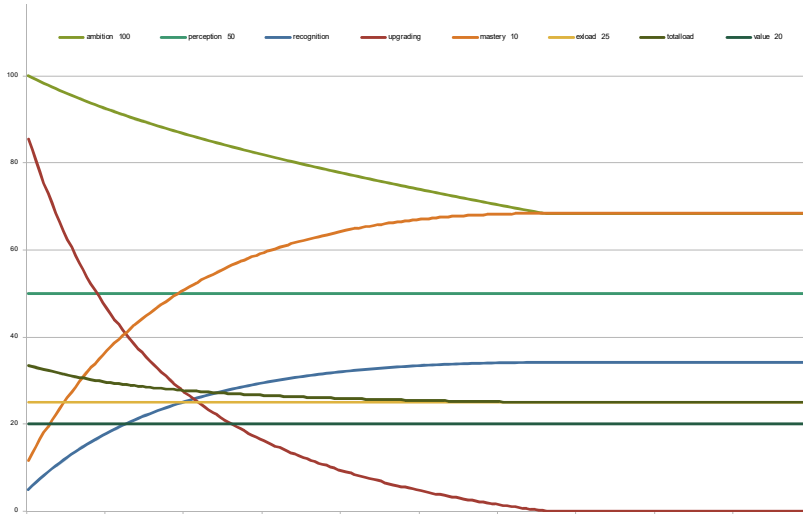


Figure 20: The effect of load exceeding value

The more generally an ambition is framed, the less appropriate will its constituent goals be set and the less chance there is that the interactions will be appropriately selected or their outcomes correctly assessed. Only when the ambition is rich, i.e., takes into consideration all concepts held in the software universe, will it be able to drive gainful situation processing. For continuous learning, the ambition should be always a little beyond the performer's 'comfort zone', in his 'zone of proximal development' but not beyond (L. Vygotsky, 1978). Therefore, a good design of documentation again exposes the full richness of the task domain and does not overly simplify.

The other, negative, input to upgrading is recognition. In the words of James Reason: "human beings are furious pattern-matchers" (Reason, 1990). Considering also the fact that the *if* clause in the knowledge patterns used in RB performance only contains features and characteristics that the current situation model *should* match and not those that it should *not* match, we begin to see that "there will be powerful cognitive and affective forces conspiring to encourage the problem solver to accept inadequate or incomplete solutions as being satisfactory at this point" (pp. 66-67). Recognition in its turn is inversely related to the degree to which relevant aspects of the environment are perceived: the more you consciously see, the less chance there is that you unjustifiably recognize it.

In Figure 21, the effect is shown of inappropriate recognition.

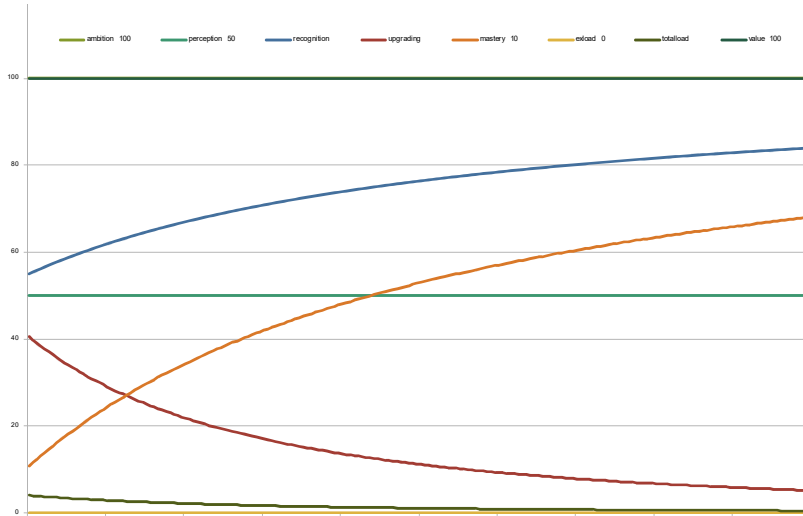


Figure 21: The effect of inappropriate recognition

Finally, the two inputs to upgrading reinforce each other. Lowering the ambition to match results is self-reinforcing. When it happens, the performer is left with a sense of the task having been difficult. That in itself is enough to lower the ambition further during future practice. In addition, as a situation continues in which no more upgrading takes place, existing knowledge is re-applied over and over again. Recognition thus grows stronger and it becomes less likely that upgrading and learning are resumed. As Glaser notes: “People may attain a level of competence only insofar as it is necessary to carry out the activities or to solve problems at the given level of complexity presented. Situations that extend competence may be less forthcoming as experts settle into their working situations” (Glaser, 1985, p. 7). When value becomes limited and/or load is seen as unacceptable, a downward spiral rapidly causes no further upgrading to take place.

Thus, the CMA model offers one possible explanation of why the two ‘biases’ underlying the Paradox of the Active User manifest themselves so much more strongly for computer-mediated activity than they do for other tool-mediated activity.

- The *production bias* hinders the balancing of (expected and perceived) load against value. It leads to the performer satisficing beyond that which would be seen as ‘good enough’ if there was no tendency to press on, and interrupts the gradual positioning of the ambition at an ever higher level: even when the performer is aware of the potential future benefits of doing so. The production bias also hinders the application of knowledge to the ambition: that which is

deemed 'too much work right now' will be ignored. The production bias throws a spoke in the engine's wheels exactly there where its smooth running is already impaired by ill-definedness.

- The *assimilation bias* stops the performer from learning both directly and indirectly. Directly, because it hinders the inclusion of new knowledge resulting from situation processing into the knowledge base. Indirectly, because it makes the performer inappropriately eager to recognize the currently perceived cues from the environment where really there are reasons to not recognize and act immediately, but rather look again and upgrade. The assimilation bias puts a spoke in the engine's wheels in both places where its smooth running is already impaired: once where ill-definedness is located, and again where uncertainty is present.

Documentation should therefore strive to minimize unproductive load, and to maximize the chances of a genuinely valuable outcome. They should also aim to direct attention to as many of the external cues as possible, and slow down recognition further by exposing the full richness of the situation, including elements in the current situation model that do not match the recognized situation.

5. *Documentation As Artefact*

A first version of the section *Assessing Use Complexity* in this chapter has been presented in a short conference paper at the 4th International Conference on Human-Centred Software Engineering (HCSE 2012), held on 29-32 October 2012 in Toulouse, France (van Loggem, 2012).

Abstract: In this chapter, we move away from a description of the situation ‘as is’, before any interventions are designed, to the goal of any such interventions. The twin concepts of expertise and mastery are explored. As not all software systems are equally difficult to master, rather than using generic denominations such as ‘simple’ and ‘complex’ tools, a more rigid three-tier categorization of software is given, based on ‘use complexity’. This is then shown to provide a map of the requirements for the software’s documentation, expressed in and following from the CMA model.

Expertise and Mastery

‘Mastery’ is expertise in a particular, applied domain. From the existing literature on expertise (regardless of the domain in which the expertise is acquired) a characteristic pattern emerges (Cellier, Eyrolle, & Mariné, 1997; Farrington-Darby & Wilson, 2006; Glaser, 1985; Shanteau, 1992). Experts have shown consistently to (1) achieve different results compared to novices. They come to qualitatively better results and solve problems overall faster. In addition, experts are seen to (2) build different representations. They have more complete representations of the task domain containing a wider range of variables that are more highly interlinked, in terms of contextual data as well as data from the knowledge base. Although just like non-experts they can hold no more than a handful of units or ‘chunks’ of knowledge in working memory at any given time, these have been aggregated over time from smaller, disparate units until they contain (much) more detail and (many) more different aspects: so that the same number of ‘chunks’ in working memory represents considerably more knowledge (de Groot, 1946). They focus on deep relational features of the situation whereas novices focus on surface features (Sloutsky & Yaras, 2000). When discerning patterns, an expert is better capable of ignoring noise, possessing a sense of what is relevant when making decisions. Then, it can be said that experts (3) employ different strategies. They devote more time to initial problem encoding and building a representation of the situation. They simplify complex problems, break problems down, and develop a divide-and-conquer strategy; and they make better predictions of process evolution and changes in a system, seeing the implications of their actions. Experts pay more attention, seek further information (whereas novices focus on directly available data) and extract more information. They encode new information more quickly and effectively and grasp the meaning behind information. They process cues

proactively rather than reactively and are less easily thrown by adversity and exceptions. Finally, experts (4) have a different knowledge base: they have a larger number of situational patterns available, have extensive content knowledge, and have a wider range of options at their disposal. This enables an expert to display self-confidence and to justify his decisions. Experts rely more on automated processes (Shanteau, 1992). Their knowledge has become highly procedural and their memory search and general processing are greatly reduced (Glaser, 1985). In layman's terms: experts do not have to labour to come up with the next interaction. They just go ahead and recognize what needs doing. It is only during the learning process that existing knowledge structures prove insufficient.

Two of Jens Rasmussen's colleagues at the Risø National Laboratory in Denmark examined expertise against the backdrop of the three SRK levels of performance. Their conclusion was that, with the acquisition of expertise over time, the knowledge available to a performer changes (Sanderson & Harwood, 1988, p. 31). People, Sanderson and Harwood state with reference to the table reproduced below as Table 1, "will enter training with a repertoire of general rules-based and skill-based behaviours (bracketed terms). By the expert stage, the SRK levels are qualitatively different from when they were being learned, and this is indicated by the numerals. For example, expert knowledge-based behaviour at K-B3 level will operate differently from the way it does for the novice at K-B1. The knowledge base itself may be considerably refined and enriched. More importantly, the knowledge will be used in a very different way."

Table 1: The development of expertise in the SRK framework

novice	intermediate	expert
KB1	KB2	KB3
(RB)	RB2	RB3
(SB)	(SB)	SB3

Expertise, it turns out, is not evidenced by all performance being carried out at SB level. Rather, experts carry out as much performance at SB level as is possible in the context of the current activity. And this is exactly what a complete and correct model of the UVM will allow software users to do.

It has been observed that when asked about their current activity ("What are you doing?"), performers may respond with different degrees of abstraction (Vallacher & Wegner, 1987; Wegner, Vallacher, Macomber, Wood, & Arps, 1984). Dependent on the particular context the answer may be, for example, "seeing if someone is at

home” or “forecasting next year’s budget”; “pushing a doorbell” or “building a spreadsheet”; “moving a finger” or “entering a formula into a cell”. The degree of abstraction on which identification takes place is mediated by the context, the difficulty, and the familiarity of the task. In general, Vallacher and Wegner observed, performers try to maintain the highest possible degree of abstraction; until the identification cannot be maintained and drops down one step. When discussing expertise in the use of a particular tool this means that, as expertise increases and ever more performance takes place at ever more effortless levels of performance, the activity will be gradually positioned at an ever higher degree of abstraction: until the full extent is taken into account of what the tool is capable of. At this stage, the intention horizon (p. 23) includes all of the UVM (p. 56), and *mastery* of the tool has been achieved.

Definition: *Mastery* is expertise in a particular domain of tool-mediated activity, enabled by the ability to instantiate, for any actual need, a complete and correct mental model of those aspects of the User Virtual Machine that are actually relevant. Mastery is evidenced by successfully completed activities at the highest possible level of abstraction that the tool allows for.

A naturalistic description of interaction with software cannot discuss the interaction and the software in isolation but must consider software-in-use, as a man-machine system in which two partners—the human and the machine (driven and defined by software)—work together (Benyon, 1992; Mirel, 1998a; Steehouder, 1997). Having thus far mostly explored the perspective of the human partner in the man-machine system, we will now shift to that of the software-based partner; and align the two perspectives to arrive at a complete analysis of software-in-use, with a view to documentation requirements.

Supporting the acquisition of mastery is what documentation ultimately aims to achieve. In order to defeat the Paradox of the Active User, the knowledge engine must not be allowed to run down prematurely, that is, before the knowledge base is full. What constitutes ‘full’, now, is dependent upon the nature of a particular software tool and the context in which it may be deployed.

Assessing Use Complexity

It seems obvious that more learning is required when the task to be learned is more ‘complex’. A user needs a thorough understanding of the User Virtual Machine or UVM to gain complete mastery of a particular software tool, and apply it successfully to every task it can possibly be applied to. To determine the extent of the UVM, I propose a series of analyses to assess *use complexity*: software complexity from the user’s point of view.

Use complexity is closely related to what Kieras and Polson (D. Kieras & Polson, 1985) call ‘cognitive complexity’ or ‘user complexity’. I have chosen not to use either term used by these authors, for a number of reasons. First, (1) ‘cognitive complexity’ is regularly, by authors other than Kieras and Polson and in areas other than software development, taken to describe a personality characteristic. A particular individual may exhibit high or low cognitive complexity just as he may exhibit, for example, high or low creativity or self-confidence (e.g. Adams-Webber, 2001; Manojlović & Nikolić-Popović, 2002; Quinn, 1980). Then (2), ‘user complexity’ linguistically suggests a characteristic of the user rather than the tool used. Finally (3), Kieras and Polson’s concept of cognitive/user complexity presupposes ‘yoking’; which, as we have seen (p. 21) cannot in itself account for the problems encountered when working with much of present-day software.

Use complexity is a characteristic of the software only, irrespective of any characteristics inherent to the user or the task environment. It provides a measure of what the UVM consists of rather than how a user interacts with it. As such it is separate from the concept of usability, which is defined in the ISO 9241-11 standard as *the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use*. Usability focuses on the user interaction component of the UVM and is prescriptive, in that high usability is desirable; whereas use complexity is descriptive, in that high use complexity is not necessarily undesirable. Use complexity is a necessary consequence of versatility (Norman, 1999, 2010). A complex system may, but need not, have low usability: just as a simple system may, but need not, have high usability.

Neither is use complexity directly related to design complexity: that which lies ‘under the hood’ of the software. A program is complex in the design sense of the word if it exhibits emergent behaviour (Aiguier, Le Gall, & Mabrouki, 2008); but sophisticated as it may be, to know whether it has use complexity a separate assessment must be undertaken. Use complexity is no more or no less than a measure of the learning requirements (and by extension the documentation requirements) for a particular piece of software.

Definition: *Use complexity* is that part of task complexity that originates from the software rather than from other elements in the task environment, including the user.

Complexity is a multi-dimensional construct (S. Kim & Soergel, 2005; Nadolski, Kirschner, van Merriënboer, & Wöretshofer, 2005). The quality and quantity of the learning required to achieve full mastery of a particular software tool can be analyzed from a number of different viewpoints. Although absolute values for the different dimensions cannot feasibly be determined, a relative scale can be envisaged ranging from, say, ‘almost none’ to ‘huge amounts’. It then becomes

possible to take the idea of dimensions literally and map out the results of the separate analyses.

Width: Quantity

The first dimension of use complexity to consider is its 'width'. This is involved with how much novelty is brought to the work by the software, and therefore with how much is to be learned.

Novelty comes in the form of hitherto unknown concepts, with their associated rules and interdependencies, that are exposed to the user and with which he may choose to engage. A first step towards assessing where there is width in the use complexity of a particular tool is to split its application up into a number of levels (usually, three or four). Several theoreticians have independently identified such a hierarchy at which to consider tools and tool use. David Marr (Marr, 1982), for example, identified the computational, algorithmic, and implementational levels on which a cognitive system should be accounted for. The computational level describes the system's goal, the reason why it is appropriate, and the logic of its strategy. The algorithmic level specifies the system's method and its representations for the input and the output. The implementational level, finally, describes the system's means and physical implementation.

Marr presented his insights in the context of human vision; closer to home and one year earlier, Thomas Moran (1981) distinguished four levels on which a user's conversation with a software tool can be described. The most general of these is the *task level*, which describes the things that a user can accomplish with the software. One level down, Moran identifies the *semantic level*, containing the conceptual entities representing the system's functional capability through which the user's tasks are accomplished. Then, Moran goes down one more to the *syntactic level*, where commands, arguments, contexts, and state variables are laid out. Finally, there is the *interaction level*, which describes physical interactions with the computer hardware such as "key presses and other primitive device manipulations." (Moran, 1981, p. 6)

In a context of standard computer use, at least for fully-grown users who are not physically restricted and who have a minimum of experience in computer use, the interaction level can always be assumed to be fully mastered. Situations in which this is not the case (involving very young children, or adults whose motor skills are impaired, or able-bodied individuals who are completely new to computers) are outside the scope of this current work: the interaction level seems best mastered through physical practice that does not involve information (van der Veer et al., 1985). In this discussion I will borrow Moran's ideas as far as the remaining three levels are considered, changing the labels slightly but maintaining the spirit in which the hierarchy was originally defined:

Definitions: The *task layer* is that aspect of a software tool that describes the possible end results of the user's engagement with the software. The *semantic layer* is that aspect of a software tool that describes the intermediate steps that the user may carry out to realize a particular result. The *syntactic layer* is that aspect of a software tool that describes a user's choice of commands with which he directs the software's behaviour.

At the task layer, the user chooses to work towards a certain end result. At the semantic layer, he chooses the next step to realize the result. At the syntactic layer, finally, he chooses which screen area to click or touch, which key or keys to press, which sequence of characters to enter, or which command to vocalize. Where the user is exposed to many novel concepts that he can choose to engage with, there is much to learn before full mastery is reached. Where at the opposite end of the spectrum there is no visible novelty, or every choice is equal to every other choice, there is hardly any requirement for learning at all.

This quantitative dimension, related to the number of meaningful choices open to the user that originate from the software world and that he is not familiar with, can be pictured as the *width* of the use complexity. The width of the use complexity shows how much procedural and situational knowledge a user needs to achieve mastery: whenever there is novelty to be engaged with, a user will need to know how to engage with it, and under which conditions a particular engagement is valid.

The use complexity is not necessarily equally wide at all three layers. The width of the use complexity can be assessed separately at every one of the three layers (syntactic-semantic-task).

Definition: The *width* of a software system's use complexity is a measure of the number of hitherto unknown concepts (with their associated rules and interdependencies), brought to the work by the software, that are exposed to the user and with which he may choose to engage so that his choice makes a difference.

When analysing the layers one by one, it is important to take all novel opportunities for choice into account, as there is no way of knowing which are important and which are not. Since the higher layers build upon and provide meaning to the lower ones, even misunderstandings at the syntactic layer can lead to grief. Concepts such as pressing the Enter key on the keyboard resulting in a paragraph being created, can—when not thoroughly understood—make creating even the simplest text document very difficult. As another example, the presence of the 'wildcards' concept at the semantic layer may lead to unexpected results for an uninitiated user attempting to search his text for the occurrence of an asterisk.

In Figure 22, the width of the use complexity is roughly plotted, using a relative scale of ‘almost no novelty’ to ‘huge amounts of novelty’ to engage with, for three different text processing applications that have all been marketed by Microsoft®. The base corresponds to the syntactic layer, then comes the semantic layer, and the top represents width of the use complexity at the task layer.

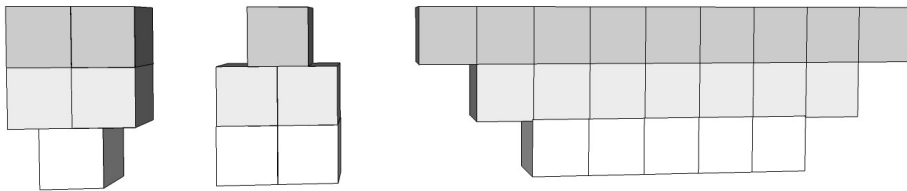


Figure 22: Width of use complexity sketched for Notepad (left), WordPad (centre) and Word2007 (right)

Notepad is judged to have very little novelty to engage with at the syntactic layer. Its interface is uncluttered and most of the options open to engagement are well known. At the semantic layer a bit more novelty is found, for example in Notepad's dealing with word wrapping and the use of variables in its page setup; while at the task layer its uses are acknowledged as a tool for holding and converting electronic text that was never intended to be printed. WordPad on the other hand holds some more novelty at the syntactic layer, mostly due to its formatting capabilities. At the task layer WordPad offers less, as it is very much geared towards one task: producing a document that is later to be printed exactly as laid out on the screen. For Word2007, much more novelty is found at all three layers, increasingly so as we move up towards the task layer.

Depth: Quality

The next dimension of use complexity is its ‘depth’. This is involved with the quality of things to be learned.

Where visible novelty in the software corresponds to pre-known ideas in the pre-existing task world, it allows for new ways of doing things. There is uncertainty but no ill-definedness, and the knowledge that needs to be acquired is mostly procedural and situational: how to do something new. This knowledge is relatively easily acquired. It is not open to discussion or interpretation and can be embodied in written instructions which, over time, are internalized. Yet where the software exposes its users to novelty that does not map directly onto pre-known ideas there is more of a problem. A user who is not even sure what it is that he should be doing in the first place, is unable to frame his engagements in such a manner as to use the software to its full potential. There are now new things to do, leading to ill-

definedness and a need for conceptual and strategic knowledge: the user needs to learn why and when he could decide to do something new.

This distinction constitutes a dimension of use complexity that can be visualized as its *depth*. The less tightly that the novelty brought to the task by the software is coupled to pre-known concepts that the user is already familiar with, the deeper the use complexity.

Definition: The *depth* of a software system's use complexity is a measure of the degree of mapping between on the one hand the novelty brought to the task by the software, and on the other pre-known concepts that the user is already familiar with.

The degree of mapping between software novelty and familiar concepts can most readily be determined by considering the outcome of the user's current engagement with the software (compare p. 22). When the outcome of the engagement lies within the software only, there is no direct mapping. This is the case, for example, at the task layer of software for creating websites and at the semantic layer of text editing software that allows for regular expressions in searching and replacing. In other cases, the outcome of the engagement lies partly outside and partly inside the software. The software then holds a model of something in the physical world or in the user's mind, and it is the model that is modified. The mapping is then much more straightforward. Think, for example, of a kitchen planner such as offered by many home furnishing stores, where the task layer is involved with constructing a model of your new kitchen; or a calculator program where at the semantic layer the various memory functions mimic a scratchpad holding intermediate results.

Figure 23 shows the sketches from Figure 22 enhanced with depth to indicate novelty for which there is no straightforward correspondence with the pre-known world.

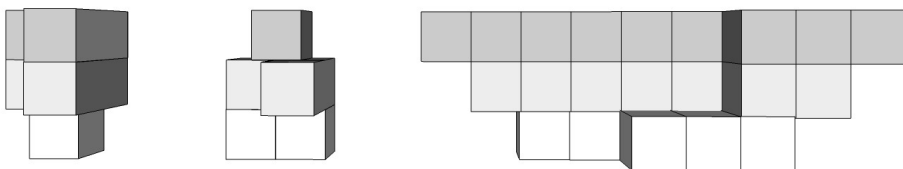


Figure 23: Depth of use complexity added to the analysis for Notepad (left), WordPad (centre) and Word2007 (right)

Approximately half of Notepad's novelty at the semantic and task layers is considered not to correspond to pre-known concepts; whereas WordPad is judged to expose the user to non-mapping novelty only at the semantic layer, and a

considerable proportion of Word2007's novelty at all layers has meaning within the software world only.

Height: Impact

The third and final dimension of use complexity is its 'height'. This is involved with the impact of things to be learned, and thus with the amount of effort that is ideally expended upon the learning. While width and depth of use complexity other must be assessed on every one of the three layers (syntactic-semantic-task), the layers themselves are the same for every software tool. If we position the three layers with reference to the user's ambition, then 'height' of use complexity becomes a variable dimension, which says something about the impact of the software on the user's work.

A coherent framework for studying the interaction with tools in the context of real-life human endeavour is found in the relatively recent dusting-off of activity theory, first developed in the Soviet Union in the early decades of the 20th century (L. S. Vygotsky, 1978) but now sparking widespread interest in the field of UID (see Bødker, 1991; Nardi, 1996). Human activity in this framework is regarded as a form of doing that is directed towards a material or immaterial object satisfying a need. Furthermore, all human activity is mediated by the context, social and otherwise, in which the activity is carried out and by the tools that are used (Bødker, 1991; Kaptelinin & Nardi, 2006). Deployment of a particular tool changes the quality of the activity, as does the knowledge currently present in the performer's knowledge base.

The performer consciously or unconsciously formulates an intention towards the object, based on what is felt to be attainable; and this is dependent on the current state of the knowledge base. The intention provides the motivation and is what distinguishes one *activity* from another. Activities are carried out through chains or networks of *actions* which find their minimal meaningful context in the activity and which each have an objective result meeting a particular goal. Actions in their turn are realised through a sequence of *operations*, which are habitual routines used as responses to conditions faced during the unfolding of the action.

Operations are observable behaviours, whereas actions and activities are mental constructs driving the behaviour. The activity provides the 'why' of something taking place, whereas the actions are 'what' takes place and the operations are 'how' the work is carried out (Bødker, 1997).

Definitions: An *activity* is a sequence of actions, undertaken one after the other in order to achieve an overall intention providing a motivation. An *action* is a sequence of operations, undertaken one after the other in order to achieve a particular goal within the wider framework of the activity's overall intention. An *operation* is a routinely carried-out observable behaviour.

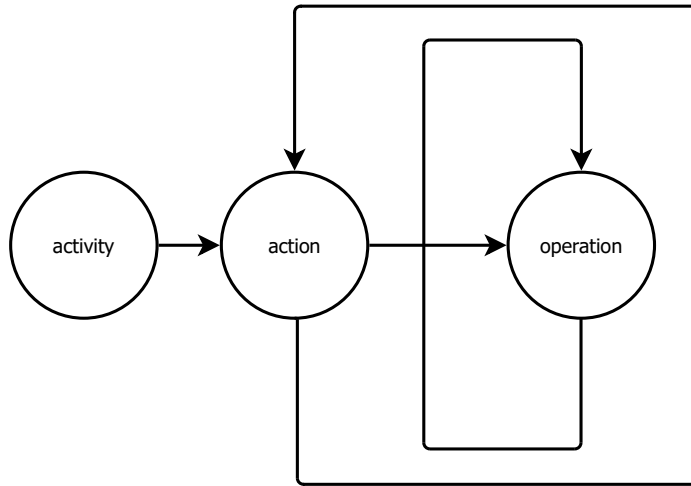


Figure 24: The relationships between an activity, its constituent actions and their constituent operations

Note that the combined long-term motivation and short-term goal that at any given moment provide the incentive to carry out an interaction, have in the CMA model been jointly referred to as the user's ambition.

An important activity-theoretical idea is that this three-tier hierarchy is not absolute, but relative within the context of the situation. Also, it is dynamic rather than static. When a 'breakdown' occurs, an operation that is no longer habitual by that token becomes an action; whereas an action that is practised many times and is thoroughly mastered is likely to become an operation. Likewise, the goal of an action can be raised to a higher plan so that the action becomes an activity, and an activity that loses its motive can become an action. Such shifts may happen more than once, perhaps even many times, before the performer ends the activity. Figure 25 (taken from Bødker & Bertelsen, 2003, p. 301) illustrates this idea, which ties in with Vallacher and Wegner's observations (see p. 72):

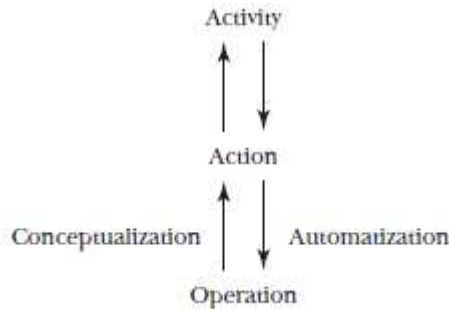


Figure 25: The dynamic relationship among levels of human activity (taken from Bødker & Bertelsen, 2003, p. 301)

The task layer in any given software tool does not necessarily correspond to the activity, nor the semantic layer to actions or the syntactic layer to operations. Task layer, semantic layer and syntactic layer are tool-centred, fixed, and objective. To determine what they consist of, it is sufficient to consider only the software. Activities, actions and operations on the other hand are human-centred, fluid, and subjective. They change over time and from one performer to another, and to determine what they are, it is sufficient to consider only the user. It cannot be stated a priori what constitutes an activity, an action, or an operation: at any given moment these levels are dependent on the current familiarity with and difficulty of the task.

Aligning the human-centred (activity-action-operation) and the tool-centred (task-semantic-syntactic) hierarchies exposes the relative roles of the two partners in a man-machine system. These are not always the same (Mirel, 1998a). The software component does not necessarily affect all the levels in an activity: situations are easily imaginable in which only part of the activity is computer-mediated. Software that affects the activity only insofar as it changes the nature of operations is less complex than that which touches upon actions, or even changes the nature of the activities that are possible. As the software's mediation reaches further 'up' into the activity, it adds more uncertainty to constructs that were not very well defined to begin with: the goals of actions and the intention of the activity.

Definition: The *height* of a software system's use complexity is a measure of the degree to which the software can affect an activity (in the activity-theoretical sense of the word).

The *height* of the use complexity is therefore determined by first asking whether the outcome of the task layer of the software could ever be imagined to constitute a valid object, providing an expert user's intention to sit down to work. Can the

end result of the work, if completed successfully and easily, ever be imagined to be something a person could look back on with satisfaction at the end of the day? Is it something he might, over breakfast, put on his mental to-do list for the day? If so, the software reaches up into the level of the activity. If not, its application reaches no further than the level of the separate actions.

The height of the use complexity modifies the 'grid' on which the width and depth are plotted, as shown in Figure 26. The degree to which they have an impact on the user's activity is reflected in the visualization by overlaying the three equally spaced interaction layers with the also equally spaced activity-theoretical levels, so that the top of the software's task layer aligns with either the level of the actions (left and centre in Figure 26) or that of the overall activity (right).

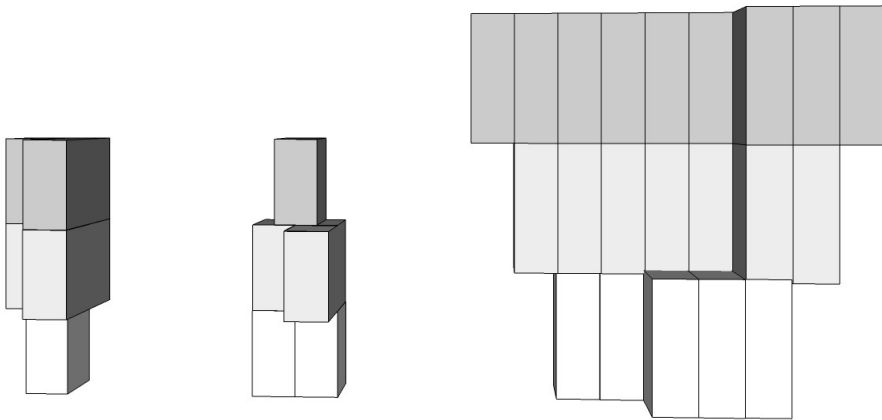


Figure 26: Height of use complexity added to the analysis for Notepad (left), WordPad (centre) and Word2007 (right)

In this manner, the use complexity of a software tool that affects the user's activities covers a larger area than that of software affecting only the separate actions. Effectively, the width and depth are multiplied with a factor that is greater as the software reaches further up into the activity.

The outcomes of Notepad's and WordPad's task layers are relatively modest, and in this example they are not seen to provide more than the goal for an action such as writing a short note or making a shopping list. The outcome of Word2007's task layer on the other hand could well be imagined to be of genuine interest in itself, such as the complete production and layout of a camera-ready manuscript, or the programming of a word processing environment for third parties to use.

Documentation Requirements for Operators, Actors and Activators

Figure 27 below is essentially the same as Figure 26, but for greater clarity the boundaries between operations, actions, and activities are included in the visualisation.

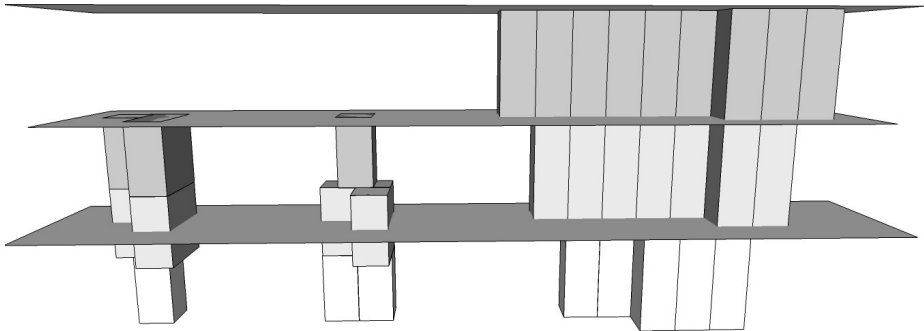


Figure 27: Aligning the human-centred (activity-action-operation) and the tool-centred (task-semantic-syntactic) hierarchies

We can immediately see how 'high' up there is depth in the use complexity. For Notepad (left), this is within the 'actions' band: this we can refer to as *actor software*. For WordPad (centre), the highest depth of use complexity is positioned around the 'operations' band and we can refer to this type of software as *operator software*. Word2007 (right) has considerable depth in the use complexity within the 'activities' band, so that it can be labelled *activator software*.

Definitions: *Operator software* is software in which there is depth in the use complexity only at the level of operations. *Actor software* is software in which there is depth in the use complexity at the level of actions, but not at that of activities. *Activator software* is software in which there is depth in the use complexity at the level of activities.

The use of discrete labels does not mean to imply that the three categories are discrete: there exists a gliding scale of use complexity, increasing from operator software through actor software to activator software. The labelling is no more than a rough positioning of a particular software tool on this scale. It shows where the highest depth of the use complexity is located: so, what we are really talking about when making the categorization, is how far up in the activity the software brings to the work environment novelty that does not correspond to pre-existing elements in the outside world. The further this is, the more important it is that the

user learns to fully understand the software's UVM. He will need to put real effort into this. And to get him to do so, the documentation will have to pull out all the stops.

Full mastery of actor software has been reached only when the user's ambition encompasses activities which are completely shaped within the software environment. Such an ambition may not be pre-existent and must then be engendered during the learning process. The further the software 'reaches into' the activity, bringing novel things to do not only to the interactions themselves but also to the short-term or long-term ambition, the more scope there is for ill-defined variability and thus for the end results to be disappointing and the perceived load outweighing the perceived value; again putting an end to further upgrading and learning. Thus, the categorization indicates the relative importance of conceptual and strategic knowledge and with it, the demands placed on support materials (such as documentation) to foster their acquisition.

The Paradox of the Active User Re-phrased

Clearly, users of operator software have the least knowledge to acquire and those of activator software the most. Then, as the *volume* of required learning *increases*, the *ease* of learning *decreases* when we move from operator to actor and on to activator software. Correctly interpreting the external cues resulting from a particular interaction is easiest in operator software and most difficult in activator software, with actor software again occupying a middle position. Actors and activators need to take into account the context in which an interaction is carried out. "Stepping back" by one or two steps is proportionally more difficult and less likely to happen than directly relating outcomes to interactions. Also, by slowing down the CMA such stepping back runs into the production bias. Error detection is a strong trigger for upgrading to higher SRK levels of performance: thus, upgrading is least likely to be triggered in activator software and most in operator software.

This means that the relatively small amount of new knowledge required for operator software is sooner acquired than the larger amounts required for actor and activator software. Then, the chances of correctly interpreting system feedback so that upgrading and learning take place are higher in operator software than in actor software, and higher again in activator software. Finally, users of operator software may run into the assimilation bias but not the production bias, which first comes into play in actor software and becomes a real problem in activator software.

From the CMA model a new interpretation now emerges of the Paradox of the Active User: *The more learning is required, the less easily it takes place*. To which a corollary can be formulated as follows: *the more learning is required, the less users are likely to seek it*.

Operator software

In operator software, the ill-definedness is mainly at the level of operations and is kept in check by the well-definedness of the actions and the activity. As long as the user knows what he is doing, the conceptual and strategic knowledge required to remove the ill-definedness at operational level has a good chance of being actively sought. Even if it is never acquired, its absence can do only limited damage. In operator software, primary and secondary tasks are still reasonably well described in the 'traditional' manner employed when discussing the design of physical tools (see p. 20). Such manuals are produced from what in instructional design is referred to as an 'objectivist' point of departure, founded on the belief that knowledge can through the provision of information be transferred from one who possesses it (author) to one who as yet does not (the reader). This type of documentation is developed to enhance short-term performance of a particular task. The underlying assumption is that if the author writes down everything that the user needs to know in order to work with the software, then this information will be stored as knowledge on how to do things. Consequently, it is hoped, skills will emerge: which in operator software is a reasonable expectation. In operator software the tasks are pre-known, so that transfer to future performance is a matter of repeated practice.

Actor and activator software

To be helpful, it is often said that documentation must be task-oriented rather than system-oriented (Schriver, 1997). Yet the distinction between task-oriented and system-oriented becomes meaningless when user tasks are ultimately shaped by system capabilities. In such a case the documentation cannot just give users the straightforward procedural information that they ask for. Instead, it must give users what they need rather than what they want: information from which the knowledge and skills can be constructed needed to find their own solutions to their own problems.

In actor software, there are places where there is no well-defined goal to counter-balance the ill-definedness at the operational level; and in activator software even a well-defined intention is missing to direct the way the activity unfolds. The higher up we move from operator software to actor and activator software, the more need there is for meaningful, constructive learning, which it is distinguished from straightforward reproductive learning "by emphasizing personal meaning making and intentionally seeking to relate new ideas to experiences and prior learning and in doing so, engaging conceptual and strategic thinking" (Jonassen, 1999, p. 236).

The ‘constructivist’ point of view, as opposed to the objectivist one, holds that meaningful, applicable knowledge cannot be directly transmitted but is actively constructed by the learners, based on their interpretations of experiences (see also p. 47). If this is so, then documentation for actors and especially activators should not ‘dumb down’ by spelling out what to do in every imaginable possible context. Instead, the information sources should provide the raw material from which meaningful knowledge can be constructed during situation processing. Van Nimwegen found that a user interface that takes the user completely by the hand, has a detrimental effect on learning (Van Nimwegen, 2008). There is no reason to assume that the same would not hold for documentation that takes the user completely by the hand, spelling out what to do next without requiring the reader to wonder when or why to act. Users of software other than operator software would in the long term be best served if the documentation were to support the acquisition of long-term learning, resulting in enhanced future performance of related tasks.

Although little attention has been given to undirected, unsupervised learning, many guidelines have been drawn up for the constructivist design of instructional environments in which directed, supervised practice engenders learning (Collins, Seely Brown, & Newman, 1989; Ertmer & Newby, 1993, p. 75; Jonassen, 1999; Seely Brown, Collins, & Duguid, 1989; Spiro, Coulson, Feltovich, & Anderson, 1988; Spiro, Feltovich, Jacobson, & Coulson, 1992). These guidelines can be summarized as follows:

- Learning should take place in **authentic and real-world environments**. It is experience that provides the opportunity for knowledge construction, and this knowledge construction is enhanced by authenticity of the experience. The learning environment should have at its focus **an authentic question or issue, case, problem, or project that learners attempt to solve or resolve**.
- An emphasis must be placed on the identification of the context in which the skills will be learned and subsequently applied, **anchoring learning in meaningful contexts**. The learning environment should be **embedded in social interaction**, which provides the stimulus for the cognitive processing required for knowledge construction. The learning environment should **foster content and skills that are relevant** to the learner. The adaptive nature of knowledge construction requires that new knowledge is merged into the learner’s current knowledge base. Any new knowledge that seems irrelevant to the learner will not be stored. The case at the focus of the learning environment should therefore be **interesting, relevant and engaging**, as the key to meaningful learning is ownership of the problem. Ownership provides motivation.
- The learning environment should **place cases at the centre of the experience**. The real world shows great variability from case to case regarding the relevance of concepts and the combinatorial patterns of existing

knowledge. If general principles are placed at the centre of the experience, then problem solving becomes a matter of trial-and-error to see which principle applies in any given situation. This process will stop as soon as a reasonably satisfactory result is obtained; which may or may not be the optimal result. In this manner, suboptimal knowledge structures may be stored. The case at the focus of the learning environment should be **ill-defined or ill-structured**, so that some aspects of the problem are emergent and definable by the learner. The learning environment should **provide related cases**, to scaffold student memory (engender induction) and to provide multiple perspectives (enhance cognitive flexibility). The learning environment should **present conceptual knowledge as knowledge-in-use**, since in domains where there is no one-on-one mapping between elements in the problem space and elements in the solution space, the meaning of a concept is strongly connected to its application.

- Since a single representation will inevitably miss important facets of complex concepts, the learning environment should **provide multiple representations and perspectives** in order to allow the learner to develop complex, viable knowledge structures relevant to the experience. The knowledge that is constructed will be used in many different ways; therefore it also has to be learned, represented and tried out in application in many different ways. Different 'lenses' through which information is viewed, provide a multitude of ways of looking at one and the same problem or situation. The learning environment should **present information in a variety of different ways**; revisiting content at different times, in rearranged contexts, for different purposes, and from different conceptual perspectives. The learning environment should also **present alternative ways of representing problems** to support the use of problem-solving skills that allow learners to develop pattern-recognition skills and go 'beyond the information given'. The learning environment should stress the interconnectedness between multiple concepts and cases, to **mirror the complexity of the domain** and to **avoid compartmentalization of knowledge**.
- The learning environment should **demonstrate complexities and irregularities**: it should **highlight component interactions** to **foster combination of concepts**. Learners must be aware that the domain is not as simple and ordered as they might have wished, in order to avoid the application of overly simple and regular existing knowledge structures. The learning environment should **promote a shift from knowledge retrieval to knowledge assembly**. In well-structured domains where complexity is low, retrieval of knowledge suffices for problem solving. In ill-structured, complex domains on the other hand, existing knowledge must be combined and applied in novel ways: this is what constitutes learning.

- The learning environment should **provide guidance and facilitation** rather than direct instruction, and **promote active participation**. Knowledge cannot be just handed *to* the learner: it must be constructed *by* the learner. The learning environment should **provide access to rich sources of information** which must **be learner-selectable and just-in-time**. The learning environment should **provide cognitive tools** that scaffold the learners' abilities to perform the tasks.
- The learning environment should **stimulate meta-cognition**. The learner is responsible for the learning. Only through active processing can new knowledge be constructed, and without meta-cognition no adequate processing can take place. The learning environment should place an **emphasis on learner control** and the capability of the learner to manipulate information to **actively use what is learned**. The learning environment should **foster content and skills that build upon prior knowledge**, as inappropriate reasoning must become explicit before it can be adapted. The learning environment should **provide formative assessment** rather than evaluative assessment. Old knowledge is enhanced or replaced by the learner only when the result of its application does not match expectations. The learning environment should **provide assessment focused on transfer of knowledge and skills** by presenting new problems and situations that differ from the conditions of the initial instruction.

"Kunnen zonder kennen kan niet" says Tamara van Gog (van Gog, 2006), a highly alliterative Dutch phrase which my best attempt at translation, "Doing without knowing cannot be done", does not do justice. Only users who have the knowledge required to interpret the value of concepts that have meaning only in the software world itself, and for which no straightforward mapping exists with elements in the outside world, are capable of developing and implementing an appropriate strategy (Mirel, 1998a).

6. *SDDPL Value System and Organizing Principle*

A first version of this chapter has been presented in a writer's workshop at the EuroPloP 2016 conference, held on 6-10 July 2016 in Kaufbeuren, Germany. The current version will be included in the Proceedings (van Loggem, 2016).

Abstract: This chapter explores the question of how design patterns could be combined into and presented as a design pattern language. A choice for the organization of the SDDPL is made on the basis of usability, so that the language may meet the requirements of the two stakeholder groups in any design community: practicing designers, and those that record design knowledge for the former group to use.

Design Patterns and Design Pattern Languages

Designers would make little progress if they had to invent every wheel for themselves, over and over again. For individual designers to come up with the best possible solution to a particular design problem, and for a particular field as a whole to grow and improve, all available design knowledge in that field must be shared between practitioners. One way of recording and sharing design knowledge is through design patterns, collected in design pattern languages.

The idea of design patterns was first developed by Christopher Alexander, an architect, in the 1970s. Alexander (initially with collaborators) published a number of books on the design of urban spaces. Two of these went on to become seminal in other design communities: *A Pattern Language* (C. Alexander et al., 1977), and *The Timeless Way of Building* (C. Alexander, 1979). In Alexander's own words:

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. (A Pattern Language, p. x.)

Design patterns are intended not as how-to-recipes to be followed blindly and resulting in identical solutions to similar problems. Rather, as the name suggests, they must be seen as recurring patterns underlying widely varying solutions. In line with current practice rather than Alexander's original definition, I will use the following definition of a *design pattern*, taken from (Kohls & Uttecht, 2009):

Definition: A *design pattern* is a description of the invariant parts of proven designs as a solution to a problem in a specific context.

Design patterns are collected and combined in so-called design pattern languages. Again quoting Alexander:

All 253 patterns together form a language. (p. xxxv) [...] A pattern language has the structure of a network. (p. xviii) [...] to present each pattern connected to other patterns, so that you grasp the collection of all 253 patterns as a whole, as a language, within which you create an infinite variety of combinations. (A Pattern Language, p. xi)

Alexander sees his pattern language as the basis for each reader's own, individual, collection. He takes considerable space to "describe a rough procedure by which you can choose a language for your own project, first by taking patterns from this language we have printed here, and then by adding patterns of your own". The reader is invited to make a copy of the list of patterns (suggestions for an alternative working method are given in case the reader does not have access to a copying machine), then find in the list the pattern that best seems to describe the overall project and read it through. This becomes the starting point. Those 'smaller' patterns at the end of the overall pattern should also be marked for inclusion in the project, unless there is doubt as to their applicability. Next, the reader should turn his attention to the next highest pattern that has been thus marked, and the process is to be repeated. This continues until "you have ticked all the patterns you want for your project" but the process doesn't stop there: the reader is instructed to change any patterns as he sees fit, and to add any patterns that are felt to be missing (C. Alexander et al., 1977, pp. xxxviii – xxxix).

The first to take the pattern approach outside its original context as a way of describing the design of urban spaces were Kent Beck and Ward Cunningham (Beck & Cunningham, 1987). Building on their work, four authors who became subsequently known as the 'Gang of Four' wrote a book *Design Patterns—Elements of Reusable Object-Oriented Software* (Gamma, Helm, Johnson, & Vlissides, 1994) which is still widely used in the world of object-oriented programming.

Since then, the development and application of design patterns and pattern languages has become a field of interest in its own right. Successful conferences in the field of design patterns are held regularly. The 'Pattern Languages of Programs' or PLoP conference has been held in the US since 1994, with regional conferences (AsianPLoP, ChiliPLoP, EuroPLoP, KoalaPLoP, MensorePLoP, MiniPLoP, ScrumPLoP, SugarLoafPLoP, VikingPLoP) throughout the world. PUARL (Portland Urban Architecture Research Laboratory) started in 2011 and PURPLSOC (In Pursuit of Pattern Languages for Societal Change) in 2014. Despite their names, the work done at these gatherings is not limited to programming, architecture, or societal change but covers patterns in a wide variety of disciplines. This is reflected by a quick, by no means exhaustive, search on the Internet carried out on 3 December 2015 which yielded (self-styled) pattern languages for the design of,

amongst others, games⁶, websites⁷, something called Smart Cities⁸, behaviour-influencing elements^{9 10}, public spheres¹¹, instruction^{12 13}, enterprise integration¹⁴, productivity¹⁵, group dynamics¹⁶, and government services¹⁷. Areas in which design patterns have become a mainstream approach include software engineering, educational design (Kohls & Uttecht, 2009; Sharp, Manns, & Eckstein, 2003) and user interaction design (Borchers, 2001; Tidwell, 2010; van Welie, 2001).

In the transfer of design patterns from the domain of urban planning to that of object-oriented programming and beyond, Alexander's ideas were changed in a number of ways.

- First, the domain being a design technique (a particular programming method), the problems to which the patterns offered a solution no longer were those experienced by end users, but rather those experienced by the designers (that is: programmers) creating artefacts for end users.
- Second, in line with conventions in the programming domain, Alexander's verbose, 'narrative' way of describing individual patterns was often replaced by a strongly-structured format.
- Third, the explicit requirement was added for a design pattern to capture proven design solutions that can be found more than once in existing practice.
- Fourth, no more suggestion is made that the user create his own pattern language by selecting patterns from the collection, then modifying them as needed and adding his own.

There is little consensus on what exactly turns a collection of design patterns into a pattern language. In some of the collections mentioned above, a common format for the description of a number of patterns is seen as sufficient. In others, the implicit claim of a 'language' seems based on the interconnectedness of the individual patterns. This interconnectedness allows patterns to be combined into a complete design solution, and it is a core feature of Alexander's work as well as that of those which brought design patterns to the fields of object-oriented

⁶ gameprogrammingpatterns.com/

⁷ developer.yahoo.com/ypatterns/

⁸ theurbantechnologist.com/design-patterns/

⁹ designwithintent.co.uk/

¹⁰ ui-patterns.com/

¹¹ www.publicsphereproject.org/patterns

¹² csis.pace.edu/~bergin/PedPat1.3.html

¹³ www.cs.kent.ac.uk/people/staff/saf/patterns/gallery/MontessoriDesignPatterns1.pdf

¹⁴ www.enterpriseintegrationpatterns.com/

¹⁵ tools-for-thought.com/category/a-pattern-language-for-productivity

¹⁶ groupworksdeck.org/

¹⁷ www.gov.uk/service-manual/user-centred-design/resources/patterns/

programming (Gamma et al., 1994) and HCI (Borchers, 2001; Tidwell, 2010; van Welie, 2001).

Fincher (1999) suggests that for a collection of patterns to be a language, interconnectedness is only the first requirement. In line with Fincher, I define a design pattern language as follows:

Definition: A *design pattern language* is a number of interconnected design patterns related to a particular design discipline which share both a value system and an organizing principle.

The common value system and organizing principle of the Software Documentation Design Pattern Language are described in the next two sections.

Common Value System

First, consider the common value system that Fincher (1999) calls for.

A distinction is sometimes made between design patterns that help solving designers' problems and those that help solving problems for the people who work with the designed product (van Welie, 2001). The difference between the two, however, is not always clear-cut: presumably because all designers—whatever it is that they are designing—are driven by a desire to alleviate the problems of those they design for. For example, Tidwell's UI design pattern collection solves designer problems, while Van Welie's collection purports to solve user problems; as we see when comparing the opening paragraph of the "Use when" section for Tidwell's *Wizard* pattern (Tidwell, 2010, p. 55):

You are designing a UI for a task that is long or complicated, and that will usually be novel for users—not something that they do often or want much fine-grained control over (such as the installation of a software package).

with the "Problem" section in Van Welie's *Wizard* pattern (van Welie, 2001, p. 174):

The user wants to achieve a single goal but several decisions need to be made before the goal can be achieved completely, which may not be known to the user.

The words are different but the sentiment is the same. Ultimately, a designer works to alleviate his users' problems; design patterns, in turn, aim to alleviate the problems that the designer encounters when doing so.

Alexander placed considerable emphasis on justifying his pattern language, and by extension the individual patterns. On the first page following the front matter of *A Pattern Language* (p. ix) it is explicitly stated that this book and the book *The Timeless Way of Building* must be seen as two parts of an indivisible whole. Of this, *A Pattern Language* describes in close to 1200 pages the patterns and how to apply them, while the 500+ pages of *The Timeless Way of Building* are devoted to what Alexander calls “the quality without a name”. Not having a name, this guiding principle can be described only circumspectly, holistically, with reference to concepts such as wholeness, life, fire, glowing, entirety.

The Timeless Way of Building is a poetic, inspirational book. It offers the value system underlying the pattern language in terms of feelings and a state of mind rather than hard description. Tidwell’s common value system is expressed similarly, although much briefer, in terms of users’ happiness with the conversation they are conducting with a computer (Tidwell, 2010, Chapter 1). Van Welie on the other hand expresses his value system in much more measurable terms, based on a systematically conducted user task analysis (van Welie, 2001). Whether expressed in ‘soft’ or ‘hard’ terms, the presence of a common value system underlying the patterns in a collection provides a reason why the immediate problem that a particular pattern addresses needs solving in the first place (Dearden & Finlay, 2006).

Definition: A design pattern language’s *value system* is the conceptual framework within which justification can be expressed for the individual patterns in the language.

Such an underlying value system, common to all the individual design patterns, provides semantic coherence to a design pattern language. It is that which the language ultimately ‘talks about’.

Alexander’s pattern language talks about incorporating the “quality without a name” in people’s living environments. Tidwell’s pattern language talks about scripting the conversation between man and machine, based on an understanding of the human’s motives and intentions. Van Welie’s pattern language talks about streamlining interaction with software to minimize effort. The Software Documentation Design Pattern Language (SDDPL) talks about documenting software to empower its users.

A documentation artefact that is designed by applying patterns from the SDDPL aims to engender, over time, full mastery of the software that is being documented (as discussed in Chapter 5). Justification of the individual patterns is explicitly found in reference to the CMA model (as discussed in Chapter 4).

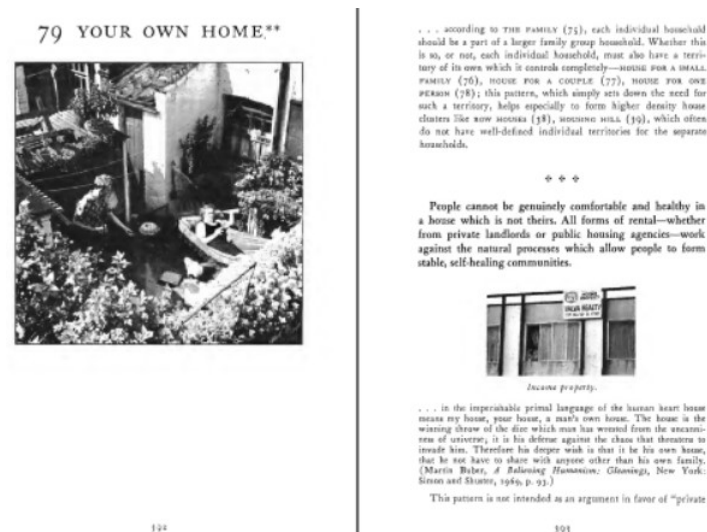
Common Organizing Principle

Now, consider Fincher's (1999) common organizing principle, or the way in which the design patterns are structured.

Definition: A design pattern language's *organizing principle* is the format in which the individual patterns in the language are presented.

Alexander's format is structured weakly. His patterns are presented in the form of narrative (running text). Every pattern opens with its name, and a confidence rating as to its general applicability. A picture showing an archetypal example is then followed by an explanation of how the pattern fits in with more general patterns. Three diamonds (❖ ❖ ❖) are followed by a headline, in bold type, summarizing the problem. After the headline comes a long stretch of text describing things such as empirical background, evidence for validity, and examples of implementation. Then, again in bold type and in the form of an instruction, the solution is given to the stated problem in the stated context and it is illustrated with a labelled diagram. Finally, following another line of three diamonds, smaller patterns are discussed which are needed to complete the one under consideration.

To illustrate the so-called 'Alexandrian' format, Figure 28 below shows one complete pattern taken from *A Pattern Language*.



TOWNS

property," or the process of buying and selling land. Indeed, it is very clear that all those processes which encourage speculation in land, for the sake of profit, are unhealthy and destructive, because they induce people to treat houses as commodities, to build things for "resale," and not in such a way as to fit their own needs.

And just as speculation and the profit motive make it impossible for people to alter their houses to their own needs, so tenancy, rental, and landlords do the same. Rental areas are always the first to turn to dums. The mechanism is clear and well known, too, for example, George Stedden, *The Tenement Landlord* (Routledge University Press, 1966). The landlord tries to keep his maintenance and repair costs as low as possible; the residents have no incentive to maintain and repair the houses—in fact, the opposite—since improvements add to the wealth of the landlord, and even justify higher rent. And so the typical piece of rental property degenerates over the years. Then landlords try to build new rental properties which are immune to neglect—gardens are replaced with concrete, carpets are replaced with linoleum, and wooden surfaces by formica; it is an struggle to make the new units maintenance-free, and to stop the dums by force; but they turn out cold and sterile and again turn into dums, because nobody loves them.

People will only be able to feel comfortable in their homes, if they can change their houses to suit themselves, add on whatever they need, rearrange the garden as they like it; and, of course, they can only do this in circumstances where they are the legal owners of the house and land; and it, in high-density multi-story housing, each apartment, like a house, has a well-defined volume, in which the owner can make changes as he likes.

This requires then, that every house is owned—in some fashion—by the people that live in it; it requires that every house, whether at ground level or in the air, has a well-defined volume within which the family is free to make whatever changes they want; and it requires a form of ownership which discourages speculation.

Several approaches have been put forward in recent years to solve the problem of providing each household with a "home."

394

can build, and change, and add on to their house as they wish.



+ + +

For the shape of the house, begin with BUILDING COMPLEX (95). For the shape of the lot, do not accept the common notion of a lot which has a narrow frontage and a great deal of depth. Instead, try to make every house lot roughly square, or even long along the street and shallow. All this is necessary to create the right relation between house and garden—MULF-HIDDEN GARDEN (111).

395

79 YOUR OWN HOME

At one extreme there are ideas like Hahnen's high density "support" system, where families buy pads on publicly owned superstructures and gradually develop their own homes. And at the other extreme there are the rural communes, where people have forsaken the city to create their own homes in the country. Even modified forms of rental can help the situation if they allow people to change their houses according to their needs and give people some financial stake in the process of maintenance. This helps, because renting is often a step along the way to home ownership; but unless tenants can somehow recover their investments in money and labor, the hopeless cycle of degeneration of rental property and the degeneration of the tenants' financial capability will continue. (Cf. Rolf Goetze, "Urban Housing Rehabilitation," in Turner and Fichter, eds., *The Freedom to Build*, New York: Macmillan, 1971.)

A common element in all these cases is the understanding that the successful development of a household's "home" depends upon these features: Each household must possess a clearly defined site for both a house and an outdoor space, and the household must own this site in the sense that they are in full control of its development.

Therefore:

Do everything possible to make the traditional forms of rental impossible, indeed, illegal. Give every household its own home, with space enough for a garden. Keep the emphasis in the definition of ownership on control, not on financial ownership. Indeed, where it is possible to construct forms of ownership which give people control over their houses and gardens, but make financial speculation impossible, choose these forms above all others. In all cases give people the legal power, and the physical opportunity to modify and repair their own places. Pay attention to this rule especially, in the case of high density apartments: build the apartments in such a way that every individual apartment has a garden, or a terrace where vegetables will grow, and that even in this situation, each family

395

Figure 28: The Alexandrian format (*A Pattern Language*, pp. 392-396)

The structure in the Alexandrian format is relatively weak. Almost all of the description comes in the form of narrative text and there are no subheadings. An evocative image helps draw the reader in. Then, typesetting changes are used to distinguish the different components of a pattern, which is separated from its larger and smaller patterns with no more than a sequence of diamond shapes at the very beginning and at the end, respectively.

In contrast, if we look at the domain of UID (this being the closest to user documentation), we see that with one notable exception (Borchers, 2001) most pattern languages apply a more strongly-structured template to the pattern descriptions, with fixed subheadings (Bernhaupt, Winkler, & Pontico, 2009; Deng, Kemp, & Todd, 2006; Kohler & Kerkow, 2008; Segerst hl & Jokela, 2006; Tidwell, 2010; van Welie, 2001). For an example, see Figure 29 below. However, even within this one discipline there is no consensus on what that structure should be. An XML DTD has been proposed for all pattern languages in the UID domain to adhere to, named PLML (Pattern Language Markup Language) (Deng et al., 2006; Fincher, 2003), but this has nowhere near been universally adopted. Every one of the pattern languages referenced in this chapter comes with its own structure evidenced by its own set of mandatory and optional subheadings.

174

Interaction Design Patterns

Wizard

Problem The user wants to achieve a single goal but several decisions need to be made before the goal can be achieved completely, which may not be known to the user.

Usability Principle User Guidance (Visibility)

Context A non-expert user needs to perform an infrequent complex task consisting of several subtasks where decisions need to be made in each one. The number of subtasks must be small e.g. typically between 3 and 10.

Forces

- The user is highly interested in reaching the overall goal but may not be familiar or interested in the steps that need to be performed.
- The task can be ordered but are not always independent of each other i.e. a certain task may need to be finished before the next task can be done.
- To reach the goal, several steps need to be taken but the exact steps required may vary because of decisions made in previous steps.

Solution Take the user through the entire task one step at a time. Let the user step through the tasks and show which steps exist and which have been completed.

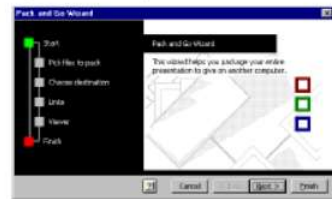
When the complex task is started, the user is informed about the goal that will be achieved and the fact that several decisions are needed. The user can go to the next task by using a navigation widget (for example a button). If the user cannot start the next task before completing the current one, feedback is provided indicating the user cannot proceed before completion (for example by disabling a navigation widget). The user should also be able to revise a decision by navigating back to a previous task.

The user is given feedback about the purpose of each task and the user can see at all times where (s)he is in the sequence and which steps are part of the sequence. When the complex task is completed, feedback is provided to show the user that the tasks have been completed and optionally results have been processed.

Users that know the default options can immediately use a shortcut that allows all the steps to be done in one action. At any point in the sequence it is possible to abort the task by choosing the visible exit.

Rationale The navigation buttons suggest the users that they are navigating a path with steps. Each task is presented in a consistent fashion enforcing the idea that several steps are taken. The task sequence informs the user at once which steps will need to be taken and where the user currently is. The learnability and memorability of the task are improved but it may have a negative effect of the performance time of the task. When users are forced to follow the order of tasks, users are less likely to miss important things and will hence make fewer errors.

Example This is the 'Pack 'n Go Wizard' from PowerPoint. The user wants to package a presentation so that the presentation can be given on another computer.



Several relevant decisions need to be taken and the wizard helps the user take these decisions. The green box shows the current position in the sequence of tasks.

Known Uses Microsoft PowerPoint Pack and Go wizard; InstallShield installation programs

Related Patterns Consider NAVIGATING SPACES and LIST BROWSER to provide the navigation controls.

175

Figure 29: Strongly-structured format (van Welie, 2001)

Usability Aspects

So, which is ‘better’, the Alexandrian narrative format or a strongly-structured format?

We should ‘practice what we preach’ and pay attention to the usability of our pattern languages (Kotzé & Renaud, 2008). The general consensus is that a strongly-structured format for pattern descriptions is a prerequisite for usability (Engel, Martin, Herdin, & Forbrig, 2013; Hennipman, Oppelaar, & van der Veer, 2008; Schobert & Schümmer, 2006; Todd, Kemp, & Phillips, 2004). However, it has also been concluded that “although the use of patterns is reported, there is little concrete evaluation of either the usefulness of pattern languages within the process or the contribution that they have made to the quality of the end product or to the design process (with notable exceptions)” (Dearden & Finlay, 2006). This conclusion was drawn after a thorough examination of well-respected design patterns and pattern languages in HCI, focusing on four key issues (“What is a pattern?” “What is a pattern language?” “How are patterns and pattern languages used?” and “Values and pattern languages”). The authors then laid out an extensive agenda for future research. In the intervening years, some of this research was carried out. Repeatedly, it was found that strongly-structured methods for presenting design patterns do *not* immediately make for easy learning, comparison, and application (Bernhaupt et al., 2009; Hennipman et al., 2008; Kohler & Kerkow, 2008; Kotzé & Renaud, 2008; Kotzé, Renaud, & van Biljon, 2008; Niebuhr, Kohler, & Graf, 2008; Segerståhl & Jokela, 2006).

Taking into account all the evidence, the assumption that a rigid description format under all conditions improves a pattern language’s usability seems untenable. Different stakeholders have different main uses of a system. The question which format is ‘better’ thus cannot be answered directly: as usual, the answer must be, ‘It depends’.

A pattern language is a way of recording information, that is, a notation. This being so, its usability can be evaluated using the framework of Cognitive Dimensions proposed by (Blackwell & Green, 2003), who pointed out that six generic types of notation-use activity can be distinguished and gave generic examples of all. They also pointed out that each has different requirements for support.

Table 2: Six types of notation-use (first two columns taken from (Blackwell & Green, 2003)).

use of the notation	use case (general)	use case (pattern-language-specific)
incrementation	adding cards to a cardfile, formulas to a spreadsheet, or statements to a program	adding patterns to the collection
transcription	copying book details to an index card; converting a formula into spreadsheet or code terms	including selected patterns into a working document, such as a plan of approach for a real-life design project
modification	changing the index terms in a library catalog; changing layout of a spreadsheet; modifying a spreadsheet or program for a different problem	changing pattern descriptions
exploratory design	sketching; design of typography, software, etc; other cases where the final product cannot be envisaged and has to be “discovered”	designing an intervention using the pattern language
searching	hunting for a known target, such as where a function is called	finding a particular, known pattern in the collection; or finding a specific aspect in a pattern description
exploratory understanding	discovering structure or algorithm, or discovering the basis of classification	discovering the underlying value system

Table 2 above shows the six uses, with examples, quoted verbatim in the first and second columns (note that the phrasing of the column headings is mine). The third column shows how the six uses are instantiated for the more specific case of the notation being a pattern language.

Individual users of a pattern language are unlikely to display all six use activities: a pattern language has more than one group of stakeholders. Design practitioners, or 'pattern users', are any pattern language's immediate stakeholder. But such a language does not come into existence without the efforts of a second type of stakeholder, one that we can label 'pattern writers': experienced designers who record their knowledge by writing patterns for others to apply in the course of their work.

For a pattern writer, the most relevant uses of a pattern language are incrementation and modification. A strong structure can ensure that nothing is forgotten and that nothing ends up in the wrong place. It may on the other hand also stifle the writer's creativity. Some of the writer's ideas may not fit comfortably under any of the language's headings, so that clarity is lost or valuable content discarded. Conversely, the writer may struggle to come up with content for all of the headings, leading to repetition and again loss of clarity.

For a pattern user, the most relevant uses of the language are transcription and exploratory design. Those patterns that are selected for real-life application are usually transcribed into a working document, for which some degree of structuring would be conducive. Before transcription can take place, however, the patterns must be selected in a process of exploratory design.

The Alexandrian format invites browsing more than searching. Indeed, as we have seen earlier (p. 90), Alexander never intended his pattern language to be searched for immediately-applicable guidelines. Design patterns do not take the creativity out of the design process and are intended to be applied by experienced designers rather than novices or those whose expertise is in a different design area (Appleton, 1997; Coplien, 1998). An experienced designer would not search a pattern language for a particular solution to a well-defined problem but approach the collection as a whole, looking for inspiration and interesting suggestions. The Alexandrian format then meets the needs of experienced designers better than a strongly-structured format would. Still, a pattern language should ideally cater for experienced and inexperienced users alike, so that the latter can over time hone their design skills in the chosen domain. Designing things, whatever these things may be, is after all a skill that can be developed only with practice. A strong structure supports searching more than browsing. As a consequence, it is better suited to disclosure through computerized means (de Moel & van der Veer; Deng et al., 2006; Engel et al., 2013). This would imaginably make strongly-structured languages more usable to novice designers or those whose experience is mainly in a different field.

A Proposed Format

Taking into account all the considerations, an attempt can be made to combine the best of two worlds, to support a user's exploratory design and transcription as well as a writer's modification and incrementation. Alexandrian weakly-structured narrative to invite browsing is then combined with prescribed subheadings to allow for a certain degree of searching. This gives a modified Alexandrian format in which typography is maintained, as is the order in which elements are presented. The reader's interest is piqued by an evocative image immediately following the pattern's name, and a short description of what the pattern looks like is placed underneath. The narrative is enhanced with a limited number of subheadings drawing attention to the main elements. Care is taken to ensure that the subheadings do not detract from the narrative and that the text can be read without reference to them; as is the case in the original Alexandrian format (shown in Figure 28) but not in a strongly-structured pattern language (e.g. Figure 29). Thus, the subheadings are complementary to the narrative. A reader who does not wish or need to understand a pattern in great depth can limit himself to scanning the overview and the paragraphs set in bold. Together with the pattern's name, these are also all that need to be copied when transcribing a pattern.

In this manner, the needs of both pattern users and pattern writers are met:

- When engaging in exploratory design, a user is encouraged to browse the language. Elements that are needed to quickly grasp the essence of a particular pattern stand out, so that patterns that are of no interest to the user can be discarded straightaway without the need for them to be read in their entirety.
- When transcribing the selected patterns into a working document, a user can skip the detail and easily focus on those same elements that together summarize the pattern.
- When modifying a pattern, a writer can quickly find the appropriate location to be changed, through the subheadings and boldface.
- When adding patterns to the language, a writer is guided by the subheadings without being stifled in his creativity.

The SDDPL presented in the next three chapters of this thesis uses the following subheadings:

- **problem**—the problem is phrased in terms of the underlying value system. Where can we expect spokes in the wheels of the knowledge engine? The problem is printed in a bold typeface.
- **discussion**—the discussion includes theoretical considerations as well as examples from practice.

- solution—the solution is phrased in practically applicable terms. Like the problem, the solution is printed in bold.
- rationale—the rationale explains how the proposed solution removes the problematic obstacle from the knowledge engine.
- consequences—consequences include considerations on the effects of the pattern on behaviour, and may lead to refinements of the proposed solution. When present, such refinements are also printed in bold, to clearly link them to the solution.

The SDDPL does not require for a solution to have been implemented repeatedly in practice before it is captured in pattern form. In the SDDPL, for a pattern to be a “proven design” as the definition requires, theoretical considerations are as acceptable as repeated practice. This was a conscious decision. Only by following the spirit of Alexander’s original definition (which requires that the *problem* occurs “over and over again in our environment”; not that the *solution* does so) can patterns be recorded that are informed by (fundamental or applied) research. This is needed to push practice beyond its current world view. “If you do what you always do, you get what you always get” as common wisdom has it; and then, little progress will be made.

The organizing principle of the SDDPL was created so as to incorporate the underlying value system and to meet the identified needs of the main stakeholders. Every pattern language will pose its own demands on the subheadings that are chosen. The subheadings in the SDDPL were chosen so as to reflect most accurately that particular language’s dual purpose. The SDDPL’s intended audiences include not only the stakeholder groups identified earlier (users and writers of design patterns in a particular field, in casu, that of software documentation) but also that personified by the examining committee of my PhD thesis. This is a non-standard audience for design pattern languages. In languages other than the SDDPL fewer references might be made to, for example, academic underpinnings.

In the next three chapters of this thesis, a small number of design patterns for software documentation is presented in this format. A hierarchy of scope is applied, as follows:

- Chapter 7 contains a number of *macro* patterns: patterns that are applied to a complete documentation set and underlie the complete approach to the problem of ‘how to document software product X’.
- Chapter 8 contains a number of *meso* patterns: patterns that are applied to a complete documentation artefact and drive the design of one particular document (which may or may not be part of a set).

- Chapter 9 contains a number of *micro* patterns: patterns that are applied to components of a documentation artefact, yielding something that cannot stand on its own.

It is hoped that practitioners, when designing a documentation artefact, will use the collection of design patterns much as Alexander envisaged his to be used (p. 90). Unlike Alexander's, however, this pattern collection is by no means even remotely exhaustive. The patterns were selected more or less randomly, based on informal discussion with various practitioners and academics. Some of those in Chapter 9 were mined in a 'focus group' during the EuroPloP 2015 conference in Kaufbeuren, Germany (see *Appendix 3. Using the Repertory Grid Technique for Mining Design Patterns*).

7. *Macro Patterns*

'Macro' patterns are applied to a complete documentation set and underlie the complete approach to the problem of 'how to document software product X'.

1. *Documentation Environment*



Documentation should empower the user. This will not happen when the documentation is something that stands between the user and the work. Rather, documentation must be part of the work environment, for the user to dip into when he feels the need; and be rewarded with a more thorough understanding and an increased sense of mastery.



problem

To master software with high use complexity, constructive learning is required in which the learner is actively engaged. A documentation journey that is driven by the author will not maintain such engagement but rather replace germane cognitive load in the form of upgrading with extraneous cognitive load, as the chance of the author having correctly predicted what the reader needs in any given situation is very small.

discussion

For novices interacting with activator software, a large amount of effort must be spent on the documentation journey to meet even the simplest of ambitions. The

documentation journey then truly becomes a separate task (see Figure 6), bringing much more load than immediate value. As a consequence, the knowledge engine can be expected to rapidly run down, and grind to a halt long before mastery has been achieved.

The German philosopher Martin Heidegger (1889–1976) tells us that people encounter things as either ready-to-hand (RTH) or as present-at-hand (PAH). Something that is PAH constitutes more load than something that is RTH. Creating artefacts (documentation) which are experienced as RTH despite being PAH may, in principle, be impossible (Wheeler, 2015). We can, however, try; by providing a documentation ‘environment’ as part of the work environment in which the user moves. The aim is to maximize the chance that when the user reaches for the documentation, there is something there that answers his felt information need. This will not reduce the amount of effort that is required, but it will reduce the subjectively experienced load.

A small-scale example of a documentation environment for users to access as and when they see fit is the *Crystal Caliburn Player Guide*¹⁸, a printed booklet accompanying a pinball simulator game. Rather than expecting the user to master the software by reading the manual, it says explicitly (in the last paragraph of the introductory page): “Please try playing first without reading the manual. Then read the explanation of features and try again. You will find new ways to enjoy the game.” This manual contains various types of information that is offered to the reader without much advice as to how or when to access it. Even the Arthurian legend is given to which the game’s name and visual design refer, as are various playing techniques that have proven their worth in the physical world of pinball machines.

A much more extensive example is found in software maker Adobe’s documentation package for the FrameMaker 5 product¹⁹, which consists of four separate printed products included in the box with the software, plus online Help, collections of templates and clip art, and additional manuals that can be downloaded in PDF format (see Figure 30, taken from the User’s Guide).

¹⁸ © 1992, LittleWing CO LTD

¹⁹ © 1995, Frame Technology Corporation

What to look for in print

The following information is available in print.

Product overview *Introducing FrameMaker* describes features and functions of FrameMaker. It includes samples of the kinds of documents you can create.

Installation instructions *Installing FrameMaker* explains how to install FrameMaker.

User's manual *Using FrameMaker* has step-by-step instructions for a comprehensive range of FrameMaker tasks. It is organized logically by task groupings.

Abbreviated task instructions and shortcuts The *Quick Reference* for FrameMaker contains abbreviated instructions for many tasks, and it lists all of the keyboard shortcuts. The *Pocket Guide* for FrameMaker lists keyboard shortcuts you can use when typing and editing text.

What to look for online

The following information is available online. You can reach most online information from online Help. You can reach online manuals from the Online Manuals pop-up menu in the Help main menu.

New and changed features If you are already familiar with earlier versions of FrameMaker, you may want to read the online manual *What's New in FrameMaker*. This manual describes the new and changed features in this release of FrameMaker and directs you to further information.

Overview *FrameMaker Overview* is a brief online introduction to key FrameMaker concepts. If you are new to FrameMaker, it is your best online starting point.

Online tutorial If you are new to FrameMaker, the online tutorial, *Learning FrameMaker*, will teach you the basics as you work with real FrameMaker documents. Do the tutorial after you read *FrameMaker Overview*. To start the tutorial, choose Tutorial from the Help main menu or choose Tutorial from the Help menu on the menu bar.

Last-minute information The online manual *Release Notes* provides late-breaking information on FrameMaker that could not be included in the printed manuals.

Online reference FrameMaker online Help includes both indexed and context-sensitive help with commands, dialog boxes, and keyboard shortcuts. (On a Macintosh, Balloon Help is also available. It provides short descriptions of palettes and document window controls.)

Custom use The online manual *Customizing Frame Products* explains how to customize FrameMaker and its environment. The Macintosh version of this document also explains how to add alternate math symbol fonts you can use in equations.

Templates Predesigned templates are available for a variety of uses—for example, common business correspondence, newsletters, reports, outlines, and books. To see a list of templates, choose New from the File menu and then click Explore Standard Templates.

Samples Sample documents, available through online Help, show the types of documents you can create with FrameMaker.

Clip art FrameMaker provides a collection of line art that you can copy and paste into your own documents.

Filters The online manual *Using Frame Filters* provides information on converting a wide range of documents between FrameMaker and other applications.

Multiple-platform use The online manual *Using Frame Products on Multiple Platforms* describes how the UNIX, Windows, and Macintosh versions of FrameMaker differ.

AppleScript commands (Macintosh only) The online manual *Using AppleScript with Frame Products* describes the commands that FrameMaker supports and how to use them.

Maker Interchange Format (MIF) The online manual *MIF Reference* explains the MIF text format, which allows FrameMaker to share information with another application while preserving graphics, text, and formatting information.

Maker Markup Language (MML) The online manual *MML Reference* describes a markup language that you can use with any standard text editor to create simple FrameMaker documents.

Figure 30: Documentation environment for FrameMaker 5 (User's Guide, pp. xiv-xv).

FrameMaker 5's set of documentation products is reminiscent of a so-called 'constructivist learning environment' or CLE. A CLE embeds instruction in authentic practice in accordance with the constructivist guidelines discussed earlier (see p. 72). CLEs provide learning experiences through an authentic problem, question or project surrounded with various interpretative and intellectual support systems. The goal of the learner is to interpret and solve the problem or complete the project. The support systems in a CLE take the shape of *modelling* (showing how the task could or should be performed), *coaching* (helping the learner to perform the task) or *scaffolding* (providing just-in-time materials that are helpful in performing the task). The scaffolding must be *faded* out of

existence as the learner's expertise progresses, as over time it will become redundant and add to the cognitive load rather than reduce it (Jonassen, 1999).

Constructivist principles have been applied extensively and successfully to multimedia and hypertext environments (Hummel, 2005; Spiro et al., 1988; Spiro et al., 1992). They could equally well be applied in documentation. Many of the constructivist guidelines are met by documentation naturally, without requiring designer effort (see Table 3). In the same way that a CLE is designed to provide explicit instruction, documentation can be designed to implicitly stimulate learning.

Table 3: Constructivist guidelines for instructional materials that are naturally met by documentation.

<ul style="list-style-type: none"> ▪ providing authenticity of practice ▪ anchoring practice in meaningful context ▪ embedding learning in social interaction ▪ stimulating the application of knowledge-in-use ▪ fostering active participation of the learner ▪ placing a case at the focus of the experience ▪ providing interest and relevance ▪ providing an ill-defined and ill-structured learning environment ▪ demonstrating complexities and irregularities ▪ demonstrating interconnectedness of knowledge ▪ revisiting content ▪ providing related cases 	<p>The use of documentation is always driven by authentic practice springing from externally imposed, real-life ambitions. Its very nature ensures that the other requirements listed are met.</p>
<ul style="list-style-type: none"> ▪ providing learner control ▪ providing access to information ▪ providing cognitive tools ▪ offering multiple representations and perspectives 	<p>Any form of documentation is accessed exclusively under the control of the user, at the exact moment that the need is felt to do so. Multiple representations and perspectives can easily be provided.</p>

solution	Therefore:
	For activator software, provide a large number of fundamentally different documentation products containing information of different types, formats, approaches, modalities etc.; offering modelling, coaching and scaffolding.
rationale	The variety of the information in a documentation environment ensures that there is always something that immediately meets the user's current need; whatever that need may be. This allows for upgrading leading to value, while the documentation seems useful and attractive so that the effort involved in the documentation journey is not experienced as high load.
consequences	Table 3 does not tell the whole story. Documentation may hold promises for fostering learning by meeting many constructivist requirements naturally, it is also handicapped in ways that explicitly designed instruction is not. Although learner control is a requirement that follows from constructivist principles, it is not under all circumstances an inducement to learning. The production bias can all too easily result in premature acceptance of a suboptimal solution, prohibiting the assembly of new knowledge structures. There exists a similar problem with the requirement of building upon prior knowledge. This will definitely take place if real-world tasks provide the learning environment, but the assimilation bias will do its utmost to stop the learner from rejecting any existing yet undesirable conceptions. As it is the learner himself who provides criteria and touch-stones for success or failure, his lack of knowledge may result in incorrect interpretation. Also, the assessment is by its very nature evaluative rather than formative and not automatically focused on transfer. Finally, documentation can stimulate meta-cognition only indirectly.

Therefore, when implementing this pattern:

Provide meta-information to explicitly define the context of applicability and the criteria for successful application.



... A DOCUMENTATION ENVIRONMENT can be seen as a complete set of JOB AIDS supporting not just the syntactic layers of operator software but also the semantic and task levels of actor and activator software...

2. Media Mix



Documentation can be offered in two fundamentally different media: hard copy (print) or soft copy (on-screen). As the choice of medium affects reader interaction with a document in a number of respects, a documentation set ideally includes both hard copy and soft copy. Only then can for every type of information the choice be based on relative strengths and weaknesses.



problem

When either hard copy or soft copy is selected for presentation of the documentation, there will always be some bits of information that are difficult to find, filter, or apply in the context of the current information need. Such information cannot be used for upgrading and constitutes extraneous cognitive load that is not offset by value.

discussion

To clarify the difference between hard copy and soft copy, the following is taken from page 14 of the Flare v11.1 Targets Guide (© 2015, MadCap Software). Note that this supplier of software for documentation development uses ‘online output’ to denote the more traditional ‘soft copy’, and ‘print-based output’ for the more traditional ‘hard copy’:

There is a fine line between what is called "online output" and what is called "print-based output." The truth is that topics in virtually any of Flare's online output types can be sent to a printer, and therefore considered print-based. Similarly, any of the print-based output types can be viewed electronically, and therefore considered online. The real distinction between online and print-based outputs has to do with their primary purpose. Online outputs are usually intended to be viewed on a screen, rather than on a printed page. The idea is to show only small pieces of content at a time and allow users to jump around to other topics or elements of the output. On the other hand, print-based output follows a more traditional format that you would find in an actual book or manual—with the pieces of the output following one after the other on pages until the end of the book (e.g., title page, table of contents,

preface, chapters, index, appendixes—with page numbers, as well as header or footer content, shown along the way). Then there is EPUB output, which is intended to be viewed on a screen, but follows a structure closer to print-based outputs.

In organizational practice, the choice between soft copy and hard copy is not usually made exclusively by a documentation designer. Other roles within a company will be involved, who bring to the decision considerations related to available resources (tooling, budget, manpower) as well as personal preference and prior experience. But the choice for soft copy or hard copy has far-reaching consequences for the reader's documentation journey. By creating a documentation set containing both hard copy and soft copy products, all of the information can be provided in the most suitable manner.

solution	Therefore:
	Rather than delivering the whole body of information in one documentation product, or multiple products deploying the same medium, create one or more hard copy products and one or more soft copy products depending on the nature of the information.
rationale	Information that is not, or not optimally, processed may as well not be present at all. By offering all information in the medium best suited to it, upgrading is enabled and extraneous cognitive load is removed from the documentation journey.
consequences	Simply providing a printed user manual alongside an online Help system does not in itself make a Media Mix. Rather, all the information must be offered through that medium that best suits it, balancing considerations such as the following.

mediation

Hard copy is a stand-alone information artefact that can be accessed without any further mediating technology. All that is needed, is sufficient ambient light. Soft copy, on the other hand, requires a device (a computer, smartphone, tablet, e-reader etc.) for the information to be accessed.

familiarity

Cognitive artefacts that are effortlessly recognized as 'books' have been around for hundreds of years. Countless generations have learned, often at a very young age, to interact with printed matter. Doing so is a skill that is reinforced by practice on an almost daily basis. Moreover, the long tradition of books means that their design has had a very long time to mature (Tebeaux, 1997). In contrast, soft copy is presented in a possibly unfamiliar environment that itself needs to be mastered before the information can be accessed.

modalities

Hard copy offers only a very limited number of modalities: text and static images. In addition, soft copy offers diagrammatic and realistic video, spoken or non-spoken audio, and any combination of these. (For a complete overview of all theoretically imaginable modalities see Bernsen, 1994.)

accessibility

In hard copy, the reader can access any and all of the information almost instantaneously and at any time, by turning to the appropriate page. It is possible to know immediately how much information there is and to see at a glance what the nature of that information is. In hard copy, the reader determines the information journey and the author can do no more than suggest an itinerary.

In soft copy, on the other hand, the reader can go from one bit of information to another only if the author has explicitly built links between them. A soft copy product has no structure other than that created by the author. Searching, too, is in soft copy limited to capabilities that have been explicitly built in by the author. In soft copy, the author creates the experience and the reader is limited in his choices: if he does not like what he gets, all he can do is walk out.

affordances

The two media allow for different ways in which the reader can get from one piece of information to another. As a consequence, they also offer different strategies to an author who wants to guide the documentation journey in a particular direction.

To understand this, the concept of *affordances* is helpful. Affordances are a powerful concept in (ecological) psychology, albeit one that means slightly different things to different people (Michaels, 2003). Its core is formulated in the following definition: "Affordances are the actions permitted an animal by environmental objects, events, places, surfaces, people, and so forth. An action is understood as a goal-directed movement (or non-movement) that entails intention, the detection of information, and a lawful relation between that information and the control of the movement. [...] Affordances exist independent of being perceived." (Michaels, 2003, p. 146). In the context of the information journey, affordances can be defined as follows:

Definition: *Affordances* are the opportunities for access to information permitted a reader by elements in the documentation.

Affordances may be purposely provided by the author; or they may occur as a consequence of unrelated design decisions. All other things being equal, the more affordances there are for a reader to access a 'next' piece of information, the less probable it is that that particular 'next' piece is accessed. Imagine a documentation

product consisting of n pieces of information. Were every piece to have exactly one affordance to another one, without any pieces being included more than once in the sequence, then there exists only one possible path through all the pieces. Unless the reader decides to opt out half-way through, the probability of the path being followed is 100%. At the opposite end of the spectrum, now imagine that every piece of information contains affordances to every other piece. Such a fully connected network structure contains $n!$ possible paths through all the pieces even if every piece would be accessed only once. The probability of a particular path being followed is $1/n!$, which decreases rapidly with increasing values of n .

Rather than leaving to chance which path a reader follows through a documentation product, the author of such a product will in most situations wish to identify a subset of the full network, creating or strengthening some affordances and downplaying others, to arrive at a design that presents information in a premeditated order. Readers choose one (perceived) affordance over another on the basis of the relative strength of the 'information scent', or the proximal cues to the nature of remote content (Pirolli, 1997, 2006). The probability of a particular affordance being selected is a function of the relative strength of its information scent (Chi, Pirolli, Chen, & Pitkow, 2001). Thus, the art and craft of stressing one particular path through a documentation product and downplaying others lies in the purposeful application of information scent.

Affordances, now, are highly dependent on the medium on which the information is recorded. In hardcopy, and in soft-copy where pages may display multiple pieces of information, there is information scent in proximity (Belew, 2000, p. 18); expectation (familiarity with a particular design leads readers to expect information of a certain type to be present in a particular location); and labelling (textual or visual cues as to the nature of the information). All these manifestations of information scent can be intentionally stressed or downplayed when designing a particular documentation product.

Over the centuries, a vast number of canonized affordances for hard copy have been devised to provide access mechanisms into the body of information. Some of these rely on the presence of page numbering (Table of Contents, index, a list of figures) while others do not ('tabbed' sections, alphabetical ordering, sequential heading numbering).

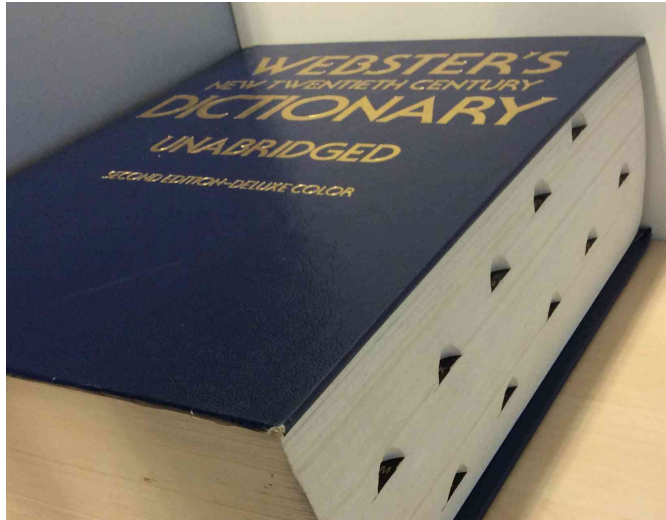


Figure 31: Tabbed sections in hard copy

Those access mechanisms that rely on page numbering can by definition not be used in soft copy, and neither can those that rely on physicality (such as the tabbed sections shown in Figure 31). Soft copy offers its own access mechanisms: menu structures, direct hyperlinks, and full-text search are the most familiar ones.

physicality

In hard copy, the reader physically interacts with the text: by reaching for a text or pushing it away, by inserting bookmarks, underlining, making written marks in the margin, folding the corner of a page or the spine or the stapled pile of sheets, etc. All these affordances are built into the medium of hard copy. In soft copy, only virtual approximations of the same actions can be carried out, and these only insofar as explicitly provisioned by the author (e.g., Small, 1999). There is evidence that when physically interacting with hard copy, readers incidentally remember where on the page and where within the text a particular piece of information was encountered (Mangen, Walgermo, & Brønnick, 2013; Rothkopf, 1971). This then helps them build a mental model of the information which in turn is positively correlated to improved comprehension (Cataldo & Oakhill, 2000; Mangen et al., 2013; Rothkopf, 1971).

Soft copy allows for 'hypertext': separate stretches of text connected by 'hyperlinks', which are hot spots in the text on which the reader can click, to be immediately taken to a different location. This can lead to a sense of being 'lost in hyperspace', where the reader no longer knows 'where' in the text he is (Charnock,

Rada, Stichler, & Weygant, 1994; Gwizdka & Spence, 2007; Otter & Johnson, 2000; Smith, 1996).

preference

At the turn of the century, the enthusiasm for the then-new medium of soft copy was such that its desirability was pre-supposed, and work was done on mimicking in soft copy highly physical ways of interacting with documents, such as that engaged in by Jewish scholars when studying Talmud and Torah (Small, 1999). Small's work bore the title "Re-thinking the Book", without considering whether the book needs re-thinking in the first place.

The novelty has worn off. Many people have a preference for hard copy over soft copy when it comes to text that needs internalizing, be it for work or for pleasure (Geske & Bellur, 2008; Schriver, 1997, pp. 383-389). Despite the immense computerization of knowledge work, the "paperless office" is still a myth (Sellen & Harper, 2002); and the market share of e-books is still limited, for example to 5.9% of total book sales in the Netherlands in the first quarter of 2016 (source: Centraal Boekhuis²⁰).

Prompted by such anecdotal evidence, numerous studies have been carried out to determine whether information presented in soft copy is processed in the same manner as that presented in hard copy. Occasionally, no differences are found (Askwall, 1985; Margolin, Driscoll, Toland, & Kegler, 2013) but more often, respondents are found to perform significantly better with hard copy than with soft copy: devoting more attention to the reading task (Geske & Bellur, 2008), scoring better on learning assessments (Emerson & MacKay, 2011), exhibiting a greater degree of understanding and creativity (Wästlund, Reinikka, Norlander, & Archer, 2005) or comprehension (Mangen et al., 2013; Mayes, Sims, & Koonce, 2001), showing more depth of recall (Noyes & Garland, 2003). Most of these studies' authors conclude that to the question of which medium is preferable, the answer must be, "It depends".

Therefore, when implementing this pattern:

- **Use hard copy:**
 - **for information that the reader should carefully consider and thoroughly understand;**
 - **for long stretches of narrative;**
 - **for information that needs to be accessed before the software is installed (such as installation instructions).**

²⁰ Accessed on 25 April 2016 at <http://www.cb.nl/wp-content/uploads/2016/04/6-E-book-barometer-NL-ENG-Q1-2016.pdf>

- **Use soft copy:**
 - **for information that is highly modular;**
 - **for information that need not be referred to for long stretches of time, nor away from the computer;**
 - **for information that is heavily cross-referenced;**
 - **when there is a requirement for animation or audio.**

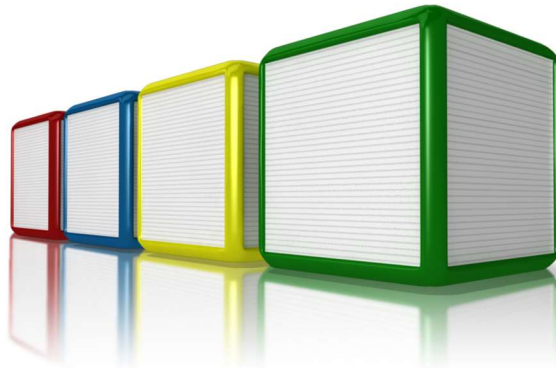


... As the choice of medium determines the possible modalities, a MEDIA MIX allows for the widest possible range of micro patterns such as SCREEN CAPTURES. A MEDIA MIX is easily combined with the SEPARATION OF PURPOSE pattern...

8. *Meso Patterns*

'Meso' patterns are applied to a complete documentation product and drive the design of one particular document (which may or may not be part of a set).

3. *Separation of Purpose*



A document is not a monolith but is constructed from smaller units of information. The author's main purpose when creating any of these is one of a very limited number. Making each unit 'stick' to its main purpose makes it easier for the reader to find and apply pertinent information.



problem **Faced with a large amount of information where no indication is given as to what to do with it, it is difficult for a reader to determine what to do with the information that is found. Applying the wrong information or applying information wrongly will yield undesirable results with low value.**

discussion A documentation product may contain a large amount of information. For software with high use complexity, it is impossible to structure the information so that it is in any context both correct and complete; and trying to do so means burdening the user with extraneous cognitive load. Rather than guessing, the solution is to start not from the reader's unknown needs but from the author's known purposes. The reader will then be able to determine where to look and what to do.

Recorded information (such as documentation) is always intentional, in that the author has a desire to have a particular effect on the reader (e.g. van der Meij, 1997). This purpose is the one taxonomy of information that is truly time-invariant, medium-invariant and method-invariant. Whether we look at Sumerian


clay tablets inscribed thousands of years BCE (see Figure 35), at technical manuals published in Renaissance England (for examples, see Tebeaux, 1997), or at the Help system for a scientific software package, we find units of information each having a particular purpose.

Over the ages, the number of possible purposes for recorded information has remained limited to the same handful. The author's purpose may be: for the reader to read the information and act on it (read-to-act); to read the information and remember it (read-to-know); to read the information and consider it (read-to-consider); or to scan for a particular piece of information that is then immediately applied and after its application, forgotten (read-to-use). In addition to these four, other purposes can be for the reader to read and enjoy the information (as in literature or entertainment) or to read the information and change his beliefs (as in sales materials or political propaganda). These two are by their very nature not often encountered in documentation.

read-to-act

Read-to-act information is created by an author wanting for the reader to act in a particular manner. Traditionally, read-to-act information is presented in the form of stepwise instructions (see p. 166), outlining the sequence of steps that must be followed. They can however equally well be presented as running text or in non-textual formats such as audio ('walk-throughs'), video (as popular on YouTube), or diagrams such as flow charts.


Voice calls

- 1 In the home screen, select  or **Dialler** to open the dialler, and enter the phone number, including code. To remove a number, select **C**.
For international calls, select ***** twice for the + (which replaces the international access code), the country code, area code (omit the leading 0 if necessary), and phone number.
- 2 To make the call, press the call key.
- 3 To end the call (or to cancel the call attempt), press the end key.

The following steps show you how to create an autonumber format for a single paragraph or the option of creating an autonumber format for a style. It is recommended that you use it whenever possible. A style allows you to apply the same format to multiple paragraphs at once, and any changes to the format are applied automatically to all the paragraphs using the style.

HOW TO CREATE AN AUTONUMBER FORMAT FOR A PARAGRAPH

1. Open the content file (e.g., topic, snippet).
2. Highlight or place your cursor in the paragraph to which you want to apply an autonumber format.
3. Do one of the following, depending on the part of the user interface you are using:

» **Ribbon** Select the **Home** ribbon. At the bottom of the **Paragraph** section, click the  icon.

You can use the **Options** dialog to switch between ribbons and the classic interface. For more information see the online Help.

Keep in mind that the smaller the application window becomes, the more the ribbons shrink. Therefore, you might only see a small icon instead of text, or only the section name displayed with a down arrow to access the options in it. You can click the small icons to see tooltips that describe them. You can also enlarge the application window to click one of the section drop-downs in the ribbon to locate a hidden feature.

» **Tool Strip** Select **Format > Paragraph**.

» **Keyboard Shortcut** Press **CTRL+ALT+B** on your keyboard.

» **Right-Click** Right-click on the paragraph and from the context menu choose **Autonumber**.

The **Paragraph Properties** dialog opens.

4. Select the **Autonumber** tab.
5. (Optional) From the **Available commands** drop-down list, you can filter the autonumbers shown in the area below by selecting one of the options.
 - » **Show All** Displays all of the commands in the area below.
 - » **Show AutoNumber Commands** Displays only the autonumber commands in the area below. These include commands such as chapter, section, and volume number and series labels.

36

Figure 32: Read-to-act information in (left) a user manual for a mobile telephone²¹ and (right) one for a Help development environment²².

Figure 32 shows read-to-act information at the syntactic layer. When internalized, this becomes procedural knowledge. Read-to-act information can also discuss

²¹ Nokia N97 mini User Guide, ©2009, Nokia

²² Flare 11.1 AutoNumbers Guide, ©2015, MadCap Software

steps to be taken at the semantic layer. When internalized, the information then becomes strategic knowledge.

read-to-know

Read-to-know information is created by an author wanting for the reader to understand concepts and facts that are part of the software world. Such information is value-free and can be re-used in different situations, under different conditions, from different perspectives and to different ends. Read-to-know information is descriptive and often presented in narrative text (perhaps containing lists and/or graphics), or in static or dynamic graphics enhanced with textual labels.

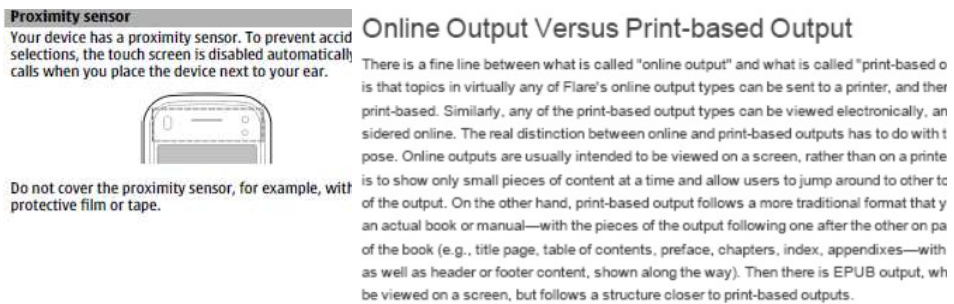


Figure 33: Read-to-know information in (left) a user manual for a mobile telephone²³ and (right) one for a Help development environment²⁴.

The leftmost image in Figure 33 shows read-to-know information at the syntactic layer. When internalized, this becomes situational knowledge. The rightmost image in Figure 33 shows read-to-know information at the semantic layer. When internalized, this becomes conceptual knowledge.

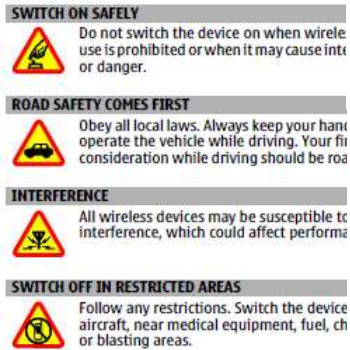
read-to-consider

Read-to-consider information is created by an author wanting for the reader to consider a particular circumstance when accessing or applying other types of information. It may be visually marked so as to indicate a position outside the main information stream. Read-to-consider information may aim to direct the use of the documentation (e.g. lists of related documentation products, instructions on

²³ Nokia N97 mini User Guide, ©2009, Nokia

²⁴ Flare 11.1 Targets Guide, ©2015, MadCap Software

how to read a manual, or a table of contents) or that of the system itself (e.g. warnings, exhortations, tips, examples, and vignettes to enhance motivation).



The following steps show you how to create an autonumber format for a single paragraph or the option of creating an autonumber format for a style. It is recommended that you use the option of creating an autonumber format for a style. A style allows you to apply the same format to multiple paragraphs at once, and any changes to the format are applied automatically to all the paragraphs using the style.

Figure 34: Read-to-consider information in (left) a user manual for a mobile telephone²⁵ and (right) one for a Help development environment²⁶.

Read-to-consider information is not likely to be internalized separately.

read-to-use

Read-to-use information is created to relieve the reader of the burden of internalizing straightforward facts that can be directly applied, without further consideration.

In a non-software context, a telephone directory is read-to-use information, as is an ingredients list, or a record of the harvest of ripe dates in a particular year (see Figure 35).

²⁵ Nokia N97 mini User Guide, ©2009, Nokia

²⁶ Flare 11.1 AutoNumbers Guide, ©2015, MadCap Software



Figure 35: Read-to-use information, recorded in 2035 BCE. This Sumerian clay tablet records the harvest of ripe dates in two gardeners' date palm orchards²⁷.

In documentation, read-to-use information provides a reference sheet, for look-up purposes. It lists facts that the user cannot be expected to remember in their entirety even after a long period of intensive use of the software. A reference sheet can simply be presented in tabular form on one or more pages in the documentation product, giving an overview of (for example) toolbar icons, parameters and their consequences, keyboard shortcuts, usage modes, installation requirements or 'known limitations'. Other formats are also possible: a series of tooltips is also a reference sheet, albeit one where only one item in the list is shown at any given time in any given context (see Figure 37). Yet another example is a plasticized template that fits on the physical keyboard and shows what the various function keys do.

²⁷ http://www.brown.edu/Facilities/University_Library/libs/hay/focus/cuneiform/#translations

Display indicators

The device is being used in a GSM network (n service).



The device is being used in a UMTS network (n service).



You have one or more unread messages in the folder in Messaging.



You have received new e-mail in the remote m



There are messages waiting to be sent in the



You have missed calls.



The ringing type is set to Silent.

AUTONUMBER COMMANDS

- » **{n}** Retains the current counter value and displays it. You might use this command, for example, if you are applying autonumber formats to multi-level paragraphs, where the first paragraph acts as the "parent" to another. Let's say the first-level paragraphs are numbered 1.0, 2.0, 3.0. If you want the second level paragraphs to keep the first number of the parent paragraph and increment the second number (e.g., 1.1, 1.2, 1.3), you would use the **{n+1}** command to continue displaying that first number, which represents the parent (in this case, 1).
- » **{n=1}** Resets the counter value to 1 and displays it. You can replace the number 1 with any other number that you want to use.
- » **{=0}** Resets the counter value to 0 but does not display it. You can replace the 0 with any other number that you want to use.
- » **{n+}** Increments the counter value and displays it. You might use this command, for example, to increment a list of step-by-step procedures (e.g., 1., 2., 3.).
- » **{}** Retains the current value and does not display it. You might use this command, for example, if you are creating an outline with Roman numerals at the first level and alpha numerals at the second level. If you are creating the format for the second level, you want the autonumber format to keep track of the fact that it is a "child" of the first level, but you do not want to display the Roman numeral from it (e.g., IV.A.). In this case, you only want to display the uppercase alpha letter (e.g., A). In order to do this, you use the **{}** command at the place where the Roman numeral would normally be displayed.
- » **{secnum}** Displays the current section number. You can use this command if you are generating online output, or Word, XPS, PDF, or XHTML output. This command does not work for FrameMaker output.



Note: To generate section numbers, you need to create an autonumber format that includes the **{secnum}** command. Then you need to specify section break codes in the outline TOC.

CHAPTER 4 | Autonumbers Guide

Figure 36: Read-to-use information in (left) a user manual for a mobile telephone²⁸ and (right) one for a Help development environment²⁹.

²⁸ Nokia N97 mini User Guide, ©2009, Nokia

²⁹ Flare 11.1 AutoNumbers Guide, ©2015, MadCap Software

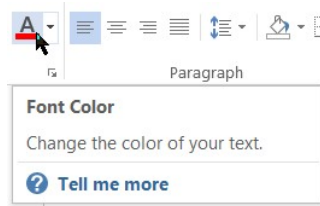


Figure 37: Tooltip (with reference to more in-depth information), in Microsoft Word 2013

Read-to-use information is not intended to be internalized other than, perhaps, over time as a by-product of use.

The problem with mixing up information of the different types becomes apparent in the fragments shown to the right-hand side in the figures in this section (the ones taken from the Flare 11.1 user manual). Figure 32 shows steps 1-5 in a block read-to-act information, starting on page 36 of the manual. Step 6 is given on page 38, and this procedure actually ends only with steps 9-11 on page 44. In between, there is mainly read-to-use information (as on page 39, Figure 36) and read-to-consider information in the form of examples and notes. As a result, the procedure is very difficult to read or follow along. At the same time, the other types of information are difficult to find. Finally, every attempt has been made to ensure completeness of the read-to-use information, resulting in considerable repetition. The sequence that occurs on page 39 (Figure 36) is also printed in full on page 29, in the context of an only marginally different procedure. It will be very hard for a reader needing that particular overview to find it; and if the two instances are subtly different because of the different contexts in which they are presented, there is a real risk of the reader ending up at the wrong instance and applying the wrong information.

solution Therefore:

Separate the different types of information in accordance with the height of the use complexity of the software. The higher the use complexity, the more strictly the different types should be separated.

rationale

When creating documentation for actor or activator software, the designer cannot reliably predict the context in which a particular piece of information is applied. One piece of information can be appropriate in many different contexts, depending on the user's current goal. Only the reader can know whether a particular piece of information is required. Having to separate the wheat from the chaff would constitute extraneous cognitive load. Doing the separation for him will help with the searching, filtering and applying stageposts of the information journey and increase the likelihood of valuable results.

In an environment with low use complexity, on the other hand (such as the fragments shown to the left-hand side of the figures in this section, taken from the Nokia N97 mini documentation), there is much less ill-definedness. Every bit of information is part of one workflow only and can be included in the documentation at the appropriate point in that workflow. Putting it ‘where it belongs’ in this situation does not add extraneous cognitive load but rather lowers it, by removing the need to look for related and relevant information elsewhere (see also Figure 47).

consequences Different types of information block bring slightly different consequences.

Read-to-act information is appropriate where there are ‘novel’ ways of interaction for the user to choose from: that is, where there is width in the use complexity. This type of information immediately increases the value. Playing along with the production bias, read-to-act information is known to be the first thing that readers look for in software documentation (Duggan & Payne, 2001; Eiriksdottir & Catrambone, 2014; D. K. Farkas, 1999; Karreman & Steehouder, 2004; Karreman, Ummelen, & Steehouder, 2005; van der Meij, Blijleven, & Jansen, 2003; van der Meij & Gellevij, 2004; van Loggem, 2007).

Read-to-know information is appropriate where there is depth in the use complexity. This type of information increases the value only in the long term; in the short term, it adds to the load. For operator software, to counteract the production bias it may be sufficient to offer the appropriate read-to-know information alongside the corresponding read-to-act information. For actor and activator software, there is no one piece of read-to-act information where the read-to-know information finds a natural home; and the only way extraneous cognitive load can be minimized is by making crystal-clear what knowledge is missing and where it can be obtained.

Read-to-consider information is meta information, not directly related to the work the user is carrying out. It will therefore always bring little subjective value to offset the load. Much read-to-consider information can be omitted without adverse effects—for example, typographical conventions in a printed manual are deployed to help the reader, and should not need explicit explanation.

Read-to-use information is ‘external memory’, and this type of information reduces cognitive load by freeing up processing power for things that matter. It adds to the value but at the cost of situation processing, and therefore should be used only when the latter is not required. Read-to-use information contains items which each have meaning in total isolation from one another and achieves the highest possible reduction of extraneous cognitive load if a particular item can be found with as little effort as possible.

Therefore, when implementing this pattern:

No special effort has to be taken in order to get readers to access read-to-act information.

When creating separate units of read-to-know information, mention them in the read-to-act contexts where they are relevant and provide clear cross-references.

Include read-to-consider information only where the benefits are obvious (note however that in the case of safety warnings and copyright information, legal requirements may exist).

Create read-to-use information only when there is little harm in the information not being internalized, and pay attention to findability of both the overview itself and one item amongst many.



... For the SEPARATION OF PURPOSE pattern to work, the information blocks must be linked through CROSS-REFERENCES. A CONCEPTUAL MODEL can be used where appropriate to link blocks of read-to-know information ...

4. Motivator



When there is much to learn, the user will need to be motivated to persevere: both with the software and with the documentation.



problem **The ambition to get the job done may not remain high enough for long enough for full mastery to be achieved.**

discussion Continuing learning in the CMA model relies on the ambition eventually covering the full functionality of the software, and remaining at that level until full mastery has been achieved. There are situations in which this may be too tall an order for ambition that springs from real-life demands. The full functionality of software with high use complexity is not known from the beginning, and such software takes by definition considerable effort to master. The load is exacerbated by its documentation being more voluminous than that for simpler software products. But the ambition may also be insufficiently high if the use complexity is relatively low: for example, when the intrinsic motivation is low in a particular individual, or when the learning is difficult for a particular group of learners (such as the elderly), or when bad design of the user interface introduces unnecessary complication. A support mechanism must then be installed to reinforce the existing ambition, to boost the engine when it is in danger of running down.

Practitioners have long been aware of the importance of a documentation product's motivational value. Not long before cell phone manufacturer Nokia was bought by Microsoft, they completely re-designed their user manuals to be attractive and inviting; Nokia's Senior Offering Manager Minna Vänskä presented the process to a fascinated audience at the 2013 conference organized by TCeurope, the European umbrella organization for technical communication³⁰. The hugely successful 'for Dummies' series of reference books (published by Wiley, Hoboken, NJ, USA) that help people "to solve problems and get up to speed on topics that may seem difficult or intimidating"³¹ started out as a series of books on popular software packages such as Microsoft Word™ and Excel™. And guidelines have been provided (Horton, 1991) for the design of "user-seductive" documentation to "woo and win the reluctant reader" (see Figure 38).

Satisficing (p. 50) is a powerful mechanism, but it is not the only determinant for ambition and upgrading. An activity may be challenging in such a manner that its challenging nature is also its appeal: athletes, artists, and amateurs do not satisfice. Learning can be rewarding in itself and 'finding out about' or FOA can be fun (Belew, 2000). In instructional design, the ARCS Model Approach to Motivational Design for Learning and Performance (John M. Keller; John M Keller, 2009) offers tools and strategies in the following areas:

- generating and sustaining attention (A)

³⁰ presentation slides accessed on 25 April 2016 at <http://slidehot.nl/resources/building-a-content-portfolio-for-multi-channel-publishing-presentation-tceurope-colloquim-brussels-2013.214524/>

³¹ <http://eu.wiley.com/WileyCDA/Brand/id-9.html>

- establishing and supporting relevance (R)
- building confidence (C)
- managing outcomes for satisfaction (S)

The first three of these were applied in Loorbach's work, who added specific motivational elements to a user manual for a cell phone and found positive effects with senior citizens on confidence, motivation and usability (N. Loorbach, Karreman, & Steehouder, 2007; Nicole Loorbach, Joyce Karreman, & Michaël Steehouder, 2013; N. Loorbach, J. Karreman, & M. Steehouder, 2013; Nicole Loorbach, Steehouder, & Taal, 2006).

solution	Motivate the performer in his role as reader to persevere with the documentation and in his role of user to persevere with the tool.
rationale	Motivation can be thought of as a 'boost' to upgrading. As a bonus, higher motivation reduces subjective load.
consequences	Although Loorbach achieved positive effects by adding motivational elements to existing user manuals, it must be borne in mind that her studies were done with a very specific group of subjects, that is, senior citizens. In addition, the software system being documented has little use complexity and as a consequence, the user documentation was limited. These two aspects may account for the results contradicting many others. As early as 1913, Dewey warned against attempts to spice up existing materials with interest-enhancing detail (Dewey, 1913). Since then, it has repeatedly been found that such a strategy is counterproductive (for an overview of the literature, see Harp & Mayer, 1998). For products with considerable use complexity, elements such as informally phrased exhortations, or personas sharing their 'personal' experiences, will be annoying rather than motivating.

1 Woo and win the reader	Warm-up act Document halitosis Editor's tip #1 Editor's tip #2	7 Practice heroic hedonism Offer psychological rewards Make the reader laugh Spark fantasies
2 Flirt with the reader	4 Keep the conversation going	8 Publish sensual pages
Make your availability known Dress for seduction Wink at the reader Break the ice	Especially for electronic documents Echo Linking words and phrases Repeated key word or object Consistent viewpoint Road map and signposts Psychological momentum Elbow joint Sudden reversal Clear structure Blaze a trail for the user	A treat for the eyes Sweet sounds The intimacy of touching
3 Make small talk	5 Not Goodbye but Au Revoir	9 Build a lasting relationship
How to learn small talk Problem-solution "Hello, I'm ..." Name badge Scenario Definition Question Technical breakthrough Contrast Shocker or tickler Funnel in Picture this First person, first interest Quotation	Summarize Recall main advantage Tell the reader to act Make the reader think and feel Point forward Funnel out Wave goodbye Test, err I mean, game	Honesty Openness Growth Commitment
	6 Get and stay in shape	Index
	Put the document on a diet Layer information Design the body for fast scanning Design for efficient lookup	Credits

Figure 38: Contents of "Secrets of User-Seductive Documents: Wooing and Winning the Reluctant Reader"(Horton, 1991)

Therefore, when implementing this pattern:

- **Treat 'motivation' as an all-pervading characteristic of the documentation (as in the ARCS model) rather than something that can be added afterwards.**
- **Look for motivational elements that do not add to the load of using the documentation.**
- **Use 'fun' elements for low-complexity software only.**



... Motivation is especially important when the learning is completely under the user's control, as in the EPPO and DOCUMENTATION ENVIRONMENT patterns...

5. Every Page is Page One (EPPO)



Since users cannot be forced to read the documentation in any order prescribed by the author, every 'page' of a documentation product can in practice be the first. Rather than fight this fact of life, we could cater for it.



problem

When all is said and done, the reader will read the documentation in an order determined by himself. In doing so, he may miss interesting and useful pieces of information. Especially when there is high use complexity, it is necessary for the user to be shown the existence of concepts that he is not yet aware of; so that his ambition is gradually raised and eventually, over a number of iterations, the documentation journey will have covered all that he needs to achieve full mastery.

discussion

In his book *Every Page Is Page One* ("EPPO"), technical writer Mark Baker points out that readers of documentation search for the solution to an actual problem rather than satisfying a more general information need. He therefore advocates documentation consisting of a potentially large number of separate 'topics' which have the following characteristics (quoted verbatim from Baker, 2013, p. 78):

- **Self-contained:** An EPPO topic is self-contained. It has no previous topic and no next topic. It does, however, rely on the whole information environment in which it is located for supporting and ancillary information.
- **Specific and limited purpose:** An EPPO topic has a specific and well-defined purpose. This is highly related to the purpose of the person who is reading it, but it is not the same thing. One topic has to serve many readers, and is designed to serve a community, not an individual.

- **Conform to type:** It turns out that, unlike book length content, Every Page is Page One topics seem to naturally conform to fairly well-defined types, often the result of a community process that develops the best way to treat a particular kind of subject. The type of a topic is based on its purpose: the type defines the information necessary to serve its purpose.
- **Establish context:** Readers can come to an Every Page is Page One topic from anywhere. An EPPO topic must establish its context in the real world so readers know where they are and what to expect.
- **Assume the reader is qualified:** An EPPO topic assumes readers are qualified to complete the specific and limited purpose of a topic. Readers who are not fully qualified can read other topics to get the information they need.
- **Stay on one level:** Books tend to change their level of abstraction and detail over the course of the narrative. But information foraging readers prefer to choose for themselves whether to go for detail or the big picture. An Every Page is Page One topic stays on one level and allows readers to change levels whenever they wish by changing topics.
- **Link richly:** An EPPO topic is meant to support effective information foraging. Therefore, it links richly along the lines of subject affinity to help the reader follow the scent of information.

Not mentioned explicitly but assumed implicitly is the need for a meaningful name for every topic.

An example of EPPO given by the author is Wikipedia³². Wikipedia articles (for an example, see Figure 39) are self-contained and can be read in isolation. They are clearly scoped, stay on one level, conform to a particular type, and are embedded in the context of a greater whole through hyperlinks to related articles. No matter how much or how little a reader already knows about a subject, after reaching a pertinent article he can always easily tailor the information journey to his particular situation.

³² www.wikipedia.org

Article [Talk](#) [Read](#) [Edit](#) [View history](#)

Icelandic horse

From Wikipedia, the free encyclopedia

The **Icelandic horse** is a [breed](#) of [horse](#) developed in [Iceland](#). Although the horses are small, at times [pony](#)-sized, most [registries](#) for the Icelandic refer to it as a horse. Icelandic horses are long-lived and hardy. In their native country they have few diseases; Icelandic law prevents horses from being imported into the country and exported animals are not allowed to return. The Icelandic displays two [gaits](#) in addition to the typical walk, trot, and canter/gallop commonly displayed by other breeds. The only breed of horse in Iceland, they are also popular internationally, and sizable populations exist in Europe and North America. The breed is still used for traditional sheepherding work in its native country, as well as for leisure, [showing](#), and [racing](#).

Developed from ponies [taken to Iceland by Norse settlers](#) in the 9th and 10th centuries, the breed is mentioned in literature and historical records throughout Icelandic history; the first reference to a named horse appears in the 12th century. Horses were venerated in [Norse mythology](#), a custom brought to Iceland by the country's earliest settlers. [Selective](#)

Icelandic horse



Icelandic horse performing the [trot](#)

Distinguishing features	Sturdy build, heavy coat, two unique gaits .
Alternative names	Icelandic Pony
Country of origin	Iceland
Breed standards	
United States Icelandic Horse Conference	Breed standards
The Icelandic Horse Society of Great Britain	Breed standards

Figure 39: Part of a Wikipedia article³³

solution

Therefore:

Do not build a prescribed or even suggested reading order into the documentation product but make it consist of separate, self-contained yet interlinked articles so that the reader can map out his documentation journey driven by the currently felt need for information. Make sure that the whole User Virtual Machine is covered but leave to the reader which parts of the documentation to access as and when required, depending on his current activity and level of expertise.

rationale

When no particular reading order is given, the reader is forced to actively consider the documentation in order to find in it what he needs at any given moment. This provides germane cognitive load and at the same time reduces extraneous cognitive load from the documentation journey, as all the information that is found will be pertinent and meet the felt information need. The wealth of cross-references, finally, allows the author to show the existence of related, relevant information which is easily reached; this raises the ambition.

³³ screen captured on 6 October 2016 from https://en.wikipedia.org/wiki/Icelandic_horse

consequences The study described in detail in *Appendix 2. "Nobody Reads the Documentation"—True or Not?* showed a predilection for 'unauthorized' sources of information such as whoever happens to be around, or sources found on the open Internet. These results fit a repeatedly-confirmed observation in areas other than that of recourse to (software) documentation. Time and again it is found that ease of access and convenience are the strongest determinants for the choice of an information source, with online browsing as the single most popular method for seeking information (Connaway, Dickey, & Radford, 2011; Fast & Campbell, 2004; Julien & Michels, 2004; K.-S. Kim & Sin, 2011). Even software users who fully realize that answers obtained from a non-authorized source are often not of the same quality as those obtained from the software's supplier—who, having created the software, at least in theory knows it better than any third party ever could— can hardly be blamed for taking the easy route first. It cannot be denied that asking the person at the next desk or entering a few keywords into an online search engine hardly 'costs' at all, be it time or any other resource. And where asking still involves admitting ignorance and imposing on somebody else's time, barriers to turning to one's web browser are non-existent. Most of the time, a software user who wants to obtain information about the program he or she is working with, is sitting at the computer; and whatever the working environment, Internet access nowadays is as good as ubiquitous.

The EPPO approach is heavily driven by this observed tendency to 'Google for the answer'. The approach, it is hoped, makes the official documentation so that it ranks high in the list of results found by search engines, and then preferred by the user over competing results. This does not mean that it is useful only for documentation that is published on the open Internet: every soft copy product can easily be searched for a particular word or phrase and supports effortless cross-referencing of topics through hyperlinks. Hard copy seems less suited, as it supports neither full-text searching nor effortless cross-referencing.



... The EVERY PAGE IS PAGE ONE pattern inevitably includes the CROSS-REFERENCE pattern and can easily be combined with a MEDIA MIX and/or the SEPARATION OF PURPOSE pattern. Include a CONCEPTUAL MODEL where appropriate to relate conceptual topics to one another...

6. Step Ladder Tutorial



Learning progresses furthest if it takes place during practice sessions that are sequenced so as to build on previous experience and previously acquired knowledge.



problem

The practice sessions offered by real-world situations are not ordered in any particular manner. A learner may therefore easily attempt to undertake work for which a large amount of learning is required all at the same time. For software with high use complexity, the load will then easily outweigh the value; causing the knowledge engine to run down.

discussion

There are two ways imaginable in which during the learning stage the practice can be broken down into smaller units, each with limited intrinsic cognitive load. In educational settings, 'part-task practice' is the traditional approach, in which component tasks are trained separately and assembled at a later stage. In contrast, 'whole-task practice' attends to the integration of constituent skills from the very beginning and ties in with the constructivist idea of embedding learning in real-life experience.

In formal training environments, the separate learning sessions in whole-task practice are sequenced so as to build on previous experience and to keep the practising learner in his or her 'zone of proximal development' (ZPD), which in a context of children's cognitive development is defined as "the distance between

the actual developmental level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance, or in collaboration with more capable peers” (L. S. Vygotsky, 1978, p. 86). In plain English: work done in the ZPD is just a little bit more taxing than that which can be done without assistance. When during the work learning takes place, the ZPD moves closer to expertise. By keeping the learner in his ZPD, a designer of instruction of any kind can push the learner gradually towards expertise, all the time managing cognitive load.

A well-known example of whole-task practice comes from driving instruction (English & Reigeluth, 1996). To set up a whole-task series of learning experiences where the learner is kept continuously in his ZPD, situate the first practice lesson not in a busy town centre during rush hour in a car with standard transmission, but in an empty parking lot in a car with automatic transmission. This is the simplest possible driving task that is still realistic. Then, lesson by lesson, confront the novice driver with ever more complex versions of the task, ensuring that every one is complete and realistic. By the time the driving task is performed in a car with manual transmission, start on the flat and add hill work only when the intricacies of clutch, gas and hand brake have been mastered.

A complete approach to the teaching of complex skills based on Cognitive Load Theory, whole-task practice, and keeping the learner in the ZPD is found in the “Four-Component Instructional Design Model” or 4C/ID model (van Merriënboer, 1997; van Merriënboer, Kirschner, & Kester, 2003). Some salient design guidelines from the 4C/ID model are:

- Design all learning tasks so that even the simplest one could be encountered in real life.
- Group the learning tasks in classes with the same complexity and present them from low to high complexity. This keeps the learner in the ZPD and reduces intrinsic cognitive load.
- Before the learner starts working on the first task in a particular class, present the necessary conceptual information. The information can then be applied to all the tasks in the class, making the practice meaningful.
- As the learner works on the tasks within a class, present the necessary procedural information; provide less of this scaffolding with each successive task within a class.
- Add part-task practice only where more repetition is required than can be offered in the whole-task sessions.

In documentation (as opposed to instruction), the designer’s problem lies not with the whole-task nature of the practice: this is built-in. Sequencing of the practice tasks, on the other hand, is very difficult to achieve. Only partial solutions will be possible.

solution	<p>Therefore:</p> <p>Add to the documentation a tutorial in which a sequence of realistic, whole-part practice sessions is presented, sequenced along the lines of the 4C/ID model.</p>
rationale	<p>A well-sequenced series of whole-task practice experiences will keep the user in his ZPD, where the ambition is continuously raised and the load is minimized, until full mastery has been achieved.</p>
consequences	<p>When it comes to sequencing the learning tasks, the Elaboration Theory of Instruction (ETI) suggests to start with the simplest version of the whole task that is still representative of the task in general (the <i>epitome</i>), then tackle progressively more complex versions of the task (<i>elaborations</i>), each slightly more complex, more divergent, more authentic and less typical (English & Reigeluth, 1996; Y. Kim & Reigeluth, 1996).</p> <p>In documentation, the epitome could be a complete task for which there is no depth of use complexity at the activity level nor at the action level. For activator software, such a real-world task may be impossible to find: in that case, accept factory defaults for those elements at the activity and action levels where there is depth and make sure to mention such choices explicitly (if not, the epitome would no longer be whole-task). Elaborations can then be defined by gradually including depth of use complexity first at the action level, and later at the activity level.</p> <p>However, a tutorial is always additional to the information need experienced by the user, and yields subjective value only if following it is a pleasurable experience in itself and/or produces immediate, real-life results.</p> <p>Therefore, when implementing this pattern:</p> <ul style="list-style-type: none"> ▪ Make sure that even the simplest of tasks that are described are realistic: explicitly mention all areas where defaults are accepted or simple conditions assumed. ▪ Use the Elaboration Theory of Instruction (ETI) to identify epitomes and elaborations, guided by the use complexity of the software. ▪ Give references to the required conceptual (read-to-know) information before any sequence of elaborations that requires them. ▪ Unless self-study is enforced (as may be the case in an organizational setting), ensure that the tutorial is not the only documentation product; or that it covers all of the software's functionality. The latter will be the case only for software where the use complexity is limited in all three dimensions.

- **Make the tutorial as attractive and easy-to-use as possible, and interrelate it to the other elements in the documentation. Use it to show the full extent of the software's functionality, raising the user's ambition.**



...The STEP LADDER is most easily implemented when presented as a COOKBOOK ...

7. Minimal Manual



When creating a documentation product that is explicitly intended to provide learning, match the way people learn by doing.



problem

People are reluctant to learn practical skills through theoretical discourse. Documentation that is not referred to cannot fuel upgrading.

discussion

The moniker “minimal manual” dates from a book written in 1990. Titled *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill* (John M. Carroll, 1990), it made a case for software instruction materials based on four principles: 1) Choose an action-oriented approach; 2) Anchor the tool in the task domain; 3) Support error recognition and recovery; and 4) Support reading to do, study and locate. The new approach (John M Carroll, 2014) condemned the then-current practice of describing the software's functionality command by command, replacing it with an active approach in which the users would follow task descriptions which placed the functionality in the context of a real-life task (principles 1 and 2). Minimal (or “minimalist”) manuals, as the end products of the approach were called, provided no more information than was absolutely necessary for carrying out the specified tasks (principle 4) and explained how to

solve any problems that were encountered (principle 3). Finally, they made careful suggestions for guided exploration (principles 1 and 4).

The minimalist approach has been shown to be an effective way of familiarizing users with the syntactic layers of software with relatively low use complexity through self-study (van der Meij, 2003; van der Meij & Carroll, 1998).

solution Therefore:

Design self-study tutorials that lead the reader through real-life tasks. Do not provide any information other than that required for the task at hand. Support error recognition and recovery, and encourage the reader to try out small variations on the solution provided.

rationale The principles of choosing an action-oriented approach and anchoring the tool in the task domain are constructivist principles, which raise ambition and lower load. The principle of supporting error recognition and recovery lowers recognition and raises both upgrading and value. The principle of supporting reading to do, study and locate raises upgrading.

consequences Minimal manuals tend to be considerably less voluminous than other documentation products, a fact which—together with the name chosen for the approach—has led some to conclude that the main maxim of minimalism is that “most manuals would benefit dramatically from a considerable reduction in size” (Mehlenbacher, 2003). This is however no more than a by-product of the approach, which from the beginning has been misinterpreted by technical communicators (Brockmann, 1998). The minimalist approach is not a panacea to cure all ills (David K Farkas & Williams, 1990). Where the use complexity is wide, a minimal manual could fast become extremely voluminous, unless it settles for incompleteness (Draper, 1998).

The emphasis on procedural information means that a minimalist documentation product has trouble counteracting any depth in the use complexity. All four minimalist principles depend heavily on the writer knowing what is correct reader behaviour and what is not: the approach relies on there being user tasks that are a priori known. Their rejection of conceptual information places serious obstacles in the way of its being appropriate for teaching complex, ill-defined decision-making tasks (Mirel, 1998b) or for satisfying the needs of advanced users (Hackos, 1998). Almost inevitably, a minimal manual for software with high use complexity will clash with the constructivist guideline to not over-simplify.

A minimal manual is a tutorial document, intended to replace the documentation journey with one or more self-study sessions. When the approach was first developed, computers were new, seen as ‘difficult’, and encountered only infrequently. Sitting down to learn before doing any real work would have been feasible as well as reasonable. Taking a training course before attempting real-life

work with software was standard practice, and self-study tutorials were often the only documentation product provided (van der Meij et al., 2009). Nowadays a person encounters new software wherever he goes. This software has often much higher use complexity than that for which the minimalist approach was developed, and learning-before-doing is a luxury which time may no longer permit. As has been pointed out earlier, a tutorial is always additional to the information need experienced by the user, and yields subjective value only if following it is a pleasurable experience in itself and/or produces immediate, real-life results.

Therefore, when implementing this pattern:

- **Unless self-study is enforced (as may be the case in an organizational setting), ensure that the tutorial is not the only documentation product; or that it covers all of the software's functionality. The latter will be the case only for software where the use complexity is limited in all three dimensions.**
- **Make the tutorial as attractive and easy-to-use as possible, and interrelate it to the other elements in the documentation. Use it to show the full extent of the software's functionality, raising the user's ambition.**



... A MINIMAL MANUAL together with a set of JOB AIDS would constitute a complete documentation set for operator or actor software with limited depth and width...

8. Cookbook



A Cookbook is a collection of solutions to real-life problems in a particular design domain, presented in such a manner that they can be easily included in the reader's own work.



problem **If multiple bits of information must be applied all at the same time, a vast amount of reading and learning will be required before the user can see 'what it all means' in a real-world context. The reader will not know where to begin and is likely to give up rather than to upgrade.**

discussion A particular class of software tools is formed by those tools that enable exploratory design, to be used at a later time by the user himself or by third parties. Such software has by definition high use complexity, and support an activity in which even relatively simple results may require a vast amount of knowledge.

The Cookbook pattern is encountered frequently where the primary audience consists of programmers, or of end-users attempting to modify the workings of their chosen software tool. To describe what a Cookbook is, I can do no better than quote from the introductory pages of the commercially available *Access Cookbook*³⁴:

³⁴ By Ken Getz, Paul Litwin & Andy Baron. O'Reilly, Sebastopol CA, USA, 2002, ISBN 0-596-00084-7

This is an idea book. It's a compendium of solutions and suggestions devoted to making your work with Microsoft Access more productive. [...] [It] offers you solutions to problems you may have already encountered, have yet to encounter, or perhaps have never even considered. [...] our goal is to show you how to push the edges of the product, making it do things you might not even have thought possible.

Like a cookbook in the original sense of the word, a Cookbook does not explain first principles. The reader of the *Access Cookbook* is expected to know how to create a query or how to switch to Design Mode in exactly the same manner that the reader of *The River Cook Book Green*³⁵ is expected to know how to turn on the oven or how to chop up a bunch of parsley. Also like a cookbook containing instructions for preparing food dishes, the separate ‘recipes’ are bundled roughly by area of interest, for the reader to browse (see Figure 40) until he sees one that he fancies.

Chapter 2. Forms	93
Section 2.1. Make Custom Templates for Forms and Reports	93
Section 2.2. Highlight the Current Field in Data-Entry Forms	97
Section 2.3. Restrict the User to a Single Row on a Form	101
Section 2.4. Use an Option Group to Collect and Display Textual Information	103
Section 2.5. Display Multiple Pages of Information on One Form	106
Section 2.6. Provide Record Navigation Buttons on a Form	110
Section 2.7. Size a Form's Controls to Match the Form's Size	116
Section 2.8. Make a Simple "Searching" List Box	121
Section 2.9. Create a Replacement for Access's InputBox	125
Section 2.10. Store the Sizes and Locations of Forms	131
Section 2.11. Open Multiple Instances of a Form	135
Chapter 3. Reports	141
Section 3.1. Create a Report with Line Numbers	142
Section 3.2. Print the Value of a Parameter on a Report	143
Section 3.3. Create a Report with Multiple Columns	147
Section 3.4. Print a Message on a Report if Certain Conditions Are Met	152
Section 3.5. Create a Page-Range Indicator on Each Page	155

Figure 40: Part of the Table of Contents of the *Access Cookbook*

All instructions are discussed and elaborated upon, giving the underlying rationale to every step and inviting the reader to customize the solution to his particular situation. When applicable, bits of code are included that can be copied into the user's own working environment.

Create a collection of complete solutions to real-world design problems and present them in such a manner that they can be applied with little effort. Walk the user through the solutions and explain the rationale behind every aspect of them.

³⁵ By Rose Gray and Ruth Rogers. Ebury Press, London, UK, 2000. ISBN 0-09-187943-4

rationale

Since the examples are solutions to real-life problems, a Cookbook embeds the learning experience in genuine practice, with learning as a by-product in a true constructivist manner.

As the quote from the *Access Cookbook* on the previous page shows, the intention of a Cookbook is explicitly to widen the user's intention horizon. The solutions contained in it work directly on the ambition. They then provide the scaffolding needed for a user to fulfil his heightened ambition; and they work on value by allowing for a more satisfying end result and on load by removing part of the work. Customization provides germane cognitive load during situation processing, resulting in learning.

consequences

If the solutions can be copied without any customization, then they will not stimulate upgrading but rather detract from situation processing. In addition, when customization is difficult, the load will increase; and when the reader picks the wrong solution or the wrong customization, the value will decrease.

Therefore, when implementing this pattern:

- **Provide complete rather than partial solutions, where different parts of the software are combined to achieve the result.**
- **Format so that invariable elements stand out from variable elements, to facilitate customization.**
- **If the solutions consist of code, consider a further reduction of the load by presenting them in soft copy to allow for copying and pasting.**



... The solutions given in a Cookbook can be seen as elaborate JOB AIDS for activator software. By ordering the solutions so that the results of one solution are the starting point for the next, a STEP LADDER can be implemented. When the Cookbook is for a programming environment where code is written, the load can be lowered further by including the solutions partially or entirely in the soft copy component of a MEDIA MIX ...

9. *Micro Patterns*

'Micro' patterns are applied to components of a documentation product, yielding something that cannot stand on its own.

9. *Job Aids*



Job aids are cognitive tools that are embedded in the work environment to immediately facilitate performance. As they are never accessed away from the work environment, the act of referring to them is not experienced as 'reading' but rather as part and parcel of doing the job (which, as a consequence of the job aid, becomes easier).



problem **When there is exactly one 'correct' way to carry out a task, and this involves multiple interactions, requiring the user to memorize the steps would add to the load without having any benefits.**

discussion Where the use complexity is wide, the user is required to select a particular interaction from a large number that are visible to him. Whenever a whole sequence of such decisions is dependent on one well-defined objective, internalizing the steps would constitute extraneous cognitive load that is not offset by any germane cognitive load. The load can then be off-loaded onto cognitive artefacts that function as external memory.

An interest in embedded information products has developed independently from that in user manuals or instructional design theory. Such cognitive artefacts are

referred to as ‘performance support systems’ (Nguyen & Woll, 2006), ‘job performance aids’ (C. P. Campbell, 1998) or simply ‘job aids’ (Bullock, 1982; Rossett & Gautier-Downes, 1991). Job aids enable the reader to successfully carry out activities that without them would be difficult, or even beyond his reach. They provide straightforward information ‘just-in-time’, at the exact moment that the information is required, thereby reducing the amount of information that has to be remembered (C. P. Campbell, 1998).

Job aids can be delivered through print, soft copy, audio or video. When print is used, delivery is not restricted to traditional book or poster formats but may be “in many sizes on many surfaces, such as plastic, adhesive tape, cardboard, metal, laminated materials and so on” (Rossett & Gautier-Downes, 1991, p. 78).

Most office workers will be familiar with finding instructions for troubleshooting a photocopier: in a booklet that fits in a sleeve on the control panel, or in a pictorial guide integrated into the control panel, or in a series of animated images on a display that show the next step to be performed at any stage in the troubleshooting procedure. These are all examples of job aids. Countless further examples can be found in everyday life. Stepwise instructions are found printed on or attached to packaging; detailing assembly, use, storage or maintenance. We find heating instructions on the packaging of pre-cooked meals, safety considerations on a sheet in the pocket of aircraft seats (or on the headrest cover of those seats, for the benefit of the person seated in the row behind), and instructions on how to brush teeth on the blister packaging of a new toothbrush. A rose bush bought at a garden centre may have a label attached to it specifying recommended planting depth, expected height when fully grown, pruning instructions, and the month in which flowers can be expected. A first-aid kit may contain a wallet-sized card outlining what to do in which type of emergency, origami paper may show folding lines, and the packing slip included with a home-delivered order from an online shop may be found to double as a return form when folded in a particular manner as indicated on the form itself. As I was writing this, a letter from the Town Council arrived, advising of an upcoming change in the way household refuse is collected in our neighbourhood. In addition to the letter, the envelope contained a sticky label to be attached to my old wheelie bin before it is left at the roadside for collection; this stated not only identifying information for the benefit of the Council but also re-iterated where to leave the bin out, when to leave it out, in which state to leave it out, and what to do in case of non-collection. Job aids are everywhere.



Figure 41: Job aid for bird-watchers³⁶

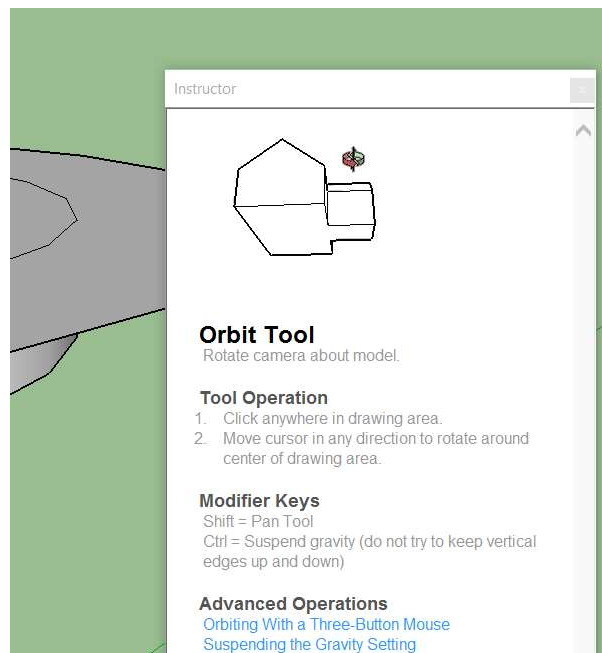


Figure 42: Job aid for software operations (Trimble SketchUp)³⁷—the illustration above the text is animated, and the appropriate topic is shown automatically when the user selects a different tool icon.

³⁶ The BirdSong iFlyer Scanning Wand: found on 6 October 2015 at <http://www.birdhousesbymark.com/iflyer.shtml>

³⁷ <http://www.sketchup.com/>

Job aids have a long tradition in process control situations. Most often encountered are stepwise instructions, reference sheets, work sheets (fill-in forms), checklists, and decision guides such as flowcharts and decision tables (Bullock, 1982; C. P. Campbell, 1998). But this is not an exhaustive list: what makes a certain part of the documentation a job aid, is that 1) it is present or can be made to be present at the exact time and place where it is needed, and 2) it immediately helps performance. For example, a fill-in form can be taken onto the shop floor to collect data relevant to the task at hand. A work sheet can model process and product, by specifying the steps to be performed during performance and forcing the user to keep track of decisions that were made. A flow chart can guide decisions where there is width of use complexity at the level of operations or at the action level.

In software environments, job aids can be included in the documentation, but tighter integration with the working environment is achieved through tooltips (Nguyen & Woll, 2006) or on-screen instructions (see Figure 37 and Figure 42).

solution Therefore:

Provide job aids where the cognitive load involved in internalizing the knowledge would be extraneous rather than germane. Link the job aid as tightly as possible to the work environment: make it printable or easily portable when the job is carried out away from the computer.

rationale Job aids remove extraneous cognitive load.

consequences Job aids play along with the production bias and are for that reason tempting to the user. Great care must however be taken that only extraneous cognitive load is removed by a job aid. If the cognitive load involved in situation processing is germane, then it should *not* be reduced: this limits the use of job aids to well-defined tasks where there is only one 'correct' way of carrying out the job.

The use of job aids may have effects other than reducing cognitive load. Once filled in, for example, a work sheet can be a useful tool for tracing wrong turns taken when an error is detected. The work sheet then supports error correction and thus learning, and induces self-reflection and formative assessment. Checklists can model process as well as provide formative assessment leading to error correction. As they list things to consider rather than actions to carry out, checklists directly scaffold the active construction of knowledge, at any level of expertise.

Therefore, when implementing this pattern:

Do not provide job aids where their use would reduce germane cognitive load. Consider additional benefits of every job aid and design for them wherever possible. When creating a decision support tool such as a flowchart or checklist, formulate the items so that they invite close consideration and so that they cover every eventuality.



... JOB AIDS are easily included in a MEDIA MIX, an EVERY PAGE IS PAGE ONE product, or the SEPARATION OF PURPOSE pattern...

10. Screen Capture



Screen captures (also known as screenshots, screen images, screen grabs or screen dumps) are representations of the screen, which help the reader place the information in the context of the work with the software. Produced directly from the memory on the graphics card, they present a 'true' facsimile.



problem **Inappropriate recognition may take place when it is not clear to the reader which particular state of the software a piece of information applies to; but it is difficult to describe the software in words only.**

discussion Screen captures show exactly what the screen should look like for the verbal description to apply. A screen capture may show the whole screen or one particular window (as in Figure 43), or only part of the screen or window. It may include highlighted, greyed-out or selected screen areas as well as any expanded menus or drop-down lists, the cursor, and the mouse pointer. A screen capture can be annotated with graphical markers and/or text labels, which in turn may be combined into a legend.

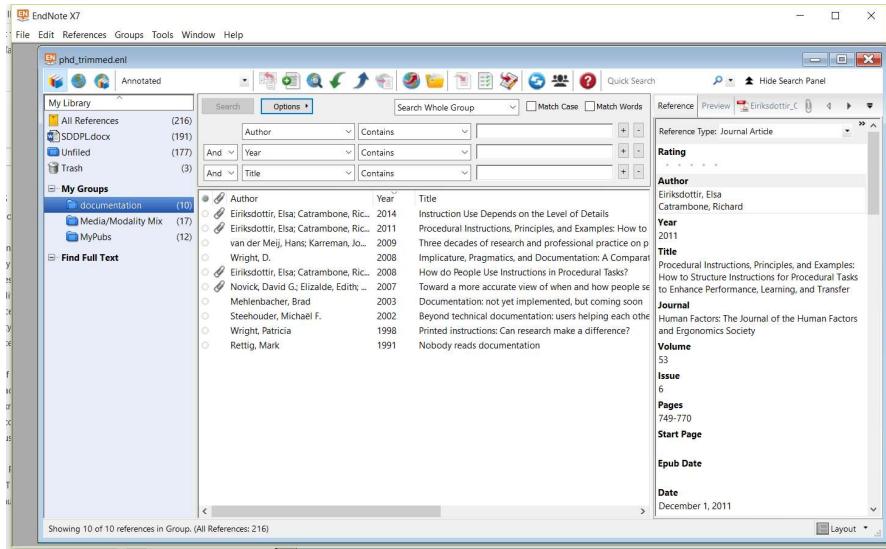


Figure 43: Screen capture showing a complete screen³⁸.

Van der Meij and Gellevij (1998) distinguish four “roles” for screen captures. First, 1) they “support the user’s switching attention between input device, manual and screen”. In terms of the CMA model, this role concerns the documentation journey: a screen capture ties the documentation journey to the work with the software. Then, 2) they “support the user’s developing a mental model of the program”. The authors are however not clear on the mechanism by which a screen capture fulfils this role, unless we define a mental model as consisting of the user interface of a program only. Thirdly, 3) they “help the user verifying screen states”; and finally, 4) they “help the user identify and locate window elements and objects”. These last two roles are related to perception and recognition.

Looking at the way screen captures are used in practice will show the list to be non-exhaustive, as even a limited number of examples will show.

³⁸ EndNote X7, ©1988-2016, Thomas Reuters

Showing a complete window as in Figure 43 helps recognition of an overall state of the software. The screen captures in Figure 44 and Figure 45, in contrast, remove the need for the user to search for the screen element under discussion.

Unsaved Changes

When a document contains unsaved changes, an asterisk (*) appears next to its tabbed name. The asterisk disappears once the unsaved changes have been saved.



Figure 44: Screen capture cut out to show only the screen area under discussion³⁹.

Located directly under the Commands Bar is the **Tool Controls Bar**.



Figure 45: Screen capture annotated with highlighting to show location⁴⁰.

In Figure 46, a screen capture is used in a procedure to show the desired result of step 1 and the starting point for step 2. This helps the user recognizing if his situation processing has resulted in success.

³⁹ Surfer 12 Full User's Guide p.13, © 2014 Golden Software, Inc

⁴⁰ InkScape Manual p.4, undated, distributed under the GNU General Public License, accessed at <https://ma.ellak.gr/documents/2015/07/inkscape-manual.pdf> on 11 December 2016

Modifying the edge color

1. Select the node which you want to modify. Click **Format** and choose **Edge Color**.

The Choose Edge Color dialogue box opens.

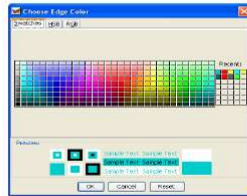


Figure 6 The Choose Edge Color Dialogue Box

2. Select the edge color of your choice. Click **OK**.

Figure 46: Screen capture to show the desired result of one interaction and the starting point for another⁴¹.

Thus far, all the examples have been of screen captures used to enhance perception and recognition. Let us now look at a few screen captures that do a different job.

The screen capture in Figure 47 looks similar to the one in Figure 46, but works in a different manner. Not linked to particular steps in the procedure and located alongside rather than inline with the steps, it does little to support recognition but rather reinforces the text by making the available choices visible. (Note that in this example, conceptual or read-to-know information is provided together with procedural or read-to-act information.)

⁴¹ Freemind Version 0.8.0 User Guide p.25, undated, distributed under the GNU Free Documentation License, accessed at [http://freemind.sourceforge.net/FreeMind%20User%20Guide%20by%20Shailaja%20Kumar%20\(mannual\).pdf](http://freemind.sourceforge.net/FreeMind%20User%20Guide%20by%20Shailaja%20Kumar%20(mannual).pdf) on 11 December 2016

Creating a New File and Adding Layers

You can create a new file in several ways.

1. Create a new, empty file from the File > New menu. The New Image dialog is presented; see the image to the right. From the New Image dialog, you have the following options.
 - ◊ **Name:** Presets the name of the image.
 - ◊ **Preset:** Lists a range of preset image dimensions for common print sizes.
 - ◊ **Width:** Sets the width. There is a drop-down menu that includes inches, cm and pixels.
 - ◊ **Height:** The same as the width, just for setting the height.
 - ◊ **Swap Dimensions button:** Pressing this button will swap the current width and height.
 - ◊ **Resolution:** Sets the resolution of the image.
 - ◊ **Color Space:** Sets the color space (icc profile) of the image. The default is set based on the user preferences. Only RGB color space profiles are available.
 - ◊ **Bit Depth:** Sets the bit depth of the image, either 8-bit or 16-bit.
1. Merge one or more images together using the Add Layer from File command.
2. Merge one or more images together using the Lightroom or Aperture plug-ins or external editor option.
3. Dragging one or more images onto the Perfect Layers icon.

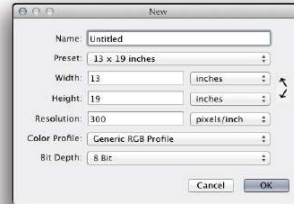


Figure 47: Screen capture to complement an overview⁴².

A screen capture can also stand for a complete series of interactions, as in Figure 48. Here, the documentation explains not how to do something, but why to do it; and the screen capture is part of an abstract image. Again, the purpose of the screen capture is not so much to support recognition as to provide conceptual (read-to-know) information.

Gears

The Gears effect is a toy effect. It generates a chain of interconnected gears from the path that has the effect applied to it. The nodes of the path define the centers of the gears. The first 3 nodes are special; the first defines the start angle of the chain, the second defines the center of the first gear and the third knot specifies the radius of the first gear. That is, to create a chain of 2 gears, you will need a path with 4 nodes; for 3 gears, 5 nodes, and so on.

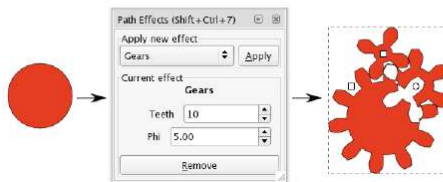


Figure 48: Screen capture to represent a complete series of interactions⁴³.

The screen capture in Figure 49, finally, enhances the verbal conceptual information with an example. Here, it links the action to the result and provides read-to-consider information:

⁴² Perfect Photo View User Manual p.70, ©2015 on1, Inc.

⁴³ InkScape Manual p.98, undated, distributed under the GNU General Public License, accessed at <https://ma.ellak.gr/documents/2015/07/inkscape-manual.pdf> on 11 December 2016



Figure 49: Screen capture to add information not explicitly mentioned in the text⁴⁴.

This limited number of examples shows that screen captures can be used not only to support perception and recognition but may do many, subtly different, jobs in user documentation.

Therefore:

Use screen captures to show what the user's software environment 'should look like' for the adjacent information to be relevant. Design them so as to do as much additional work as possible and ensure that they have no undesired side effects.

A screen capture links the text to a particular state of the software, allowing the user to verify that the text he is referring to is indeed appropriate in the current context. It thus reduces the load involved in the documentation journey and encourages the user to stick with the documentation. Depending on how a screen capture is implemented, it can do additional work. In the images, we have seen that screen captures may:

- help shape appropriate recognition, by showing what the screen should look like after one step and before the next one (Figure 46);
- help raise ambition, by showing what options are open to the user (Figure 47), by providing directly applicable conceptual information (Figure 48), or by working together with the text to create an example (Figure 49);

⁴⁴ InkScape Manual p.69, undated, distributed under the GNU General Public License, accessed at <https://ma.ellak.gr/documents/2015/07/inkscape-manual.pdf> on 11 December 2016

- help lower the load in the interaction with the software itself, by removing the need to search for the appropriate screen element (Figure 44 and Figure 45).

consequences Screen captures make a text considerably longer as well as more difficult to scan. This could easily add extraneous load. In soft copy, screen captures can confuse the reader, as the difference between the screen capture and the real software environment is not immediately obvious.

Therefore, when implementing this pattern:

- **Do not use more than you need and use them only where they add information. Take care to make every screen capture more informative than the screen itself by using cut-outs or annotations, and/or by setting the screen up before you capture it so that the mouse pointer highlights salient areas.**
- **Be careful when using screen captures in soft copy, especially in context-sensitive Help: use them only when there is no other solution, do not show a complete screen or window but use only cut-outs, and use colouring, shading, and annotations to ensure that the screen capture cannot be confused with the real software environment.**



... When your MEDIA MIX includes hard copy, a SCREEN CAPTURE helps tie the documentation to the reader's interaction with the software. SCREEN CAPTURES help provide the situational information required to assess the applicability of STEPWISE INSTRUCTIONS ...

11. Conceptual Model



A conceptual model uses words and/or diagrams to show the main conceptual parts of a system, together with the way the parts relate to each other.



problem

Read-to-know information (see page 120) is required for successful upgrading but often fails to engage (Figure 33 is fairly typical in this respect). Also, new concepts in the software may be interconnected and mutually dependant, so that describing them in isolation does not help the reader construct a valid UVM.

discussion

Read-to-know information is called for where there is depth in the use complexity. It does however cause the reader to experience load. When the depth corresponds to interconnected features in the User Virtual Machine, the user will find in every read-to-know block describing one concept references to others which are not yet known. This will cause the reader to experience even more load, with little value to counterbalance. If the use complexity is low, this may not be much of a problem; but if it is high, then it probably is.

It has been proposed (Norman, 1987) that a correct and complete mental model is more easily constructed if instruction is provided in the form of a conceptual model. This premise has been studied by a number of authors, who tried to establish the effect on learning and performance of presenting a conceptual model. Unfortunately, the designs of these studies vary widely, making comparison rather difficult. Sometimes, the effects of different types of conceptual model were compared; without a control group being exposed to no conceptual model at all (Sein & Bostrom, 1989; Shayo & Olfman, 1998). At other times, the target system

under consideration was not a software tool to be used to an end, but custom-built to simulate a physical system: so that effectively, the target system itself constituted a model of reality (Fein, Olson, & Olson, 1993; D. E. Kieras & Bovair, 1984).

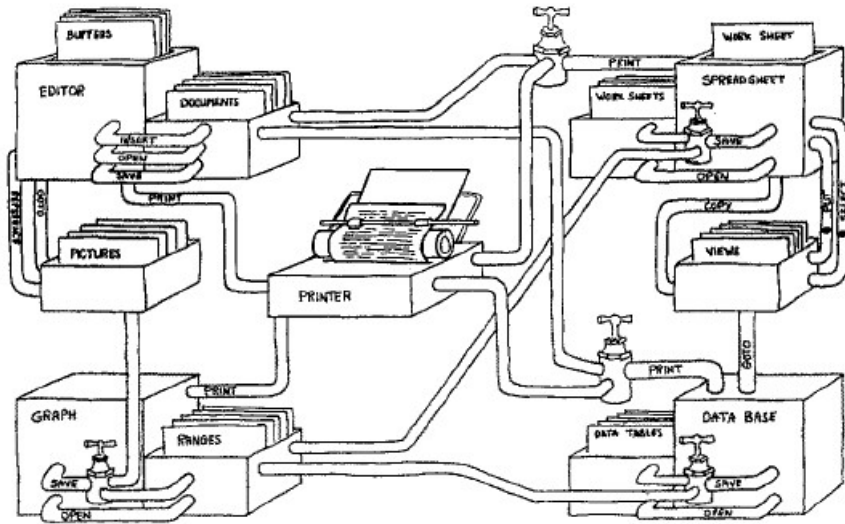
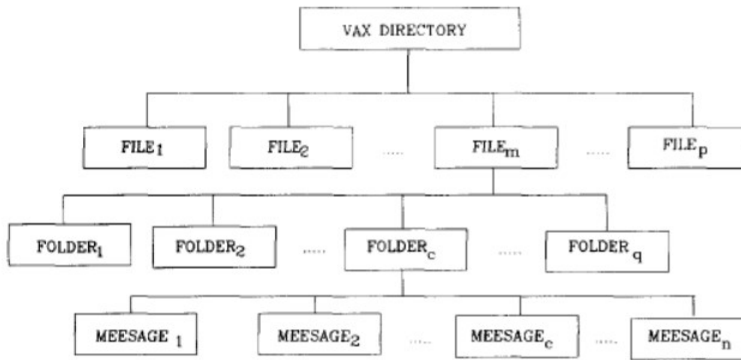


Figure 50: Analogical conceptual model illustrating an office suite (from: van der Veer & Felt, 1988)



Note: FILE_m is the MAIL file.

*Figure 51: Abstract conceptual model illustrating the VAX Mail system
(from: Sein & Bostrom, 1989)*

Even when the effect of a conceptual model on mastering software is studied, study designs vary. Table 4 attempts to normalize the findings of a number of such research studies, presenting every finding in a separate row so as to make comparison possible. It is important to realize that concepts and approaches that are currently familiar (such as GUIs and word processors), were not necessarily so at the time a particular study was carried out. As computers began to play a role of ever increasing importance, conceptual model studies targeted the period's most visible computer application: ranging from the BASIC programming language (Bayman & Mayer, 1988) through the VAX mail system (Santhanam & Sein, 1994) to WWW search engines (Colaric, 2003) and a sophisticated word processor (Ben-Ari & Yeshno, 2006). My assessment of the depth of the use complexity of the target system (or rather, that part of the target system under consideration in the studies' post-tests) is based on the temporal context of the study.

*Table 4: Summary of some research studies on
the effectiveness of conceptual models in
instruction (in chronological order)*

	post-test	depth	height	model	effect
RPN-based calculator (F.	reproductive	deep	task layer	abstract: text and diagram	none

	post-test	depth	height	model	effect
G. Halasz & Moran, 1983)	constructive (problem-solving)	deep	task layer	abstract: text and diagram	positive
programming language BASIC (Bayman & Mayer, 1988)	constructive (programming)	deep	task layer	abstract: text and/or diagram	positive (for low-ability subjects)
	knowledge	deep	syntactic layer	abstract: text and/or diagram	positive (for low-ability subjects)
	constructive (programming)	deep	task layer	abstract: text and/or diagram	negative (for high-ability subjects)
	knowledge	deep	syntactic layer	abstract: text and/or diagram	none (for high-ability subjects)
integrated office system (van der Veer & Felt, 1988)	knowledge	deep	task layer	analogical : diagram	positive
VAX Mail (Santhanam & Sein, 1994)	constructive (near and far transfer)	shallow	task layer	abstract and analogical	none
	knowledge	shallow	task layer	abstract and analogical	none
MS Word (Ben-Ari &	constructive	deep	task layer	abstract: text	positive

	post-test	depth	height	model	effect
Yeshno, 2006; Yeshno & Ben-Ari, 2001)	knowledge	deep	task layer	abstract: text	none
Web search engine (Colaric, 2003)	knowledge	deep	task layer	abstract: text with or without diagram	none (as compared to instruction by example)

In addition to a rough assessment of the depth of the use complexity of that bit of the target system relevant to the post-test, Table 4 lists the nature of the post-test (constructive or reproductive performance, or knowledge elicited from the respondents in some way other than through observing interaction with the target system); the layer of the software involved in the post-test; and the effect of providing the respondents with a conceptual model during the training phase.

The data presented here suggests that, as predicted, performance increases after instruction in which some sort of a conceptual model is presented when the nature of the task that is post-tested is constructive and the task layer involved is higher than the syntactic layer. In one study (Bayman & Mayer, 1988), this effect was found to hold only for low-ability students (as determined by their scores on the SAT test, a measure of general intelligence): the authors hypothesize that high-ability students do not need the scaffolding offered by the model, so that for them the model presents extraneous rather than germane load. Interestingly, the data also suggests that a conceptual model does *not* help learners to score significantly better when it comes to knowledge that can be elicited away from hands-on practice.

solution Therefore:

When describing interconnected concepts that are mutually dependent and that do not map directly on pre-known concepts in the outside world, include a conceptual model to show how components of (part of) the UVM relate to each other.

rationale

A conceptual model explicitly encourages upgrading. By making the information more engaging, it lowers the subjective load. By placing the separate read-to-know blocks in an interrelated context, it lowers the load even further, and adds to the value. As the model already places the subsequent learning in a ready-made structure, the knowledge resulting from situation processing will be more easily

assimilated into the knowledge base. Finally, the model may dampen subsequent recognition because it makes the interrelatedness of concepts in the UVM explicit.

consequences However, studying a conceptual model in itself may add to the load. An explicit conceptual model may be text-based or diagrammatical. Hegarty and Just conducted a number of experiments, studying the construction of a correct mental model of a mechanical system (a system consisting of pulleys) from text and/or diagrams (Hegarty & Just, 1993). Their results strongly suggested that readers learn more from a text-and-diagram description than from either text or diagrams alone. Diagram inspection was shown to be text-directed, and the authors propose that the diagram acts as an external memory aid, primarily when visualizing the dynamics of the system. The text, in turn, is required to stimulate readers to carry out this visualization: without it, they are less prone to do so. When it comes to generalization of the results to other domains, however, the authors point out that readers may have more difficulty integrating information from text and diagrams into a coherent mental model when the diagram cannot directly depict the system under consideration (as is the case when we provide a representation of a User Virtual Machine).

Another consideration concerns the nature of the model, rather than the way in which it is presented. Such a model may be *abstract* (that is, not referring to objects in the pre-known world) or *analogical* (that is, presenting a coherent analogy). A problem with analogical models of software systems is that the analogy will inevitably run out before it has done much explaining. Conversely, there are always bits of the analogy that do not apply. So “To make effective use of analogical models, the new user is faced with the confusing task of sorting out the relevant inferences from among the many possible irrelevant or incorrect inferences suggested by the analogy” (F. Halasz & Moran, 1982, p. 384). A convincing argument can be presented for conceptual models to be presented in abstract, rather than analogical form; because “Computer systems are unique. The tasks they carry out may often be familiar, but their underlying conceptual structures are not” (F. Halasz & Moran, 1982, p. 384). Halasz and Moran do see, however, a role for analogy not in the form of complete models (“the computer is like a typewriter”) but as literary devices when explaining an isolated concept (“the computer’s keyboard is used to type letters, just like a typewriter keyboard”).

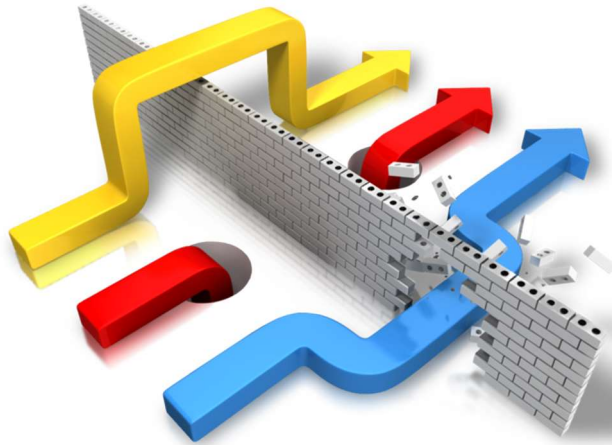
Therefore, when implementing this pattern:

- **Use a diagram together with text and ensure that both are easy to relate to the visible reality on the screen.**
- **Avoid drawn-out analogy.**



... A CONCEPTUAL MODEL is especially useful when using the SEPARATION OF PURPOSE pattern and/or the EVERY PAGE IS PAGE ONE pattern, where you may struggle to present interrelated blocks of read-to-know-information in a coherent manner...

12. Cross-Reference



Cross-references explicitly invite the reader to access a non-adjacent part of the documentation and allow him to do so easily.



problem

The order in which information is presented does not necessarily correspond to the order in which it is needed by a particular reader in a particular context. For the reader to work out which other bits of the documentation to access next and where to find them constitutes considerable extraneous load.

discussion

Every information product, and therefore every documentation product, has a sequential access order built in, in accordance with the language in which it is written: left-to-right (for English and many other European languages) or right-to-left (for, amongst others, Hebrew and Arabic). The separate pages of the information product are read from top to bottom and follow upon one another (Belew, 2000, p. 18). Although an author will always do his best to make the sequence logical and useful, there will always be situations in which the sequence does not match the information need of the reader. The documentation journey then breaks down.

In hard copy, cross-references may point to a page number and/or to a section. In soft copy, they are usually hyperlinks: the reader is directly taken to the referenced location when he clicks on the cross-reference. The cross-reference itself then no longer needs to explicitly mention the location within the document of the remote information.

- From the drop-down list, select the **Preferred unit system** (metric or US). This is the system in which all [data classes and types](#) are expressed by default.

Figure 52: Cross-reference in the online Help system for a seismic exploration tool⁴⁵

- 1 Insert an anchored frame (see “Creating anchored frames” on page 346), and draw a text frame in it. For information on drawing a text frame, see “Using text with graphics” on page 323.
- 2 Insert the table in the text frame. For details, see “Inserting tables” on page 158.

Figure 53: Cross-references in the printed user manual for a document development tool⁴⁶

The examples above show in-line cross-references; cross-references can also be collected in a separate read-to-use block, providing an overview of related sources of information that are relevant to the current topic. Such an overview may, but need not, be categorized.

Related Topics

- [Floating and Docking](#)
- [Working with Projects](#)
- [Using the Help System](#)
- [Jason Installation Guide](#) (including hardware and software requirements)
- [Known Limitations](#)

Figure 54: An uncategorized cross-reference list in the online Help system for a seismic exploration tool⁴⁷

⁴⁵ Jason Workbench 9.1, © 2015 CGG

⁴⁶ Adobe FrameMaker 8 User Guide, © 2007 Adobe Systems Incorporated, p. 176

⁴⁷ Jason Workbench 9.1, © 2015 CGG

The Byte Ryte Bookshelf is a hand-picked list of publicly accessible publications related to what we do and what interests us. Here you can find complete courses and books covering: technical writing, software documentation, knowledge management, layout, markup, e-publishing, information design, and more.



[Basic Technical Writing Skills](#) | [Graphic Design and Layout](#) | [Electronic Publishing](#) | [Information Design](#) | [Usability and User Interface Design](#) | [Tooling and Scripting](#) | [XML and Markup](#) | [Language and Literature](#) | [On-line Reference Works](#) | [Teaching and Training](#) | [Miscellaneous](#)

Basic Technical Writing Skills

- ▶ A complete course in technical writing:
[Online Technical Writing Textbook](#)
- ▶ A colleague in the US also placed a "manual on writing manuals" online:
the [Dozuki Tech Writing Handbook](#)
- ▶ The US Securities and Exchange Commission published an Adobe Acrobat (PDF) file that contains an updated [Plain English Handbook](#)
- ▶ Two "style guides", to be used as they are or as a starting point when developing your own:
the [Apple Publications Style Guide](#) (for software-related publications) and
the [Modern Humanities Research Association](#) style guide (for factual and academic publications)

Figure 55: A categorized cross-reference list from the author's website⁴⁸

solution

Therefore:

Use cross-references to direct the reader to relevant, non-adjacent pieces of information, in the same or in a different documentation product.

rationale

Cross-references direct the documentation journey by bringing the user to exactly that information that he needs. This removes extraneous cognitive load from the documentation journey.

consequences

As we have seen earlier (page 113), in a soft copy product that is presented in the form of hypertext, the risk for the reader to get 'lost in hyperspace' is proportional to the number of cross-references (hyperlinks). The risk is more limited in hard copy, where following a cross-reference requires more effort and is therefore perhaps less easily undertaken. In both hard copy and soft copy however, the reader's following a cross-reference means him leaving the default reading order.

⁴⁸ <http://www.byteryte.nl/en/bookshelf.html>, screen grabbed on 8 December 2016

Another consequence has to do with the ‘information scent’ of the cross-reference, which is carried by the way it is phrased. A simple reference to ‘page 38’ gives no suggestion of what is to be found on that page, so that the information seeker cannot judge whether he should follow the cross-reference or not. He might follow cross-references that are irrelevant, resulting in a suboptimal documentation journey bringing relatively high load, or he might neglect to follow cross-references that enhance the documentation journey and result in increased value.

Therefore, when implementing this pattern:

- **Deploy cross-references only where it is imaginable that an objective need for the remote information could exist, in a particular working context and/or for a particular type of reader.**
- **Phrase the cross-references so that the reader is helped to recognize whether there is a need for the remote information or not.**



... In many documentation products, CROSS-REFERENCES can optionally be deployed to enhance the documentation journey. In the EVERY PAGE IS PAGE ONE and SEPARATION OF PURPOSE patterns, cross-references are indispensable ...

13. Chapter Summaries & Chapter Introductions



Chapters within a documentation product can be preceded with an introduction or concluded with a summary. The introduction tells the reader what to expect; the summary tells him what he has learned.



problem	Reading irrelevant bits of information constitutes load, while skipping relevant bits hinders upgrading. A chapter's title is usually too short to let the reader know if he should read it.
discussion	<p>Readers of documentation need help determining what to read and what to remember. Chapter introductions and summaries provide this help, by describing the chapter's contents in a few sentences.</p> <p>People will invest time and effort in the documentation journey only when they feel there is value to outweigh the load. Therefore, they need more than a chapter title in order to determine whether to read a chapter, or bits of it; and to determine which bits to read. All six participants to the 'focus group' held on 9 July 2015 during the EuroPLOP 2015 conference in Kaufbeuren, Germany (described in <i>Appendix 3. Using the Repertory Grid Technique for Mining Design Patterns</i>) indicated to read chapter summaries to this end.</p> <p>In addition, a chapter introduction can be designed to be an "advance organizer". This is a brief narrative or diagram which is presented before the lesson itself and which introduces the content of the lesson at a higher level of abstraction, generality, and inclusiveness than the content itself. The advance organizer provides a simple framework into which further learning can be assimilated and which relates that which a student already knows to that which is being learned. In formal learning situations, an advance organizer makes for better understanding as well as retention of the lesson (Ausubel, 1968).</p> <p>Similar to a chapter introduction is a chapter summary. This is placed at the end of the chapter and relates its content to the reader's real-life tasks and to other content in the document or the documentation set; often, the next chapter. It is used most regularly in long chapters, to reinforce retention. The participants to the above-mentioned EuroPLOP 2015 focus group all mentioned reading chapter summaries to make sure that they had not, based on the chapter introduction, skipped any useful information. This suggests that it is worth writing the summary from a different point of view than the introduction, giving the reader two different ways of evaluating the chapter's content without reading it from beginning to end.</p>
solution	<p>Therefore:</p> <p>Start every chapter with a few paragraphs outlining the chapter's content. When the chapter contains much information, also conclude it with an overview of what the reader should have learned by reading the chapter, and where to find related information.</p>
rationale	Chapter introductions and summaries make it easier for the reader to skip information that is not interesting at a given moment and to access those that are.

Thus, they remove load from the documentation journey and improve the chances of upgrading. At the same time, the reader will obtain an idea of what is in those bits of the documentation that he decides to skip; and this may help keeping the ambition up. When a different information need is present at some other time, this will more easily become manifest when the reader remembers having seen 'something'; so that he can decide to go back to a chapter.

consequences Short chapters can be skimmed quickly enough and don't need an introduction or summary. To the contrary, this would add more load than it removes. Following on from this consideration, participants to the EuroPloP focus group felt that there is no place for introductions or summaries in soft copy, where "chapters are always very short" and an introduction or summary would be "an annoying waste of space".

Therefore, when implementing this pattern:

- **Ensure that the length of the introduction and summary taken together is in proportion to the length of the chapter and the amount of information it contains.**
- **In soft copy, turn introductions and summaries into separate topics.**



... In many documentation products, CHAPTER INTRODUCTIONS AND SUMMARIES can optionally be deployed to enhance the documentation journey. They can be very easily incorporated in the EVERY PAGE IS PAGE ONE pattern ...

14. Stepwise Instructions



Tasks with well-defined starting and end points are easily presented as a sequence of steps.



problem **When the software calls for a fixed sequence of interactions and the separate steps are not clearly described, users will experience considerable load determining how to act.**

discussion Users frequently want to be told ‘what to do’ and look in documentation for instructions that take them step by step through the execution of a particular task. This is not only what readers ask for, it is also what technical authors work hard to produce and what many researchers regard as key (Duggan & Payne, 2001; D. K. Farkas, 1999; Karreman & Steehouder, 2004; Karreman et al., 2005; van der Meij et al., 2003; van der Meij & Gellevij, 2004).

A series of stepwise instructions, often called a ‘procedure’, has a rhetorical component in that its goal is to get a person to actually do something. In addition, there is a system–theoretical one: a procedure describes states (desired state, prerequisite state, interim states and unwanted states) as well as the actions (human, system and external) that bring about transition from one state to another (D. K. Farkas, 1999).

After studying a large number of procedures as they are presented in actual, existing hardcopy and softcopy documentation products, Farkas defined a

canonical form for them: the “streamlined-step procedure”. This consists of a series of brief steps, each consisting of one action statement, perhaps enhanced with an equally concise description of the resulting system state. The steps are formatted simply and built around an imperative verb. Conceptual information is optionally included, but only just before the first step and after the last: before the first step, a brief paragraph may explain the conditions under which the procedure applies while after the last step, supplementary information may be given.

Stepwise instructions provide procedural and possibly situational information at the syntactic layer of the software. They are extremely common in documentation, not just for software but for a wide range of products (see Figure 32, Figure 46 and Figure 56).

To insert a marker in a source document:

- 1 Click where you want to insert the marker and choose Special > Marker.
- 2 Choose a marker type from the pop-up menu. You can use any predefined marker type except Conditional Text, Header/Footer \$1, Header/Footer \$2, or Cross-Ref. You can also define your own marker types. (See “Adding custom marker types” on page 440.)
- 3 Enter text that you want to appear as the list entry in the list. You can enter up to 255 characters (127 Japanese double-byte characters). You can type the text or use alternative methods to enter marker text without typing. (See “Adding index markers” on page 428.)
- 4 Click New Marker. A marker symbol T appears when text symbols are visible.

Figure 56: Streamlined-step procedure in the printed user manual for a document development tool⁴⁹

solution	Therefore:
	Include stepwise instructions when a fixed sequence of interactions must be carried out to transform the system from one well-defined state into another.
rationale	By providing a ready-to-use situation model, stepwise instructions remove load. The streamlined-step format reduces additional load from the documentation journey, by making the procedure that is described as easy to follow as possible.
consequences	Where there is width in the software’s use complexity, the wrong procedure will be selected if the user incorrectly recognizes a situation in which the procedure applies. It is therefore advisable to dampen an overly-strong recognition that may be present by including the optional non-procedural elements in the streamlined-step procedure.
	Stepwise instructions remove load by removing situation processing altogether. Thus, little learning will follow from carrying out the steps. So, even if such a

⁴⁹ Adobe FrameMaker 8 User Guide, © 2007 Adobe Systems Incorporated, p. 421

procedure is what software users want from and look for in documentation, it is not necessarily always what they need.

Although stepwise instructions are traditionally given at the syntactic layer, they can also be given at the semantic layer. Rather than procedural and possibly situational information, they then contain strategical and possibly conceptual information. Elaborating on earlier work (Duff & Barnard, 1990), Eiríksdóttir and Catrambone (Eiríksdóttir & Catrambone, 2011) found that “specific instructions help initial performance, whereas more general instructions, requiring problem solving, help learning and transfer”. “Specific” refers here to the syntactic layer, and “general” to the semantic layer. In a study conducted a few years later (Eiríksdóttir & Catrambone, 2014) it was confirmed that using instructions at the semantic layer is not the users’ first choice. However, as their skill increased, participants were seen to move away from specific instructions at the syntactic layer in favour of general instructions at the semantic layer.

To bring the need for situation processing back where situation processing is required for learning, the streamlined-step format can be elaborated in a number of ways.

Therefore, when implementing this pattern:

- **To distinguish procedural from strategic knowledge, position the procedure at either the syntactic or the semantic layer: do not mix.**
- **Where there is width in the software’s use complexity, include non-procedural elements in the streamlined-step format to discuss the conditions under which the procedure applies.**
- **For software with high use complexity, include in the non-procedural discussion goal-dependent user decisions, as well as the long-term consequences of applying the procedure.**



... STEPWISE INSTRUCTIONS can be provided as separate JOB AIDS or embedded in a wider narrative providing situational or conceptual information. Situational knowledge of when to apply the instructions can be provided with SCREEN CAPTURES. In an implementation of the SEPARATION OF PURPOSE pattern, STEPWISE INSTRUCTIONS are used to implement read-to-act blocks...

10. *Discussion and Conclusions*

In this thesis, I set out to answer the following question:

Can design rationale for a coherent approach to the design of software documentation artefacts be found in a framework that describes the system dynamics of human performers interacting with, and learning from the interaction with, software and information?

The separate chapters of this work deal with a number of component research questions. These will now be discussed one by one, after which I shall return to the overall research question.

Component Research Questions: Conclusions

RQ1: What is the relevance of the current work to society and academia?

Chapter 2 shows how decades after the use of software became ubiquitous in all aspects of life, users still tend to not fully master the software tools that they work with. Inefficient use, ineffective use and under-use of software are the order of the day. With repeated practice over time, users reach an intermediate level of mastery; after which they no longer make further progress and find themselves stuck at a suboptimal level of competence. The situation has become known as the Paradox of the Active User.

If we define ‘design’ as purposeful endeavour resulting in a particular product for the benefit of a human performer, then there exist various design disciplines that work to alleviate the Paradox of the Active User. Human-Computer Interaction or HCI design targets the software tool itself, aiming to make it as usable and as useful as possible. Instructional design targets the user before any work is undertaken, educating him in a classroom setting or through self-study. The user can also be targeted during his work with the software: when this is done through the provision of pre-recorded information, purposefully selected and presented to assist future users, we speak of documentation design. Of these support disciplines, documentation design has the least status. It is regularly dismissed as irrelevant because only there to make up for defects in the software’s user interface; and because “nobody reads it”—assumptions that do not stand up to close scrutiny. There exists no recognized ever-growing body of knowledge in the field, constructed by an established academic community that is served by multiple dedicated conferences and journals. The literature in the field is

fragmented so that ‘keeping up with the literature’ is an impossible task for scholars and practitioners alike. As a consequence, the field as a whole does not make much progress. When research into documentation is carried out, this tends to embrace new technology-enabled communication media without questioning.

The Paradox of the Active User entered our lives only when software-driven tools became commonplace. There exists a fundamental difference between physical tools and software-driven tools that is pivotal to all attempts to defeat the Paradox of the Active User. Yet user documentation in the 2010s looks very much as it has looked for a long time. At the same time, the authors of commercially available software-related books develop new genres in abundance. But whether their new designs are indeed useful, and if so, under which conditions, is not known.

The persistence of the Paradox of the Active User is frustrating for users and detrimental to the economy, leading as it does to many wasted man-hours. Dissatisfaction of users with documentation should be an incentive to improve the documentation, rather than giving up on this form of user support altogether. To create documentation artefacts that helps users to achieve mastery, practitioners look to academia for research results which are not forthcoming.

Before we can think about designing interventions to guide a process towards a particular outcome, we need to understand the mechanism of the process when it runs its natural course. This will not in and by itself dispel the Paradox of the Active User through documentation but it will offer a framework for description, discussion, and design of documentation artefacts.

Conclusion: There exists no coherent body of research into user documentation for software. As a result, documentation artefacts are not evidence-based and often fail to help users fully master the software they work with. This leads to considerable waste of resources in the workplace, which is economically damaging.

RQ2: What is known about the way in which people interact with documentation?

Chapter 3 shows how the way in which people interact with information was initially conducted from a viewpoint of not the users of the information but the artefacts in which it resides and the venues in which it is sought. More recently the focus has shifted and widened to become more naturalistic and person-oriented, placing the search for information in the particular context in which the information is sought.

Seeing this process as an ‘information journey’ which is iterative and self-shaping, we can identify four separate stageposts along the journey that are all visited, although not necessarily sequentially. The information journey when narrowed

down to documentation of software tools becomes a 'documentation journey'. An information seeker is under no compulsion to visit all the stageposts but can decide at any moment to abandon the journey altogether. We can offer the user information but we cannot make him do anything with it. Yet all of the stageposts can, and should, be leveraged when designing user documentation. Depending on how the documentation journey is ideally shaped, the preferred 'route' can be made easy whilst others are downplayed.

Conclusion: Although there is no way that a user can be forced to access documentation in a particular, pre-defined manner, it is possible to shape his documentation journey by designing for ease of access to the different stageposts.

RQ3: How can the process be modelled that software documentation is designed to support?

Publications for practitioners (such as Schriver, 1997) provide guidelines for making the documentation journey as effortless and effective as possible. The documentation journey describes however no more than a secondary task, which has meaning only in the context of a primary task. Chapter 4 shows what the documentation journey described in Chapter 3 looks like when embedded in the context of repeated interaction with a particular software tool.

A model of Computer-Mediated Activity (CMA) is presented that describes how the total amount of knowledge that an individual possesses with regard to a particular software tool (the 'knowledge base') is added to during practice over time. The process can be seen as a 'knowledge engine'. The engine is fuelled by the performer's current ambition and transforms information into value, with knowledge as a by-product. The fuller the knowledge base, the more it resists further expansion. The process therefore slows down and eventually it comes to a halt. Ideally, but not necessarily, this is at the exact moment that the knowledge base is full and mastery has been achieved. The engine can easily stop prematurely, before complete mastery has been achieved.

The CMA model presented in this chapter proposes a meaningful description of the Paradox of the Active User and in doing so, points out goals for documentation artefacts to achieve. It offers an explanation of why the knowledge engine is much more likely to stop running prematurely in computer-mediated activity than it is in other tool-mediated activity. It also shows how the engine can be made to continue running for longer if ways are found to strengthen certain cognitive constructs such as ambition and perceived value, or to weaken others such as recognition and perceived load. Thus it shows potentials targets for documentation artefacts.

Conclusion: Combining tried-and-tested frameworks of human cognitive behaviour in naturalistic settings, a model can be constructed of Computer-Mediated Activity (CMA) that offers a meaningful and practically applicable description of the problem space in which documentation designers move.

RQ4: How can a generic, theoretical model of computer-mediated activity (CMA) be instantiated to apply to specific, practical documentation design problems?

Chapter 5 shows that ‘mastery’ means different things for different types of software, and how to set the boundaries for any given documentation design project by assessing the software’s use complexity.

Use complexity is a characteristic of the software only, irrespective of any characteristics inherent to the user or the task environment. Use complexity is a multi-dimensional construct. It is a measure of the quality and quantity of the learning required to achieve full mastery of a particular software tool and can be analyzed from a number of different viewpoints.

Although absolute values for the different dimensions cannot feasibly be determined, a relative scale can be envisaged ranging from, say, ‘almost none’ to ‘huge amounts’. It then becomes possible to take the idea of dimensions literally and map out the results of the separate analyses. This leads to a classification of software according to its use complexity. A particular software tool can be classified as operator software (least complex), actor software (more complex) and activator software (most complex). As we move up, the demands placed on the documentation quickly become more challenging.

Moving up in use complexity from operator software through actor software to activator software, ever more constructive learning (as opposed to reproductive learning) is required. Constructivist principles of learning as worked out for ‘constructivist learning environments’ become steadily more important: for activator software, what is required is really a ‘constructivist documentation environment’. From the CMA model a new interpretation emerges of the Paradox of the Active User: *The more learning is required, the less easily it takes place. Also: the more learning is required, the less users are likely to seek it.*

Conclusion: The CMA model can be instantiated for a particular documentation design task through analysis of the use complexity of the software being documented.

RQ5: How may a Software Documentation Design Pattern Language (SDDPL) be constructed on the foundation of the CMA model?

Chapter 6 shows how so-called design patterns, combined into design pattern languages, provide one way of recording and transferring design knowledge. Each design pattern offers a description of the invariant parts of proven designs as a solution to a problem in a specific context. A design pattern language is a collection of design patterns in a particular domain, all sharing an underlying value system and an organizing principle.

In this chapter a Software Documentation Design Pattern Language or SDDPL is constructed, with empowerment, through the achievement of mastery, as the underlying value system. The language contains patterns for the design of documentation artefacts that aim to engender, over time, full mastery of the software that is being documented. Justification of the individual patterns is found in the CMA model.

Next, to determine the optimum organizing principle, practicing documentation designers are identified as the main stakeholder group in the pattern language. This group has two major interests in the language. During design practice, patterns may be applied to solve a particular problem. Quite separately, experienced designers may wish to record their knowledge by writing patterns. The SDDPL's usability is therefore discussed with a view to these two separate uses of the language. The resulting organizing principle caters for both interests,

Conclusion: The SDDPL has the achievement of mastery as described by the CMA model as its underlying value system. Its organizing principle aims to maximize usability for its main stakeholders: those who apply existing patterns to current design problems as well as those who record their design knowledge for the benefit of others.

RQ6: Can proposed and existing documentation design solutions be expressed in terms of the SDDPL?

Chapters 7, 8 and 9 present a number of SDDPL patterns. The first of these three chapters contains 'macro' patterns: those are applied to a complete documentation set. The next presents 'meso' patterns, for application to a complete documentation artefact. The last of the three chapters presents a number of 'micro' patterns, for application to elements within a documentation artefact.

Some of the patterns originate from theory (e.g. CONCEPTUAL MODEL, DOCUMENTATION ENVIRONMENT, EVERY PAGE IS PAGE ONE), while others were mined from existing documentation artefacts (e.g. COOKBOOK, SCREEN CAPTURE, CROSS-REFERENCE).

Conclusion: Design patterns for software documentation can be expressed in the SDDPL at all three hierarchical levels (macro, meso and micro) and regardless of whether a pattern originates from theoretical considerations or from existing practice.

Overall Research Question: Discussion

RQ0: Can design rationale for a coherent approach to the design of software documentation artefacts be found in a framework that describes the system dynamics of human performers interacting with, and learning from the interaction with, software and information?

The SDDPL (Chapter 6) is a design pattern language, and as such it is a coherent approach to the design of software documentation artefacts. Its design rationale is found in the CMA model constructed step-by-step in Chapters 3, 4 and 5, which is a framework that describes the system dynamics of human performers interacting with, and learning from the interaction with, software and information. Since Chapters 7, 8 and 9 successfully present a number of SDDPL patterns, the overall research question can be briefly answered with ‘Yes’.

There are however a number of caveats.

- The research question has been answered by creating a design pattern language. This is not the only ‘coherent approach to the design of software documentation artefacts’ imaginable. To the contrary: the design pattern approach is relatively unknown in fields other than architecture, UID or programming. Outside these fields, design practitioners are more commonly provided with straightforward guidelines.
- Although the CMA model is based on a number of well-accepted descriptive frameworks of human cognitive behaviour in naturalistic settings, validation is problematic. The mechanics of the knowledge engine describe a complex process that takes place almost exclusively within people’s heads. It cannot be studied directly or even indirectly. There exist no metrics for ‘recognition’, say, or ‘ambition’. Nor can we study people in a naturalistic environment for a long enough period of time for them to actually add to their knowledge base; and if

we could, there would be no way to distinguish events that are part of the process from those that are not. All we can do, therefore, is some sort of algorithmic validation. The simple plots in Chapter 4 do certainly not reflect quantifiable constructs working in a predetermined manner: they show no more than a pattern of development of key elements in the CMA model over time.

- The SDDPL thus far is no more than an academic exercise. Although it provides an answer to the research question, it has not been tested in design practice. Currently, it is not known whether the SDDPL can help documentation designers create documentation artefacts that defeat the Paradox of the Active User.
- The SDDPL has been structured with a dual purpose: to be a working design pattern language, and to satisfy the demands of a PhD thesis examination committee. The latter audience may well have compromised usability requirements for the former.
- Although the hierarchy of scope for patterns within the language (expressed as macro, meso and micro) proved workable, it is not perfect. Some patterns could arguably be placed in a different group than that in which they are now placed: for example, SEPARATION OF PURPOSE and MOTIVATOR are labelled meso patterns, to be applied to a particular documentation artefact; but the principles could just as easily be scaled up to apply to a complete documentation set.
- No coherent approach has been applied to the mining of patterns in the language. An attempt to ask users of documentation artefacts (*Appendix 3. Using the Repertory Grid Technique for Mining Design Patterns*) yielded only micro patterns. It is difficult to see how this could ever be otherwise: other mining methods should be devised.

Research Agenda

These considerations bring me to a proposed research agenda, outlining further work that needs to be done before the CMA model and the SDDPL can make a genuine difference to current documentation design practice.

- Further validating the CMA model by ‘running’ it in an independently-developed qualitative modelling environment, such as Garp3 (Bredeweg, Linnebank, Bouwer, & Liem, 2009).
- Testing the SDDPL in design practice, to determine if it indeed helps defeat the Paradox of the Active User through documentation.
- Testing the organizing principle (format) of the SDDPL for usability (see p. 100).

- Testing the contents of the individual patterns to ensure that those parts of the narrative that are mainly of academic interest do not make the patterns less accessible to practitioners.
- Further investigating the applicability of the hierarchy of scope for patterns within the language (macro, meso and micro).
- Devising methodical methods for mining SDDPL patterns at all levels in the hierarchy.
- Investigating whether the CMA model can be used to justify sets of guidelines as well as a design pattern language.
- Adding further patterns to the SDDPL, until it offers a more or less complete reference to the problem of “designing documentation”.

A Word To Practitioners

All the work outlined in the research agenda will not be finished in a few months or even years. Even then, the SDDPL will never be complete. Therefore, I would like to conclude this work by outlining a few very general guidelines (not patterns!) for practicing documentation designers to bear in mind when documenting software for end users.

It is my contention that for practice to make perfect in a naturalistic setting the knowledge engine must be fuelled; specifically by:

- enabling and stimulating upgrading to as close to 100% of the difference between the current ambition and the current level of expertise as possible;
- maintaining or eventually raising the ambition to match an intention horizon enclosing 100% of that which the software is capable of.

Then:

- Cater for the production bias by making every information product visibly useful to performance of either the task at hand or in the near future.
- Within the limitations posed by the production bias, induce generative cognitive processing (self-reflection and self-explanation), to stimulate learning and to raise a barrier against the overly eager application of the first information that is found and that seems applicable to the situation at hand—whether it actually is or not.
- Avoid oversimplification.
- Limit the information content of each separate information product to a well-defined scope that is immediately visible to the user.

- Ensure completeness within scope, to guarantee usefulness.
- Ensure that the set as a whole contains information products that throw light on the same cases from different angles, providing different perspectives.
- Ensure that the set as a whole caters for expertise increasing over time.

If you bear all this in mind, then all the rest will follow. May your books be great, your readers happy, and your pay rise imminent.

11. Appendixes

Appendix 1. Software Documentation: A Standard for the 21st Century

The study described in this Appendix has been presented in a full conference paper at the Information Systems and Design of Communication (ISDOC 2014), held on 16-17 May 2014 in Lisboa, Portugal (van Loggem, 2014b).

To lend credence to the author's proposition that the 'mainstream' approach to documentation has remained unchanged for a number of decades, and moreover is hardly distinguishable from that adopted when documenting non-software-based, physical tools, in this Appendix the level of conformance will be determined of an instruction manual for a table loom, published in 1925, to *ISO/IEC 26514:2008 (Systems and software engineering — Requirements for designers and developers of user documentation)*; from now on, referred to as "the Standard".

Method

Conformance to the Standard was verified for the publication *Weaving with Small Appliances, Book III: Table Loom Weaving*, written by Luther Hooper and published by Pitman & Sons in 1925⁵⁰. It contains 71 pages. Although one in a series of three, this manual is intended to be read stand-alone. Other books in the series describe how to use weaving appliances even simpler than the table loom: the weaving board (Book I) and the weaving tablet (Book 2).

Verification of conformance was carried out by applying Annex G of the Standard: *Requirements clauses and checklist for documentation products*. It exhaustively lists those clauses in the body of the Standard "which contain requirements (*shall* statements) for documentation products." Annex G is, as is all of the Standard, protected by copyright and may not be reproduced. Permission has been granted, however, to put it at the disposal of the Examination Committee of this PhD thesis, provided it is not included in the body of the work. This will be done separately.

Figure 57 below shows the first requirement as it is listed in Annex G.

⁵⁰ http://www.cs.arizona.edu/patterns/weaving/books/hl_table.pdf. Downloaded on 12 March 2014 from http://www.cs.arizona.edu/patterns/weaving/topic_loom.html.

Clause no.	Guideline	Applicability		Conformance			
		Yes or No	Reason not applicable	Yes	Partial	No	Comments
10	Structure of documentation ... When a document set will address audiences with widely differing needs, at least one of the following structures shall be used ... (list follows)						

Figure 57: The first requirement listed in Annex G; in this Appendix, given reference #1

In the Annex, the requirements are separately listed but not uniquely identified. Only a reference is given to the clause containing the requirement. A single clause in the body of the Standard may contain multiple requirements. To enable reference to the original, in the discussion that follows all requirements have therefore been assigned a unique reference number by the author. The 101 requirements identified in the Annex have been numbered consecutively in the order in which they are listed (which is the order in which they appear in the Standard), from 1 to 101. For example, the requirement shown in Figure 57, summarizing the first of a number of requirements following from clause 10 in the body of the Standard, is given reference #1.

Analyses

In this section, the result of the verification process is looked at from two angles.

First, conformance is determined to all those requirements that are applicable in the case of *Weaving with Small Appliances*. Homing in on requirements that are not met, it is then discussed whether the non-conformance is related to a fundamental mismatch between the type of manual verified (for a hand-weaving loom) and that of the Standard against which it is verified (for software manuals): by seeing if the manual could imaginably be changed so as to conform to all applicable requirements.

Then, we turn our attention to requirements that turned out to be not applicable. Arguably, any user manual could be made to conform to any Standard for user manuals simply by declaring most of the Standard not applicable; which does away with the need to conform altogether. It is therefore important to, once again, determine whether the non-applicability is related to the fact that the manual does not describe a product of the type that the Standard was intended for: by seeing if the same set of requirements could equally well be not applicable in the case of a software-related manual.

Conformance and non-conformance

As a first step, straightforward conformance of *Weaving with Small Appliances* to the Standard was determined. The result of this analysis is that out of in total 101 requirements, 49 are immediately met. For example, the document is structured into units with unique content (#3), each page is uniquely labelled (#4), and each documented feature is related to the overall process or task (#31).

Only the following 6 requirements are not or only partially met:

#10, referencing clause 10.4 of the Standard which lists, amongst a number of other required elements, a glossary as required “if documentation contains unfamiliar terms”. *Weaving with Small Appliances* does contain unfamiliar terms, which are explained on first use, but it does not contain a glossary. The same requirement, which is not separately counted, is #48, referencing clause 11.12 of the Standard where it states that “Documentation shall include a glossary if terms or their specific uses in the software user interface or documentation are likely to be unfamiliar to the novice users in the audience.”

#13, referencing clause 10.5.1 of the Standard which states that “the introduction is the first chapter or topic of the document. The introduction shall describe the intended audience, scope, and purpose for the document and include a brief overview of the software’s purpose, functions, and operating environment.” The first chapter of *Weaving with Small Appliances* does describe the product and its “purpose, functions, and operating environment” but the “intended audience, scope, and purpose for the document” is missing. Clause 10.5.1 goes on to say that “Introductions shall be provided within a document for each chapter and topic. Introductory sections should be provided for each major feature or function of the software being documented. The introductory sections shall provide an overview of the topic, the purpose of the function, and environmental requirements, warnings, cautions, or user requirements unique to the topic.” All chapters in *Weaving with Small Appliances* are very short, typically no more than three or four pages, and none open with an introductory section as prescribed by the Standard.

#29, referencing clause 11.4 of the Standard which states that “The documentation shall include information on how it is to be used (for example, help on help), and an explanation of the notation (a description of formats and conventions)”. There is no information in *Weaving with Small Appliances* on how the book is to be read. Neither is there a description of formats and conventions, although the latter cannot be seen as a requirement as the book does not use a particular notation (and this, itself, is not a requirement).

#33, referencing clause 11.7 of the Standard which states that “Procedures shall include: preliminary information; instructional steps; completion information.” In *Weaving with Small Appliances*, procedural instructions are given in running text, not visually or stylistically distinguished from narrative. A related requirement is

for this reason not separately counted: #99, referencing clause 12.16 of the Standard where it states that “Instructional steps shall be consecutively numbered.”

#46, referencing clause 11.10 of the Standard which states that “The documentation on resolving problems shall also include contact information for reporting problems with software or its documentation, and suggesting improvements.” The author of *Weaving with Small Appliances* (who is also the manufacturer of the table loom) does not solicit readers’ questions or comments and does not provide contact information.

#91, referencing clause 12.14.3 of the Standard which states that “The titles in the list of tables, figures, or illustrations shall be identical in wording to those in the document”. There is a list of illustrations but the titles of the illustrations appear only in this list: the illustrations themselves are numbered but have no captions.

Six requirements turned out in this analysis to be not or not fully met. Could the manual be modified so that all requirements other than those that are not applicable are met? It seems that this would be quite feasible. Full conformance would necessitate no more than the following:

- adding a glossary;
- stating the intended audience, scope, and purpose of the document in the introductory chapter and adding a few introductory lines to every chapter;
- explaining how the book is to be used in conjunction with the product itself;
- breaking up the procedures into separate steps (and ensuring that they then conform to a number of requirements pertaining to stepwise procedures, which now were not applicable; see below);
- adding contact information for reporting problems with and suggesting improvements to the product;
- providing every illustration with a caption corresponding to that printed in the list of illustrations. (Alternatively, the list of illustrations could simply be removed from the document. Its inclusion is not required when illustrations are referred to in text immediately preceding or following them, as is the case in *Weaving with Small Appliances*).

From this list we may conclude that *Weaving with Small Appliances* can be made to conform to the Standard for software user documentation relatively easily. Required changes are superficial: there is no need to re-consider the approach, the structure, or the format of the book.

Non-applicability

The Standard offers the possibility to mark requirements as not applicable. As pointed out earlier, all requirements that are software-specific would by definition

be not applicable in a manual for a physical tool such as a table loom. If this is the reason why conformance of *Weaving with Small Appliances* can be achieved by making only superficial modifications to the text as it was written in 1925, then its conformance says little or nothing about the Standard's specificity to software documentation. The next step must be, therefore, to analyse the requirements that were declared not applicable. We can then determine if the same set of requirements could imaginably be not applicable in the case of a software-related manual.

The following four main reasons were found for non-applicability of requirements in the case of *Weaving with Small Appliances*:

- The requirement applies to a particular aspect of the context in which the document is made available, and this is not the context in which *Weaving with Small Appliances* is made available. In total, 8 requirements were not applicable to *Weaving with Small Appliances* for this reason.
 - There being only one intended audience results in non-applicability of requirements #1, #2, #26, #28, and #84. These all reference clauses which are explicitly declared valid only for document sets or sections each addressing a separate audience.
 - There being no updates or upgrades in table looms (or any other physical tools) results in non-applicability of requirement #23 referencing clause 11.2 where it states that "If the previous documentation version is no longer accurate, current documentation shall be available for customers acquiring software updates or upgrades."
 - The publication being a book, which is not usually packaged, results in non-applicability of requirement #25, referencing clause 11.3 where it states that "Identification data shall appear on a package label, legible without opening the package, and on a title page. A package label is not required if the title page is legible without opening the package."
 - The author (who is also the manufacturer of the product) not being part of an organization results in non-applicability of requirement #27, referencing clause 11.3 here it states that "The identification of the document and the software shall be consistent with the CM practices of the issuing organization or the acquiring organization."
- The requirement applies only to on-screen documentation. In total, 16 requirements were not applicable to *Weaving with Small Appliances* for this reason. These would be not applicable also for any other paper-based documentation product, including software documentation.
 - The paper-based format of the documentation results in non-applicability of requirements #6, #50, #52, #53, #54, #56, #57, #58, #59, #62, #63, #79,

#80, #81, #92, and #94. These all reference clauses which are explicitly declared valid only for on-screen documentation.

- The requirement applies only when a particular element is present in the documentation, and in *Weaving with Small Appliances* this element is not used. In total, 21 requirements were not applicable to *Weaving with Small Appliances* for this reason. These would be not applicable also for any other documentation product, including software documentation, that does not make use of a particular element. Note that presence of the element may (but need not) be a separate requirement.
 - The absence of reference information (not required) results in non-applicability of requirements #8, #20, #21, #42, and #45.
 - The absence of introductory sections (required) results in non-applicability of requirement #14.
 - The absence of warnings, cautions and notes (not required) results in non-applicability of requirements #15, #16, #47, #64, #95, #96, #97, and #98.
 - The absence of procedural steps (required) results in non-applicability of requirements #34, #35, and #36.
 - The absence of a glossary (required) results in non-applicability of requirement #49.
 - The absence of unusual, flexible, or complicated navigational features results in non-applicability of requirement #75.
 - The absence of sections within the chapters results in non-applicability of requirement #88.
 - The absence of long tables spanning more than one page results in non-applicability of requirement #101.
- The requirement applies only when a particular element is present in the documented system, and this element is not present in the table loom. The following 9 requirements were not applicable to *Weaving with Small Appliances* for this reason:
 - #38, referencing clause 11.7.2 which states that “Procedural steps shall include or provide reference to documentation of the acceptable range, maximum length and applicable format, and unit of measurement of data fields for user-supplied data.”
 - #41, referencing clause 11.8 which states that “the documentation developer shall explain the formats and procedures for user-entered software commands, including required parameters, optional parameters, default options, order of commands, and syntax”

- #43, referencing clause 11.8 which states that “Documentation shall explain how to interrupt and undo an operation during execution of a command and how to restart it, if possible.”
- #44, referencing clause 11.8 which states that “Documentation shall describe how to recognize that the command has successfully executed or abnormally terminated.”
- #70, referencing clause 12.11 which states that “Graphical user interface (GUI) elements of the software, such as buttons, icons, and variable cursors and pointers; special uses of keyboard keys or combinations of keys; and system responses shall be represented in documentation by consistent graphic or typographical formats so that the various elements are each distinguished from the text.”
- #71, referencing clause 12.11.1 which states that “Documentation formats for user-entered commands or codes shall clearly distinguish between literals (to be input exactly as shown) and variables (to be provided by the user).”
- #72, referencing clause 12.11.1 which states that “Formal notation, such as the use of brackets, braces, greater than (>) and less than (<) characters, and other marks, shall be defined in every document that uses it”
- #73, referencing clause 12.11.1 which states that “Quotation marks shall not be used in command representations unless the user should input them literally.”
- #74, referencing clause 12.11.2 which states that “The documentation shall establish consistent conventions for representing special keyboard keys and explain the conventions in documentation for software that uses special keys for input.”

It is this last category, of requirements that are not applicable as a direct result of the product that is documented being a physical rather than a software-driven tool, that is of the most interest. Is the conformance of *Weaving with Small Appliances* contingent on all software-specific requirements being not applicable? Or, turning the question around: Are software manuals imaginable to which this same complete set of software-specific requirements in the Standard would be not applicable?

The answer is that such a software user manual is easily imaginable. In fact, one would be hard pressed nowadays to find a software manual, at least for common business software packages, for which the software-specific requirements in the Standard are all applicable. Only one requirement out of the set does not refer to software that is operated through a so-called Command Line Interface or CLI. In a CLI, the keyboard is used to enter character-based commands and data values following strictly prescribed syntactic rules. Yet in all but a very limited number of

present-day software packages for end users the CLI has been long superseded by the Direct Manipulation Interface or DMI, where use of the keyboard is not required for sending commands to the software. Microsoft® Windows NT 3.1 dates from 1993, at which time earlier versions of Windows were already well established and competed with OS/2 on what was known as the “IBM-compatible PC” and Mac OS on the Apple Macintosh. All these operating systems already implemented a DMI rather than a CLI user interface. The CLI, to which most of the software-specific requirements in the Standard refer, has not been the predominant paradigm for at least twenty years.

Generally speaking, if software poses formal restrictions on user-supplied data (#38) then nowadays it would convey these through the user interface itself, making it impossible or at least very difficult for the user to enter them. End users need hardly ever, if at all, enter commands with required or optional parameters in a particular order and according to a particular syntax (#41) where literals and variables must be distinguished (#71) and which need to be documented using a formal notation (#72) with or without quotation marks (#73). Also, commands are executed almost immediately, leaving no scope for interruption or restarting (#43) nor doubt as to whether they have successfully completed (#44); and special keyboard keys (#74) are not often of particular interest to the user.

Only one software-specific requirement in the Standard relates to present-day, DMI-driven software: that graphical elements in the user interface be graphically or typographically treated in a consistent manner and formatted differently from the surrounding text (#70). But even this one remaining requirement is not applicable when documenting the very latest generation of software products, where such elements are used as sparingly as possible and only those are present that are already thoroughly known to the user. When documenting, for example, apps for the iPad it is hard to see where a description of “buttons, icons, and variable cursors and pointers; special uses of keyboard keys or combinations of keys; and system responses” would be appropriate.

Discussion

The previous section described how the user documentation for a physical tool, written in 1925, was verified for conformance to an ISO standard for the user documentation of software, last updated in 2008. Table 5 below summarizes the total number of requirements that were immediately met; that were not or only partially met; and that proved not applicable.

Table 5: Conformance to the requirements

met	not (fully) met	n/a because of the context of distribution	n/a because of the medium used (print)	n/a because a particular element is absent in the document	n/a because a particular element is absent in the system
49	6 (+2 duplicates)	8	16	21	9

Taking into account not just the bare figures but also the considerations laid out in the previous section, we conclude that:

- *Weaving with Small Appliances* all but conforms to the Standard;
- it would completely conform after only slight modifications and without altering its style, approach, or structure;
- conformance is not contingent on the non-applicability of requirements that are generally applicable to software-related user documentation.

The question now is whether conclusions can be drawn from this very limited test as to the adequacy of present-day guidelines for the documentation of software. It is the author's contention that this is indeed the case.

The Standard must be taken as an example of state-of-the-art guidelines for software user documentation. In ISO's own words⁵¹: "ISO standards represent, by an international consensus among experts in the technology concerned, the state of the art. To ensure that ISO standards retain this lead, they are reviewed at least every five years after their publication. The technical experts then decide whether the standard is still valid, or whether it should be withdrawn or updated". Further evidence for the standard representing mainstream software documentation practice is found in the fact that its Annex E contains "excerpts from the checklists that are used to judge manuals and online help in international competitions of the Society for Technical Communication". The Society for Technical Communication (STC) is the world's largest professional society for technical communication, and the STC's involvement in the Standard must be seen as an endorsement from the shop floor.

All 101 requirements in the Standard relate to surface characteristics of the documentation, many of which are relatively trivial. Do we need this particular ISO

⁵¹ http://www.iso.org/iso/home/faqs/faqs_standards.htm, accessed on 12 March 2014

standard to prescribe the use of page numbers (#4) or the consecutive numbering of instructional steps (99)? The attention to detail expressed by #93, which specifies the allowed depth of cross-referencing within the index, is commendable; but it is not necessarily spent on a particularly important consideration when documenting software. Naturally, such surface aspects of documentation need be prescribed somewhere. However, leaving them out of a standard that was explicitly drawn up for the documentation of software would leave more room for software-specific considerations. In the Standard, only about 10% of requirements are software-specific—the same proportion of requirements, for example, as deal with the presence and presentation of warnings, notes and cautions (#14, #15, #16, #35, #47, #64, #95, #96, #97, and #98).

Appendix 2. “Nobody Reads the Documentation”—True or Not?

The study described in this Appendix has been presented in a full conference paper at ISIC 2014: The Information Behaviour Conference, held on 2-5 September 2014 in Leeds, United Kingdom (van Loggem, 2014a).

Introduction

The study presented in this Appendix was carried out to provide empirical evidence to confirm, or not, the widely held belief that software users don’t read documentation.

A handful of research studies have previously been carried out to determine whether users are indeed reluctant to consult the documentation that is delivered with a product, and these are surprisingly unanimous in their findings. The different studies vary greatly as to respondents, method and the exact nature of the question. However, they invariably conclude that—at least for complex and unfamiliar products—the documentation *is* consulted; even if it is not read, marked, learned, and inwardly digested in its entirety. Table 6 below shows a summary of the available research, showing the percentage of respondents reported as ever having consulted documentation of a particular type.

Table 6: Earlier studies into the use of documentation

% ‘yes’	N	consultation of	ref
82.9	44	the printed documentation for complex equipment such as VCRs	(P. Wright, Creighton, & Threlfall, 1982)
96	201	instruction manuals	(Schrivver, 1997)
99	400	the printed manual for a major word processing program	(Smart, DeTienne, & Whiting, 1998; Smart, Whiting, & DeTienne, 2001)
65	400	the online Help for a major word processing program	(Smart et al., 1998; Smart et al., 2001)

% 'yes'	N	consultation of	ref
95.5	224	the printed manual for an accounting software package	(Vromen & Overduin, 2000)
58.9	365	the manual of the vehicle that they drive most often	(Mehlenbacher, Wogalter, & Laughery, 2002)
92	201	the manual that comes with a product they buy	(Jansen & Balijon, 2002)
59	107	the printed manual for any piece of software	(Martin, Ivory, Megraw, & Slabosky, 2005)
57	107	the online Help for any piece of software	(Martin et al., 2005)
91.2	70 (older adults)	product manuals for technological products	(Tsai, Rogers, & Lee, 2012)

In addition to the studies summarized in Table 6, there are a few reporting considerably lower rates of recourse to documentation (Ceaparu, Lazar, Bessiere, Robinson, & Shneiderman, 2004; Mendoza & Novick, 2005). These, however, are characterized by a different focus, studying as they did the occurrence of users turning to sources of support on experiencing “frustration” with software as a result of poor usability. Many of the frustrations described did not result from lack of mastery of the software but followed from network breakdowns, time delays, and recognized user errors such as saving a file to the wrong location. In such situations the origin of the frustration is already known, and it is not surprising that no attempt is made to alleviate it by turning to documentation. Also, there is no reason to assume that recourse to documentation is necessarily preceded by a sensation of frustration. It is well imaginable for an individual to turn to documentation while experiencing a non-frustrating desire to know. For these two reasons, the “frustration” studies recorded only a subset of all recourses to documentation, of unknown proportion. They are therefore not included in Table 6.

Doubt has been expressed (Novick et al., 2007) as to the validity of the results summarized in Table 6. Moreover, some of them pertain to consumer products whilst others discuss the use of documentation when working with software. The

study described here, therefore, specifically focused on the sources of information that users of software refer to in the course of their work.

Study Design

Instrument and Method

A single-sheet questionnaire was prepared which offsets six possible sources of information against four possible situations in which information can be required when working with software. In the Netherlands, where the study was conducted, the national language is Dutch but there are many environments where English is the working language. For this reason, the questionnaire was prepared in two language versions (Dutch and English). In all respects other than language, the two versions of the questionnaire were identical. An image of the English language version of the questionnaire is presented in Figure 58.

The information sources were described as follows (all quotations are taken from the English language version of the questionnaire):

- instructions delivered with the software (in print or as a PDF)
- Help that you call up from within the software
- online sources (through a web browser)
- shop-bought book (either your own or borrowed)
- ask someone you know (colleague, family member, etc.)
- other, that is: _____

The situations in which information might be sought were described as follows:

- understand how the program or part of it works, and what it can do
- look up how to carry out a particular procedure
- quickly look up a particular fact that you don't know or cannot remember right now
- find out what went wrong when there is a problem

Below you see (from top to bottom) four situations in which you could imagine needing information on how a software package that you are working with works. From left to right, a number of possible sources of such information are listed.

Now imagine that, whenever a particular situation occurs, you can spend up to 8 cents on actually looking for the information that you need. There is however a limitation: you may never spend more than 2 cents at a time on one particular source of information. How would you spend your 8 cents? Please note, you need not spend them all!

	instructions delivered with the software (in print or as a PDF)	Help that you call up from within the software	online sources (through a web browser)	shop-bought book (either your own or borrowed)	ask someone you know (colleague, family member, etc.)	other, that is: _____
1. understand how the program or part of it works, and what it can do	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>
2. look up how to carry out a particular procedure	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>
3. quickly look up a particular fact that you don't know or cannot remember right now	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>
4. find out what went wrong when there is a problem	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>	<input type="radio"/> <input type="radio"/>

Figure 58: English language version of the questionnaire

Rather than asking respondents for their preferences, a little narrative was introduced to exert some gentle pressure, prompting respondents to consider their choices more thoroughly than they might have done when presented with a more straightforward questionnaire. Printed on the questionnaire itself were the following instructions:

Below you see (from top to bottom) four situations in which you could imagine needing information on how a software package that you are working with works. From left to right, a number of possible sources of such information are listed.

Now imagine that, whenever a particular situation occurs, you can spend up to 8 cents on actually looking for the information that you need. There is however a limitation: you may never spend more than 2 cents at a time on one particular source of information. How would you spend your 8 cents? Please note, you need not spend them all!

This unusual design, it was hoped, would slow down respondents enough to make them carefully consider their answers; rather than breeze through a list of similar-looking questions.

In November 2013, the 69 students who attended class on the first day of an introductory course on web applications were presented with the Dutch language version of the questionnaire. These students were at the very beginning of their studies at an institution for short tertiary education, and the course they were taking was part of a study program in Communication and Multimedia Design. Most were under 25 years old, having enrolled in the program directly or almost directly after completing upper secondary education; 5 were between 25 and 30 years of age. None of these 69 respondents, who all returned the filled-in questionnaire, held a tertiary educational qualification.

Then, in February 2014, the English language version of the questionnaire was administered to employees at the Netherlands office of a developer of a scientific software suite for the interpretation of seismic data. This is a highly skilled area of expertise, as is reflected in these respondents' relatively high level of education. Out of the 30 professionals (as I shall refer to this group from now on) who were in the office on the particular day on which the questionnaire was administered, and who all returned the filled-in questionnaire, 11 held a doctoral degree or equivalent and a further 10 held a master's degree or equivalent. Their overall age was, inevitably, higher than that of the students studied a few months earlier.

Data Processing

First, all the filled-in questionnaires were checked for clarity and for violations of the rules that 1) on no situation that information could be needed, more than 8 'cents'—from now on, I shall refer to these as *tokens*—in total could be spent and 2) no more than 2 tokens could be spent on a particular information source in a particular situation. None of the questionnaires violated either of these rules, nor was it ever unclear how many tokens were spent.

The filled-in questionnaires were also checked for 'empty' rows, to see if there were situations in which a respondent had not spent any tokens at all. This was not the case. All respondents had allocated at least one token to at least one information source for every one of the four situations.

Separately considered were the explanations given for the information source 'other' in cases where tokens were spent on this source: as it turned out, only two 'other' sources of information were mentioned. One, exclusively mentioned by students, was 'YouTube'; the other, exclusively mentioned by professionals, was 'Help Desk'.

To allow for the data obtained from the filled-in questionnaires to be analysed in different ways, a relational database was then created which could be queried using SQL, the standard query language for such databases. In doing so, the six information sources were given a code determining whether that particular source of information is an instance of 'documentation'; where 'documentation' was

defined as *pre-recorded information that is purposefully selected and presented by a human designer to assist future human users deploying a particular tool*. Application of this definition led to the information sources ‘instructions delivered on paper or PDF’, ‘Help system’, and ‘shop-bought book’ being labelled ‘documentation’, where the remaining three categories of ‘information found online’, ‘asking somebody’, and ‘other’ (which had turned out to be either YouTube or Help Desk) were not.

The database was then filled with the data from the questionnaires. For each combination of a respondent, an information source, and a situation a value was entered into the database recording whether 0, 1 or 2 tokens (the maximum allowed) were allocated.

Finally, pertinent SQL queries were run and the results analysed as described below.

Results

In this section, to determine if there is truth in the saying that “nobody reads the documentation”, the question is looked at from a number of different angles.

First, an extension to Table 6 is drawn up, allowing for a summary of the results to be compared to earlier studies.

Then, the relative use of documentation compared to other sources of information is investigated.

Next, the results are investigated in more depth and split up by the type of information source that respondents report they refer to.

Finally, the results reported in the first three paragraphs are split up by situation in which information is sought.

Reference to documentation

The first question asked was, very simply and in line with the information presented in Table 6: What is the percentage of respondents in both groups (students and professionals) who at least sometimes, in some situations, refer to documentation? The raw data is presented in Table 7.

Table 7: Numbers of respondents who spent at least 1 token

	students (N=69)	professionals (N=30)
book	13	8

manual	48	27
Help	44	24
online	69	30
ask	59	25
other	12	7

For the professionals this percentage is 100. Out of the six information sources to which respondents were invited to allocate tokens in different situations, some were coded 'documentation' whilst others were not. None of the professionals allocated less than 1 token to the combined information sources instruction manual, Help system and book. Even if we remove the tokens allocated to books from the equation, the answer is still 100: all professionals indicate referring to the 'traditional' documentation genres of instruction manual and Help system at least some of the time.

For the students these percentages are a little lower. 10 students never refer to the instructions or the Help system, although 2 of these may occasionally refer to a shop-bought book. This means that in this group 85% consult the instructions or the Help, be it ever so occasionally, while 88.4% may consider turning to any source of information that can be seen as documentation. These numbers are summarized in Table 8 below, in a format similar to that used in Table 6.

Table 8: Use of documentation (present study)

% yes'	N	consultation of	group	year
88.4	69	Help system or instructions (in print or PDF) or shop-bought book	students	2013
85	69	Help system or instructions (in print or PDF)	students	2013
100	30	Help system or instructions (in print or PDF) or shop-bought book	professionals	2014
100	30	Help system or instructions (in print or PDF)	professionals	2014

Documentation versus other sources of information

High percentages of users indicating that under certain conditions they refer to documentation are not difficult to obtain and therefore not very meaningful. Indeed, almost the same result can be achieved if we count the percentage of respondents that indicate referring, at some time or other, to sources of information *other* than documentation: for the students as well as for the professionals this turned out to be 100%.

In order to determine the relative importance of documentation, we need to know how frequently other sources of information will be referred to when users have a choice. The next question therefore was: To which of the two groupings of information sources, documentation or not, was the highest proportion of tokens allocated? Remember that ‘documentation’ may be a shop-bought book, an instruction manual or a Help system. The raw data is presented in Table 9.

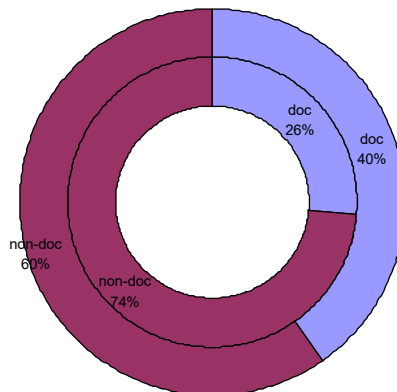


Figure 59: Proportions of tokens spent on documentation and non-documentation information sources (outer ring: professionals; inner ring: students)

Figure 59 shows a clear preference for information sources that cannot be seen as documentation. Out of a total of 944 tokens spent by the 69 students, only 248 (26%) were allocated to documentation. The 30 professionals together spent 472 tokens, 190 (40%) of which were allocated to documentation.

Reference to different information sources

Thus far, the information sources labelled ‘documentation’ and those labelled ‘not documentation’ have been lumped together. It is worth our while looking at the six sources of information separately.

One way of doing this is by asking: How is the total number of tokens spent distributed over the various information sources? (See Table 9.)

Table 9: Number of tokens spent

	students		professionals	
book	22	248	16	190
manual	107		73	
Help	119		101	
online	447	696	153	282
ask	215		113	
other	34		16	
	944		472	

As it turns out, students and professionals had the same relative preferences. Figure 60 shows how of all tokens that were allocated, by far the most are spent on the information source ‘online’, with ‘ask somebody else’ as a good second. Only then come the Help system and the instructions, followed by ‘other’ (which was YouTube for the students and Help Desk for the professionals) and finally, the shop-bought book.

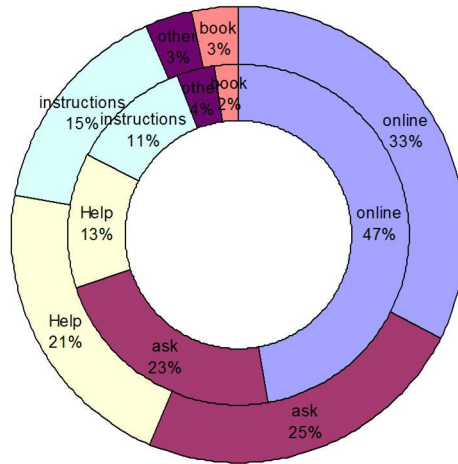


Figure 60: Proportions of tokens spent on the different information sources (outer ring: professionals; inner ring: students)

Another way of looking at the use made of the various sources of information is by counting those respondents who have allocated tokens to a particular information source in at least one situation. As there are six categories rather than one, and the percentages under consideration add up to more than one hundred, the numbers do not lend themselves to presentation in a format comparable to the earlier tables, nor the 'doughnut' chart format used thus far. Figure 61 presents the data as a simple clustered column chart.

Although direct comparison between Figure 60 and Figure 61 is difficult, we can still see that shop-bought book and 'other' are the least popular sources of information in both analyses, just as asking somebody and online browsing are the most popular. However, the differences between the information sources are smaller when we ignore the total number of tokens allocated, and the classical documentation genres of instruction manual and Help system manifest themselves much more strongly.

Of the 30 professionals, 27 (90%) would refer to the user instructions in at least one situation, and 24 (80%) would look for information in the Help system. Fewer students than professionals would ever turn to traditional documentation products, but still considerably more than half the students would in at least one situation do so: out of the 69 students, 48 (almost 70%) would sometimes consult the instructions and 44 (almost 65%) would sometimes consult the Help system.

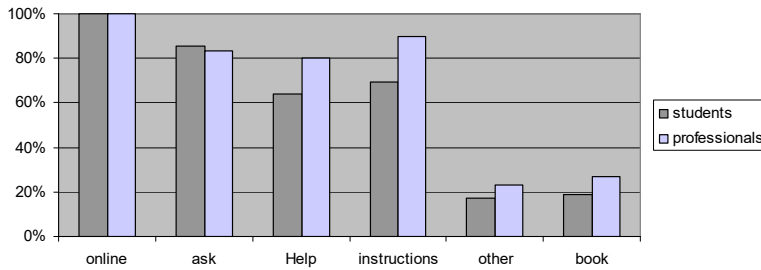


Figure 61: Percentages of respondents who consult the various information sources in at least one situation

User instructions and Help in different situations

To conclude this analysis, let us consider the relative preferences for the information sources ‘instructions’ and ‘Help’ against the different situations in which information is sought. In which situations are these traditional documentation products most frequently referred to?

Again, the different percentages do not add up to one hundred; this time because only those tokens allocated to either instructions or Help are taken into consideration while the others are ignored. Table 10 and Table 11 show the raw data; Figure 62 and Figure 63 present the data once again in the format of a set of clustered column charts.

Table 10: Number of tokens spent (students)

	how the program works	carry out a procedure	quick lookup	problem-solving
book	8	7	3	4
manual	45	36	6	20
Help	38	32	11	38
online	113	114	119	101
ask	60	57	37	61

	how the program works	carry out a procedure	quick lookup	problem- solving
other	15	9	2	8
<i>total: 944</i>	<i>279</i>	<i>255</i>	<i>178</i>	<i>232</i>

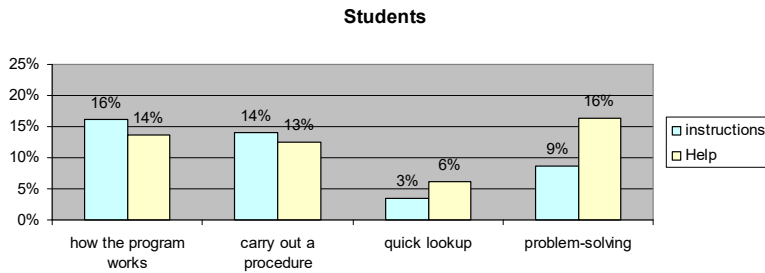


Figure 62: Per situation, proportions of tokens spent on consulting the instructions and the Help in the Students group

Table 11: Number of tokens spent (professionals)

	how the program works	carry out a procedure	quick lookup	problem- solving
book	4	9	3	0
manual	28	23	11	11
Help	26	23	23	29
online	34	38	44	37
ask	31	24	27	31
other	5	2	0	9
<i>total: 472</i>	<i>128</i>	<i>119</i>	<i>108</i>	<i>117</i>

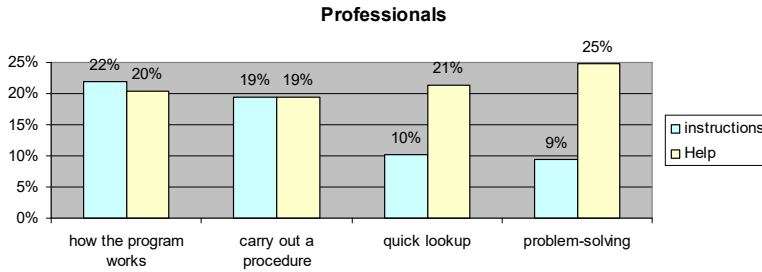


Figure 63: Per situation, proportions of tokens spent on consulting the instructions and the Help in the Professionals group

Figure 62 and Figure 63 show that for problem-solving and quick lookup of simple facts, Help is generally preferred over instructions delivered as a printed manual or as PDF. For finding out how the program works and what it can do, and for learning how to carry out a particular procedure, however, the two sources of information are almost equally popular. If anything, the instruction manual is marginally preferred to the Help.

Limitations

The study described here was based exclusively on self-reports. Doubts can be cast on the validity of the results, as there is no way of knowing whether respondents indeed interact with documentation as they say they do (Novick et al., 2007). Indeed, at least once it was found (albeit in a relatively small-scale study), that respondents' replies to the question whether they consult the documentation before starting a task or only after they have done so, bore no relationship to their actual behaviour only minutes earlier (Eiriksdottir & Catrambone, 2008).

The unusual design of the questionnaire seemed to have met its objective of forcing respondents to think before answering. No obvious patterns could be detected simply by looking at the filled-in forms: almost all respondents had spent their tokens so that the distribution over the information sources was different for each of the four situations.

However, neither the students nor the professionals spent all the tokens that they had at their disposal. Remember that every respondent was presented with four distinct situations and eight tokens to spend on looking for information in any one situation; they were explicitly told that they need not spend all eight tokens per situation. So, every respondent had a total of 32 tokens to spend. Considering that the students spent on average 13.68 tokens and the professionals 15.73, we see in Figure 64 that in both groups more than half the tokens remained unspent.

It is difficult to determine how to interpret this. Does the proportion of unallocated tokens somehow provide a measure for how often the average respondent would give up and live with a lack of information rather than look for it? Or is it so that at least some respondents have spent their tokens assigning a score of 2, 1 or 0 in order to indicate a relative preference? Are there respondents who fully expect the answer to a question in a particular situation to be found in one or two particular sources of information, and do not feel the need to spend any 'money' on other information sources? Or is the fact that not all information sources are always available taken into consideration? Informal discussion with a few of the professionals after they had filled in the questionnaire revealed that all of these explanations, and undoubtedly more, in some cases apply. It is not possible therefore to draw hard conclusions from the *absolute* numbers found; but we can conclude that users of software refer to documentation as well as to other sources of information, and that non-documentation sources are referred to *relatively* more.

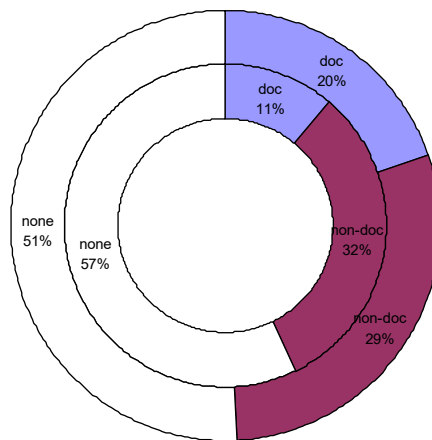


Figure 64: Percentages of tokens spent on documentation and non-documentation information sources, including unallocated tokens (outer ring: professionals; inner ring: students)

Appendix 3. Using the Repertory Grid Technique for Mining Design Patterns

Report of a 'focus group' held on 9 July 2015 during the EuroPLoP 2015 conference in Kaufbeuren, Germany. A shorter version of this Appendix is included in the conference Proceedings (van Loggem, 2015).

The act of extracting design patterns from existing design artefacts is called *pattern mining*. Pattern mining consists of distinguishing the invariant parts of existing design solutions (the pattern) from their variant parts (the current context). Attempts have been made to formally or even automatically mine design patterns in software engineering from code, but in domains where there is no formal intermediate layer between the design and its implementation, patterns are by necessity mined through the application of human judgment, in a semi-formal manner.

In the literature, a number of such semi-formal methods for pattern mining are described. The authors of (Kohls & Uttecht, 2009) first scanned their own work for possible patterns, then analyzed the work of others in the same field, which was educational interactive graphics. Next, a panel of experts validated the candidate patterns: first through direct comparison of pairs of artefacts, then through sorting the same artefacts. A different approach is described in (Iacob, 2011), where recurrent design issues were collected first during a series of workshops in which team of interaction designers created the graphical user interface for a particular application, then by analyzing a collection of existing interfaces. Those design issues that occurred the most frequently, were treated as candidate patterns. In yet another publication (Winters & Mor, 2009), the pattern mining process is described as starting from case studies, provided by designers of educational software.

In all these cases, the candidate patterns were mined by designers rather than end users. As argued earlier, however, software documentation is a field in which the end users can easily decide not to use an artefact at all; and therefore, it would be desirable to take into account the end users' preferences for certain design patterns over others. Looking at a fairly typical simple pattern structure containing the elements Problem; Solution; Use when; Don't use when; How; Why, it seems possible for the first four structure elements (Problem; Solution; Use when; Don't use when) to be provided by end users, even if the final two (How and Why) are left to expert documentation designers. To test this hypothesis, I developed an approach to pattern mining that involves end users, and conducted a simple study.

A 'focus group' was organized on 9 July 2015 during the EuroPLoP 2015 conference in Kaufbeuren, Germany. This is a conference on design patterns and pattern languages, drawing an international audience and having English as the

working language. The focus group was advertised as an attempt to find out if the Repertory Grid Technique could be used for mining patterns. No information was given beforehand on the nature of the artefacts to which the technique would be applied; and the technique itself was outlined in only a few sentences. No pre-notification was required for participating in the focus group.

Method

The method was based on the Repertory Grid Technique, originally devised as a psychological diagnostic tool, mapping out characteristics of people rather than artefacts (Kelly, 1955). However, variations on the technique have really taken off as a tool for market research (Marsden & Littler, 2000). It is easy to administer, and has the advantage of having researcher objectivity built in: in no way are respondents guided in any direction.

The first step is *triading*. A number of subsets of three elements from a larger set are presented to the respondent, who is asked to choose one that is in some way different from the other two. In this focus group, triading yielded a list of features of designed products (candidate patterns). The second step is *laddering*. The respondents are asked to indicate which pole of the identified feature is 'good' and which is 'not good', and how the feature is characterized. In this focus group, laddering yielded a first, bare-bones description of the design pattern underlying a feature. The final step is *pyramiding*. Ways are determined to get at the desired positive pole of a feature. In this focus group, pyramiding fleshed out the results of the previous step and yielded the beginnings of a full pattern description.

Execution

The group drew six participants, all male and between 30 and 40 years old. None had any experience with or even interest in the design of user documentation of any kind. After a brief introduction, they were given unrestricted access to five hardcopy documentation artefacts that had been placed on a free-standing table: three commercially available books, and two user manuals that came with a piece of software. All were written in English. On the cover of each a white sticky label was attached, with a number ranging from 1-5. The books and the numbering scheme were as follows:

1: Mastering Prezi for Business Presentations

Anderson-Williams, Russell
Packt Publishing, Birmingham, UK, 2012
ISBN 978-84969-302

2: MYOB Accounting Plus(v15): User Guide

©2006, MYOB Technology Pty Ltd

3: Google SketchUp: The Missing Manual

Grover, Chris

O'Reilly, Sebastopol, USA, 2009

ISBN 978-0-596-52146-2

4: Beginning Joomla!: From Novice to Professional

Rahmel, Dan

Apress, Berkeley, USA, 2007

ISBN 978-1-59059-848-1

5: Paint Shop Pro 7 & Animation Shop 3: Getting Started Guide

©2001, Jasc Software

Each participant was handed writing materials and one printed form listing all possible combinations of 3 out of the 5 documents. (This is $10: n! / (n-r)! * r!$ where $n=5$ and $r=3$.) Participants were then asked to individually determine and write down, for each combination of three documents, which was the 'odd one out' and why. In the Repertory Grid Technique, this step is called *triading*. The participants were also asked to note whether the characteristic thus isolated was a 'good thing' or a 'bad thing', corresponding to the *laddering* step in the Repertory Grid Technique. It was stressed that they should not look for superficial characteristics, such as whether a book has a hard or a soft cover.

Filling in the forms, repeatedly referring to the documents, took about 45 minutes. When all six participants were finished, the hand-written forms were scanned for legibility, and clarification was asked where needed.

A list was drawn up of all the design features resulting from the triading and laddering. Finally, in an adaptation of the Repertory Grid Technique step of *pyramiding*, one of these was elaborated on during a group discussion, to give the beginnings of a design pattern (often called a "patlet") according to a pre-prepared structure.

Candidate patterns

First it must be noted that two of the six participants looked for differences in the subject matter (the software being described) only. For example, participants D and E both saw 2 as different from 1 and 3, and for the same reason: "topic is business/finance related, compared to forms of 3D design" (participant D) or "math/accounting, other two are visual design" (participant E). Both participants

completed the form along these lines, never considering the design of the materials. These two participants were excluded from the analysis.

Also excluded was participant F, who agreed with D and E that 2 was different from 1 and 3 because “1 and 3 are creative; 2 is some boring accounting book” and then looked exclusively for other, although equally superficial features: “costs 34.99 \$ (the others 44.99 \$)” for example, or “contains advertisement”.

In addition, it also seemed that if for one combination of samples someone had noticed a particular feature, he looked for that feature in the next combinations as well. It is likely that a larger number of design features would have been elicited if participants had been instructed:

- not to look at subject matter;
- not to look at non-designer decisions;
- try and find something new for every combination of samples that was evaluated.

As participant A on one occasion found the ‘odd one out’ to be characterized by “no OS-specific port”, which refers to the subject matter rather than to the document design and is therefore excluded, the forms filled in by participants A, B, and C mention 29 non-unique design features of documentation products. Re-phrasing every negative feature to yield a positive one, then consolidating and standardizing for terminology, gives the unique positive design features listed below. The list is sorted by frequency of mentioning and the number of times is given that a particular feature was mentioned (a full transcription of the forms filled in by respondents A, B and C is given in the final section of this Appendix).

- cross-references [4]
- introduction to the book (rather than to the software) [4]
- tips/cautions/notes, typographically marked [3]
- chapter summaries (at the end) [3]
- chapter introductions (at the beginning) [2]
- running example [2]
- back matter [2]
- running head (perhaps containing subheader or chapter number/name) [2]
- stepwise instructions (perhaps with main action highlighted) [2]
- limited number of pages [1]
- tool icons shown in instructions [1]
- glossary [1]
- quick start section [1]
- section headings phrased as questions [1]

During the discussion of which of these features to choose for working up into a “patlet”, the group agreed that chapter summaries and chapter introductions had very similar effects on the reader. Therefore, they were treated as one, under the name Chapter Summary.

Patlet: Chapter Summary

A simple pattern structure had been pre-prepared, consisting of the following elements: 1) Problem; 2) Solution; 3) Use when; 4) Don’t use when; 5) How and 6) Why.

The identified feature (“Chapter Summary”) was filled in under Solution. Next, the Problems that a chapter summary could solve were identified as not remembering things; having skipped something without realizing; wanting to know if a chapter is interesting enough to read. The group in their capacity as end users of documentation easily came up with Use when: in narrative and in hard copy, and Don’t use when: chapters are short. It was agreed that the remaining two pattern elements, How and Why, should be left to expert documentation designers. This was in line with the hypothesis.

Problem

- Not remembering things.
- Having skipped something without realizing it.
- Wanting to know if a chapter is interesting enough to read.

Solution

Provide a chapter introduction, explaining what will be discussed in the chapter, and/or a chapter summary, describing what the reader has learned.

Use when

- Hard copy.
- Narrative.

Don’t use when

Chapters are short.

How and Why

[not discussed]

Forms filled in by respondents A, B and C

Participant A

which of the three is the 'odd one out'?			what makes it different from the other two?	is that a good thing or a bad thing?
1	2	3	running example	+
1	2	4	no tips/notes marking	-
1	2	5	no OS-specific port (BvL: refers to the subject matter)	~
1	3	4	no summary after chapters	-
1	3	5	running example	+
1	4	5	explicit questioning [BvL uncertain; almost undecipherable]	+
2	3	4	no "tips"	-
2	3	5	like a "Glossary", to look up; not read through	?
2	4	5	no 1. 2. 3. steps	?
3	4	5	no intro	-

Participant B

which of the three is the 'odd one out'?			what makes it different from the other two?	is that a good thing or a bad thing?
1	2	3	A lot of page number references (for more info) *	a bad thing and a good thing
1	2	4	see previous [BvL that is: A lot of page number references (for more info)]	see previous [BvL that is: a bad thing and a good thing]
1	2	5	see previous [BvL that is: A lot of page number references (for more info)]	see previous [BvL that is: a bad thing and a good thing]
1	3	4	Subheader on upper page heading instead of chapter number/name	a good thing
1	3	5	In step instructions the main action is highlighted	a good thing
1	4	5	Doesn't have dedicated Tip/caution/note boxes	a bad thing
2	3	4	[same as] * [BvL: refers to first combination in the form, that is: A lot of page number references (for more info)]	[same as] * [BvL: refers to first combination in the form, that is: a bad thing and a good thing]
2	3	5	Tool icons given in instructions	a good thing
2	4	5	Doesn't have any upper heading info regarding chapter number/name	a bad thing
3	4	5	It is the shortest in terms of pages	a good thing

Participant C

which of the three is the 'odd one out'?			what makes it different from the other two?	is that a good thing or a bad thing?
1	2	3	Missing introductory part with overview and "When and Why to Read"	bad
1	2	4	Has a quickstart section	good
1	2	5	Has a summary part to each chapter	good
1	3	4	Has no summary for chapters	bad
1	3	5	No back matter	bad
1	4	5	see above [BvL that is: No back matter]	see above [BvL that is: bad]
2	3	4	has a structure above chapters	good
2	3	5	see above [BvL that is: has a structure above chapters]	see above [BvL that is: good]
2	4	5	Has no "what has changed" part	bad
3	4	5	see above [BvL that is: Has no "what has changed" part]	see above [BvL that is: bad]

Appendix 4. Terms List

The overview in this Appendix lists everyday-English words and phrases that I have given a meaning that is more restricted than their standard dictionary definitions. Italics indicate a term that is itself included in the overview.

action	An action is a sequence of <i>operations</i> , undertaken one after the other in order to achieve a particular goal within the wider framework of the <i>activity's</i> overall intention.
activator software	Activator software is software in which there is <i>depth</i> in the <i>use complexity</i> at the level of <i>activities</i> .
activity	An activity is a sequence of <i>actions</i> , undertaken one after the other in order to achieve an overall intention providing a motivation.
actor software	Actor software is software in which there is <i>depth</i> in the <i>use complexity</i> at the level of <i>actions</i> , but not at that of <i>activities</i> .
ambition	The ambition is the intention with which a performer embarks on a course of action. It has a long-term and a short-term component.
assimilation bias	The assimilation bias is the tendency to actively or passively ignore newly-acquired knowledge when using a tool in order to meet an objective.
browsing	In contrast with <i>searching</i> , browsing is interaction with information sources that is driven by a general desire for as-yet unspecified knowledge related to a particular topic.
depth	The depth of a software system's <i>use complexity</i> is a measure of the degree of mapping between on the one hand the novelty brought to the work by the software, and on the other pre-known concepts that the user is already familiar with.
design	Design is purposeful endeavour resulting in a particular product for the benefit of a human performer.
design pattern	A design pattern is a description of the invariant parts of proven designs as a solution to a problem in a specific context.

design pattern language	A <i>design pattern language</i> is a number of interconnected design patterns related to a particular design discipline which share both a <i>value system</i> and an <i>organizing principle</i> .
documentation	Documentation is pre-recorded <i>information</i> that is purposefully selected and presented to assist future users deploying a particular tool.
documentation journey	The documentation journey is the search for <i>information</i> on the deployment of a particular tool.
height	The height of a software system's <i>use complexity</i> is a measure of the degree to which the software can affect an activity (in the activity-theoretical sense of the word).
information	Information is knowledge that is implicitly or explicitly formulated so as to be of value to a human being.
intention horizon	The intention horizon is the perimeter of the solution space that a user perceives to be offered by a particular tool.
interaction	An interaction is an observable behaviour, carried out by a user to direct the functioning of a software-driven system.
knowledge base	The knowledge base is the total of a user's current knowledge about the particular computer-mediated activity in which he engages.
load	The load of an <i>activity</i> is the effort that the performer experiences in carrying out the activity.
mastery	Mastery is expertise in a particular domain of tool-mediated activity, enabled by the ability to instantiate, for any actual need a complete and correct mental model of those aspects of the User Virtual Machine that are actually relevant. Mastery is evidenced by successfully completed activities at the highest possible level of abstraction that the tool allows for.
mental model	A mental model is a simplified representation in the mind of a complex system, that remains available in the knowledge base and evolves over time.

operation	An operation is a routinely carried-out observable behaviour.
operator software	Operator software is software in which there is <i>depth</i> in the <i>use complexity</i> only at the level of <i>operations</i> .
organizing principle	A <i>design pattern language</i> 's organizing principle is the format in which the individual patterns in the language are presented.
production bias	The production bias is the tendency to put short-term before long-term results when using a tool in order to meet an objective.
satisficing	Satisficing is an approach to decision-making in which no more cognitive effort is expended than that which is required to arrive at a decision that meets the current aspiration level.
searching	In contrast with <i>browsing</i> , searching is directed information seeking, undertaken to find the answer to a well-formulated question.
semantic layer	The semantic layer is that aspect of a software tool that describes the intermediate steps that the user may carry out to realize a particular result.
situation model	The situation model is an internal model of the current state of the environment as perceived, comprehended, and projected into the near future.
situation processing	Situation processing is the process by which a <i>situation model</i> is constructed.
syntactic layer	The syntactic layer is that aspect of a software tool that describes a user's choice of commands with which he directs the software's behaviour.
task layer	The task layer is that aspect of a software tool that describes the possible end results of the user's engagement with the software.

upgrading	Upgrading is undertaking <i>situation processing</i> at a higher, more effortful SRK level of cognitive performance.
use complexity	Use complexity is that part of task complexity that originates from the software rather than from other elements in the task environment, including the user.
User Virtual Machine (UVM)	The User Virtual Machine or UVM is a conceptual mechanism, containing all meaningful elements of a particular software system that the user can perceive or experience, as well as those that are not directly visible but influence others which are.
value	The value of an <i>activity</i> is the worth that the performer attaches to the situation that results from it.
value system	A <i>design pattern language's</i> value system is the conceptual framework within which justification can be expressed for the individual patterns in the language.
width	The width of a software system's <i>use complexity</i> is a measure of the number of hitherto unknown concepts (with their associated rules and interdependencies), brought to the work by the software, that are exposed to the user and with which he may choose to engage so that his choice makes a difference.

12. References

- Abran, A., Desharnais, J.-M., & Cuadrado-Gallego, J. J. (2012). Measurement and quantification are not the same: ISO 15939 and ISO 9126. *Journal of Software: Evolution and Process*, 24(5), 585-601. doi:10.1002/smr.496
- Adams-Webber, J. R. (2001). Cognitive Complexity and Role Relationships. *Journal of Constructivist Psychology*, 14(1), 43-50. doi:10.1080/107205301451353
- Agarwal, N. K., Xu, Y., & Poo, D. C. C. (2011). A context-based investigation into source use by information seekers. *Journal of the American Society for Information Science and Technology*, 62(6), 1087-1104. doi:10.1002/asi.21513
- Aiguier, M., Le Gall, P., & Mabrouki, M. (2008). *A Formal Definition of Complex Software*. Paper presented at the Proceedings of the 2008 The Third International Conference on Software Engineering Advances.
- Al-Diban, S. (2012). Mental Models *Encyclopedia of the Sciences of Learning* (pp. 2200-2204): Springer.
- Alexander, C. (1979). *The timeless way of building* (Vol. 1). New York: Oxford University Press.
- Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press.
- Alexander, P. A., Schallert, D. L., & Hare, V. C. (1991). Coming to Terms: How Researchers in Learning and Literacy Talk About Knowledge. *Review of Educational Research*, 61(3), 315-343.
- Appleton, B. (1997). Patterns and software: Essential concepts and terminology.
- Askwall, S. (1985). Computer supported reading vs reading text on paper: a comparison of two reading situations. *International Journal of Man-Machine Studies*, 22(4), 425-439.
- Atkin, C. (1973). Instrumental utilities and information seeking. In P. Clarke (Ed.), *New Models for Mass Communication Research* (pp. 205-239). Beverly Hills: Sage Publications.
- Ausubel, D. P. (1968). *Educational Psychology: A Cognitive View*. New York, NY, USA: Holt, Rinehart and Wilson, Inc.
- Baker, M. (2013). *Every Page Is Page One*. Laguna Hills, CA, USA: XML Press.
- Bates, M. J. (1989). The design of browsing and berrypicking techniques for the online search interface. *Online Information Review*, 13(5), 407-424.

- Bayman, P., & Mayer, R. E. (1988). Using Conceptual Models to Teach BASIC Computer Programming. *Journal of Educational Psychology*, 80(3), 291-298.
- Beck, K., & Cunningham, W. (1987). *Using pattern languages for object-oriented programs*. Paper presented at the OOPSLA-87.
<http://c2.com/doc/oopsla87.html>
- Belew, R. K. (2000). *Finding Out About*. Cambridge, Mass.: Cambridge University Press.
- Belkin, N. J. (1982). ASK for information retrieval: Part I. *Journal of Documentation*, 38(2), 61-71.
- Belkin, N. J. (1993). Interaction with texts: Information retrieval as information-seeking behavior. *Information Retrieval*, 93, 55-66.
- Ben-Ari, M., & Yeshno, T. (2006). Conceptual Models of Software Artifacts. *Interacting with Computers*, 18, 1336-1350.
- Benyon, D. (1992). The role of task analysis in systems design. *Interacting with Computers*, 4(1), 102-123.
- Bernhaupt, R., Winkler, M., & Pontico, F. (2009). *User interface patterns: A field study evaluation*. Paper presented at the IADIS international conference-interfaces and human computer interaction 2009.
- Bernsen, N. O. (1994). Foundations of multimodal representations: a taxonomy of representational modalities. *Interacting with Computers*, 6(4), 347-371.
- Bhavnani, S. K., & John, B. E. (1997). *From sufficient to efficient usage: an analysis of strategic knowledge*. Paper presented at the Proceedings of the SIGCHI conference on Human factors in computing systems, Atlanta, Georgia, United States.
- Bibby, P. A. (1992). Mental models, instructions and internalization. In Y. Rogers, A. Rutherford, & P. A. Bibby (Eds.), *Models in the Mind: Theory, perspective and application* (pp. 153-172). London [etc.]: Academic Press.
- Blackwell, A., & Green, T. (2003). Notational systems: the cognitive dimensions of notations framework. In J. M. Carroll (Ed.), *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science* (pp. 103-135): Morgan Kaufmann.
- Blandford, A., & Attfield, S. (2010). *Interacting with Information* Vol. 3. G. Marchionini (Ed.) *Synthesis Lectures on Human-Centered Informatics* (pp. 1-99). Retrieved from
<http://www.morganclaypool.com/doi/abs/10.2200/S00227ED1V01Y200911HCI006> doi:doi:10.2200/S00227ED1V01Y200911HCI006

- Blessing, L. T. M., & Chakrabarti, A. (2009). *DRM, a Design Research Methodology*. London: Springer Verlag.
- Bødker, S. (1991). *Through the Interface – A Human Activity Approach to User Interface Design*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Bødker, S. (1997). Computers in Mediated Human Activity. *Mind, culture, and activity*, 4(3), 149-158.
- Bødker, S., & Bertelsen, O. W. (2003). Activity Theory. In J. M. Carroll (Ed.), *HCI Models, Theories and Frameworks* (pp. 291-324). San Francisco: Morgan Kaufman Publishers.
- Bødker, S., & Christiansen, E. T. (2012). *Poetry in motion—appropriation of the world of Apps*. Paper presented at the ECCE2012, Edinburgh, Scotland, UK.
- Borchers, J. (2001). *A Pattern Approach to Interaction Design*: John Wiley and Sons.
- Bredeweg, B., Linnebank, F., Bouwer, A., & Liem, J. (2009). Garp3—Workbench for qualitative modelling and simulation. *Ecological informatics*, 4(5), 263-281.
- Brockmann, R. J. (1998). Minimalism: A Case of Information Transfer in Technical Communication. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg funnel* (pp. 375-392). Cambridge, MA, USA: The MIT Press.
- Bullock, D. H. (1982). Guiding job performance with job aids. *Training and Development Journal*, 36(9), 36-42.
- Byrd, K. S., & Caldwell, B. S. (2011). Increased memory load during task completion when procedures are presented on mobile screens. *Behaviour & Information Technology*, 30(5), 643-658.
doi:10.1080/0144929x.2010.529944
- Byström, K., & Järvelin, K. (1995). Task complexity affects information seeking and use. *Information Processing and Management: an International Journal*, 31(2), 191-213. doi:[http://dx.doi.org/10.1016/0306-4573\(94\)00041-Z](http://dx.doi.org/10.1016/0306-4573(94)00041-Z)
- Campbell, C. P. (1998). Training course/program evaluation: principles and practice. *Journal of European industrial training*, 22(8), 323-344.
- Campbell, D. J. (1988). Task Complexity: A Review and Analysis. *Academy of Management Review*, 13(1), 40-52.
- Caposecco, A., Hickson, L., & Meyer, C. (2014). Hearing aid user guides: Suitability for older adults. *International Journal of Audiology*, 53, S43-S51.
doi:10.3109/14992027.2013.832417
- Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

- Carroll, J. M. (1990). *The Nurnberg funnel: designing minimalist instruction for practical computer skill*. Cambridge, MA, USA: MIT press.
- Carroll, J. M. (2000). *Making use: scenario-based design of human-computer interactions*. Cambridge, MA: The MIT press.
- Carroll, J. M. (2014). Creating Minimalist Instruction. *International Journal of Designs for Learning*, 5(2).
- Carroll, J. M., & Rosson, M. B. (1987). Paradox of the Active User. In J. M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction* (pp. 80-111). Cambridge, MA: MIT Press.
- Case, D. O. (2007). *Looking for Information* (2 ed.). Baton Rouge: Emerald.
- Cataldo, M., & Oakhill, J. (2000). Why are poor comprehenders inefficient searchers? An investigation into the effects of text representation and spatial memory on the ability to locate information in text. *Journal of educational psychology*, 92(4), 791.
- Ceaparu, I., Lazar, J., Bessiere, K., Robinson, J., & Shneiderman, B. (2004). Determining causes and severity of end-user frustration. *International journal of human-computer interaction*, 17(3), 333-356.
- Cellier, J.-M., Eyrolle, H., & Mariné, C. (1997). Expertise in dynamic environments. *Ergonomics*, 40(1), 28-50.
- Charness, N., Tuffiash, M., Krampe, R., Reingold, E., & Vasyukova, E. (2005). The Role of Deliberate Practice in Chess Expertise. *Applied Cognitive Psychology*, 19, 151-165.
- Charnock, E., Rada, R., Stichler, S., & Weygant, P. (1994). Task-based method for creating usable hypertext. *Interacting with Computers*, 6(3), 275-287. doi:[http://dx.doi.org/10.1016/0953-5438\(94\)90016-7](http://dx.doi.org/10.1016/0953-5438(94)90016-7)
- Chaucer, G. (1391). *Treatise on the Use of the Astrolabe*. Retrieved from <http://www.chirurgeon.org/treatise.html>
- Chi, E. H.-h., Pirolli, P., Chen, K., & Pitkow, J. E. (2001). *Using information scent to model user information needs and actions and the Web*. Paper presented at the Proceedings of the SIGCHI conference on Human factors in computing systems, Seattle, Washington, United States.
- Chinn, C. A., & Brewer, W. F. (1993). The Role of Anomalous Data in Knowledge Acquisition: A Theoretical Framework and Implications for Science Instruction. *Review of Educational Research*, 63(1), 1-49.

- Cleverdon, C. W. (1970). *The effect of variations in relevance assessments in comparative experimental tests of index languages*. Retrieved from <https://dspace.lib.cranfield.ac.uk/handle/1826/967>
- Cleverdon, C. W., Mills, J., & Keen, M. (1966). *Factors determining the performance of indexing systems*. Retrieved from <https://dspace.lib.cranfield.ac.uk/handle/1826/863>
- Colaric, S. M. (2003). Instruction for Web Searching: An Empirical Study. *College & Research Libraries*, 64(2), 111-122.
- Collins, A., Seely Brown, J. S., & Newman, S. E. (1989). Cognitive Apprenticeship: teaching the craft of reading, writing, and mathematics. In L. B. Resnisk (Ed.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser* (pp. 453-494). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Connaway, L. S., Dickey, T. J., & Radford, M. L. (2011). "If it is too inconvenient I'm not going after it:" Convenience as a critical factor in information-seeking behaviors. *Library & Information Science Research*, 33(3), 179-190. doi:<http://dx.doi.org/10.1016/j.lisr.2010.12.002>
- Coplien, J. O. (1998). Software design patterns: Common questions and answers *The Patterns Handbook: Techniques, Strategies, and Applications* (pp. 311-320). New York, USA: Cambridge University Press.
- Craik, K. J. W. (1943). *The nature of explanation*. Cambridge: Cambridge University Press.
- Dautovic, A., Plosch, R., & Saft, M. (2011). *Automatic Checking of Quality Best Practices in Software Development Documents*. Paper presented at the Quality Software (QSIC), 2011 11th International Conference on.
- de Groot, A. D. (1946). *Het denken van den schaker. Een experimenteel-psychologische studie*. (PhD Thesis), Universiteit van Amsterdam, Amsterdam.
- de Jong, M. D. T., & Karreman, J. (2017). The Image of User Instructions: Comparing Users' Expectations of and Experiences with an Official and a Commercial Software Manual. *Technical communication*, 64(1), 38-49.
- de Jong, T., & Ferguson-Hessler, M. G. M. (1996). Types and Qualities of Knowledge. *Educational Psychologist*, 31(2), 105-113.
- de Kleer, J., & Brown, J. S. (1981). Mental models of physical mechanisms and their acquisition. *Cognitive skills and their acquisition*, 285-309.
- de Moel, N., & van der Veer, G. (2011). *Design pattern based decision support*.

- de Souza, J., Marconi, B., & Dyson, M. (2008). Are animated demonstrations the clearest and most comfortable way to communicate on-screen instructions? *Information Design Journal*, 16(2), 107-124.
doi:10.1075/idj.16.2.03bez
- Dearden, A., & Finlay, J. (2006). Pattern languages in HCI: A critical review. *Human-Computer Interaction*, 21(1), 49-102.
- Deng, J., Kemp, E., & Todd, E. G. (2006). *Focussing on a standard pattern form: the development and evaluation of MUIP*. Paper presented at the Proceedings of the 7th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: design centered HCI, Christchurch, New Zealand.
- Dewey, J. (1913). *Interest and effort in education*: Houghton Mifflin.
- Draper, S. W. (1998). Practical problems and proposed solutions in designing action-centered documentation. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg funnel* (pp. 349-374). Cambridge, MA, USA: The MIT Press.
- Duff, S. C., & Barnard, P. J. (1990). *Influencing behaviour via device representation; decreasing performance by increasing instruction*. Paper presented at the Proceedings of the IFIP TC13 Third International Conference on Human-Computer Interaction.
- Duggan, G. B., & Payne, S. J. (2001). Interleaving reading and acting while following procedural instructions. *Journal of Experimental Psychology: Applied*, 7(4), 297-307.
- Eiriksdottir, E., & Catrambone, R. (2008). How do People Use Instructions in Procedural Tasks? *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 52(8), 673-677.
doi:10.1177/154193120805200814
- Eiriksdottir, E., & Catrambone, R. (2011). Procedural Instructions, Principles, and Examples: How to Structure Instructions for Procedural Tasks to Enhance Performance, Learning, and Transfer. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 53(6), 749-770.
doi:10.1177/0018720811419154
- Eiriksdottir, E., & Catrambone, R. (2014). Instruction Use Depends on the Level of Details. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 58(1), 2365-2369. doi:10.1177/1541931214581492
- Ellis, D. (1989). A behavioural approach to information retrieval system design. *Journal of Documentation*, 45(3), 171-212.

- Ellis, D. (1993). A comparison of the information seeking patterns of researchers in the physical and social sciences. *Journal of Documentation*, 49(4), 356-369.
- Emerson, L., & MacKay, B. (2011). A comparison between paper-based and online learning in higher education. *British Journal of Educational Technology*, 42(5), 727-735.
- Endsley, M. R. (2000a). *Situation Models: An Avenue to the Modeling of Mental Models*. Paper presented at the 14th Triennial Congress of the International Ergonomics Association and the 44th Annual Meeting of the Human Factors and Engineering Society.
- Endsley, M. R. (2000b). Theoretical Underpinnings of Situation Awareness: a Critical Review. In M. R. Endsley & D. J. Garland (Eds.), *Situation Awareness Analysis and Measurement* (pp. 3-32). Mahwah, NJ: Lawrence Erlbaum Associates.
- Engel, J., Martin, C., Herdin, C., & Forbrig, P. (2013). Formal Pattern Specifications to Facilitate Semi-automated User Interface Generation. In M. Kurosu (Ed.), *Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments* (Vol. 8004, pp. 300-309): Springer Berlin Heidelberg.
- English, R. E., & Reigeluth, C. M. (1996). Formative research on sequencing instruction with the elaboration theory. *Educational Technology Research and Development*, 44(1), 23-42.
- Ericsson, K. A. (2005). Recent Advances in Expertise Research: A Commentary on the Contributions to the Special Issue. *Applied Cognitive Psychology*, 19, 233-241.
- Ertmer, P. A., & Newby, T. J. (1993). Behaviorism, cognitivism, constructivism: Comparing critical features from an instructional design perspective. *Performance Improvement Quarterly*, 6(4), 50-70.
- Farkas, D. K. (1999). The logical and rhetorical construction of procedural discourse. *Technical communication*, 46(1), 42-54.
- Farkas, D. K., & Williams, T. R. (1990). John Carroll's the Nurnberg funnel and minimalist documentation. *IEEE Transactions on Professional Communication*, 33(4), 182-187.
- Farrington-Darby, T., & Wilson, J. R. (2006). The nature of expertise: A review. *Applied Ergonomics*, 37, 17-32.
- Fast, K. V., & Campbell, D. G. (2004). "I still like Google": University student perceptions of searching OPACs and the web. *Proceedings of the American*

- Society for Information Science and Technology*, 41(1), 138-146.
doi:10.1002/meet.1450410116
- Fein, R. M., Olson, G. M., & Olson, J. S. (1993). A mental model can help with learning to operate a complex device *Interact '93 and CHI '93 conference companion on Human factors in computing systems* (pp. 157-158). New York: ACM Press.
- Fincher, S. (1999). *What is a pattern language*. Paper presented at the Patterns Workshop at INTERACT.
- Fincher, S. (2003). Perspectives on HCI patterns: concepts and tools (introducing PLML). *Interfaces*(56), 26-28.
- Frøkjær, E., Hertzum, M., & Hornbæk, K. (2000). *Measuring usability: are effectiveness, efficiency, and satisfaction really correlated?* Paper presented at the Proceedings of the SIGCHI conference on Human Factors in Computing Systems, The Hague, The Netherlands.
- Fu, W.-T., & Gray, W. D. (2004). Resolving the paradox of the active user: stable suboptimal performance in interactive tasks. *Cognitive Science*, 28, 901-935.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley.
- Gartenberg, M. (2005). The High Cost of Not Training. *Computerworld*, July, 11, 2005, 21.
- Gentner, D., & Stevens, A. (Eds.). (1983). *Mental Models*. Hillsdale, NJ: Erlbaum.
- Gerstberger, P. G., & Allen, T. J. (1968). Criteria used by research and development engineers in the selection of an information source. *Journal of Applied Psychology*, 52(4), 272-279.
- Geske, J., & Bellur, S. (2008). Differences in brain information processing between print and computer screens: Bottom-up and top-down attention factors. *International Journal of Advertising*, 27, 399-423.
- Glaser, R. (1985). *The nature of expertise* (107). Retrieved from Columbus:
- Goodstein, L. P., Andersen, H. B., & Olsen, S. E. (Eds.). (1988). *Tasks, Errors and Mental Models*. London, New York, Philadelphia: Taylor & Francis.
- Griffin, T. D., & Ohlsson, S. (2001). *Beliefs Versus Knowledge: A Necessary Distinction for Explaining, Predicting, and Assessing Conceptual Change*. Paper presented at the Twenty-third Annual Conference of the Cognitive Science Society, Edinburgh, UK.

- Gwizdka, J., & Spence, I. (2007). Implicit measures of lostness and success in web navigation. *Interacting with Computers*, 19(3), 357-369.
- Hackos, J. T. (1998). Choosing a minimalist approach for expert users. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg funnel* (pp. 149-177). Cambridge, MA, USA: The MIT Press.
- Halasz, F., & Moran, T. P. (1982). *Analogy considered harmful*. Paper presented at the Conference on Human Factors in Computing Systems, Gaithersburg, Maryland, United States.
- Halasz, F. G., & Moran, T. P. (1983). *Mental Models and Problem Solving in Using a Calculator*. Paper presented at the CHI '83.
- Harp, S. F., & Mayer, R. E. (1998). How seductive details do their damage: A theory of cognitive interest in science learning. *Journal of Educational Psychology*, 90(3), 414.
- Hegarty, M., & Just, M. A. (1993). Constructing mental models of machines from text and diagrams. *Journal of Memory and Language*, 32(717-742).
- Hennipman, E.-J., Oppelaar, E.-J., & van der Veer, G. (2008). Pattern Languages as Tool for Discount Usability Engineering. In T. C. N. Graham & P. Palanque (Eds.), *Interactive Systems. Design, Specification, and Verification* (Vol. 5136, pp. 108-120): Springer Berlin Heidelberg.
- Horton, W. (1991). *Secrets of user-seductive documents: wooing and winning the reluctant reader*: Society for Technical Communication.
- Hummel, H. G. K. (2005). *Design of Cueing in Multimedia Practicals*. (PhD Thesis), Open University of The Netherlands, Heerlen.
- Iacob, C. (2011). *A design pattern mining method for interaction design*. Paper presented at the Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems.
- Jansen, C. (2002). *Reflecting on Information Mapping®: does the method live up to the expectations?*
- Jansen, C., & Balijon, S. (2002). How do people use instruction guides? Confirming and disconfirming patterns of use. *Document Design*, 3(3), 195-204.
- Jansen, C., Korzilius, H., le Pair, R., & Roest, M. (2003). Testing an Information Mapping text: Does the method live up to the expectations? *Document Design*, 4(1), 48-59.
- Johnson-Laird, P. N. (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness*: Harvard University Press.

- Jonassen, D. H. (1999). Designing Constructivist Learning Environments. In C. M. Reigeluth (Ed.), *Instructional design theories and models, Volume 2: A new paradigm of instructional theory* (pp. 215-239). Mahwah, NJ: Lawrence Erlbaum Associates.
- Julien, H., & Michels, D. (2004). Intra-individual information behaviour in daily life. *Information Processing & Management*, 40(3), 547-562.
doi:[http://dx.doi.org/10.1016/S0306-4573\(02\)00093-6](http://dx.doi.org/10.1016/S0306-4573(02)00093-6)
- Kahneman, D., & Tversky, A. (1981). *The simulation heuristic*. Retrieved from
- Kaptelinin, V., & Nardi, B. A. (2006). *Acting with Technology: Activity Theory and Interaction Design*. Cambridge, MA: The MIT Press.
- Karreman, J., & Steehouder, M. F. (2004). Some effects of system information in instructions for use. *IEEE Transactions in professional communication*, 47(1), 34-43.
- Karreman, J., Ummelen, N., & Steehouder, M. F. (2005). *Procedural and Declarative Information in User Instructions: What We Do and Don't Know About These Information Types*. Paper presented at the Professional Communication Conference, 2005. IPCC 2005.
- Keller, J. M. Development and use of the ARCS model of instructional design. *Journal of instructional development*, 10(3), 2-10.
doi:10.1007/bf02905780
- Keller, J. M. (2009). *Motivational design for learning and performance: The ARCS model approach*: Springer Science & Business Media.
- Kelly, G. A. (1955). *The psychology of personal constructs* (Vol. 1). New York, USA: WW Norton and Company.
- Kemp, D. A. (1974). Relevance, pertinence and information system development. *Information Storage and Retrieval*, 10(2), 37-47.
- Kieras, D., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22(4), 365-394.
- Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science*, 8, 255-273.
- Kim, K.-S., & Sin, S.-C. J. (2011). Selecting quality sources: Bridging the gap between the perception and use of information sources. *Journal of Information Science*, 37(2), 178-188.
- Kim, S., & Soergel, D. (2005). Selecting and measuring task characteristics as independent variables. *Proceedings of the American Society for Information Science and Technology*, 42(1), n/a-n/a. doi:10.1002/meet.14504201111

- Kim, Y., & Reigeluth, C. M. (1996). *Formative Research on the Simplifying Conditions Method (SCM) for Task Analysis and Sequencing*. Paper presented at the 1996 National Convention of the Association for Educational Communications and Technology, Indianapolis.
- Kirlik, A. (2006). Cognitive Engineering: Toward a Workable Concept of Mind. In A. Kirlik (Ed.), *Adaptive Perspectives on Human-Technology Interaction* (pp. 3-9). Oxford, NY: Oxford University Press, Inc.
- Klahr, D., & Dunbar, K. (1988). Dual Space Search During Scientific Reasoning. *Cognitive Science*, 12(1), 1-48.
- Klein, G. A. (1989). Recognition-primed decisions. In W. Rouse (Ed.), *Advances in Man-Machine Systems Research* (Vol. 5, pp. 47-92). Greenwich, CT: JAI Press.
- Klein, G. A. (1993). A recognition-primed decision (RPD) model of rapid decision making. In G. A. Klein, J. Orasanu, R. Calderwood, & C. E. Zsombok (Eds.), *Decision Making in Action: Models and Methods* (pp. 138-147). Norwood, NJ, USA: Ablex Publishing Corporation.
- Klein, G. A. (2006a). Making Sense of Sensemaking 1: Alternative Perspectives. *IEEE Intelligent Systems*(4), 70-73.
- Klein, G. A. (2006b). Making Sense of Sensemaking 2: A Macrocognitive Model. *IEEE Intelligent Systems*(5), 88-92.
- Kohler, K., & Kerkow, D. (2008). Building and evaluating a pattern collection for the domain of workflow modeling tools. In J. Gulliksen, M. Borup Harning, P. Palanque, G. C. Van der Veer, & W. Janet (Eds.), *Engineering Interactive Systems* (pp. 555-566): Springer.
- Kohls, C., & Uttecht, J.-G. (2009). Lessons learnt in mining and writing design patterns for educational interactive graphics. *Computers in Human Behavior*, 25(5), 1040-1055.
- Kotzé, P., & Renaud, K. (2008). Do we practise what we preach in formulating our design and development methods? In J. Gulliksen, M. Borup Harning, P. Palanque, G. C. Van der Veer, & W. Janet (Eds.), *Engineering Interactive Systems* (pp. 567-585): Springer.
- Kotzé, P., Renaud, K., & van Biljon, J. (2008). Don't do this—Pitfalls in using anti-patterns in teaching human-computer interaction principles. *Computers & Education*, 50(3), 979-1008.
- Kuhlthau, C. (1991). Inside the search process: Information seeking from the user's perspective. *Journal of the American Society for Information Science and Technology*, 42(5), 361-371.

- Kuhlthau, C. (1993). A Principle of Uncertainty for Information Seeking. *Journal of Documentation*, 49(4), 339-355.
- Lipshitz, R., & Pras, A. A. (2005). Not Only for Experts: Recognition-Primed Decisions in the Laboratory *How Professionals Make Decisions* (pp. 91-106). Mahwah, NJ: Lawrence Erlbaum Associates.
- Loorbach, N., Karreman, J., & Steehouder, M. (2007). Adding motivational elements to an instruction manual for seniors: Effects on usability and motivation. *Technical communication*, 54(3), 343-358.
- Loorbach, N., Karreman, J., & Steehouder, M. (2013). Confidence-Increasing Elements in User Instructions: Seniors' Reactions to Verification Steps and Personal Stories. *Technical communication*, 60(3), 190-204.
- Loorbach, N., Karreman, J., & Steehouder, M. (2013). Verification Steps and Personal Stories in an Instruction Manual for Seniors: Effects on Confidence, Motivation, and Usability. *IEEE Transactions on Professional Communication*, 56(4), 294-312. doi:10.1109/TPC.2013.2286221
- Loorbach, N., Steehouder, M., & Taal, E. (2006). The Effects of Motivational Elements in User Instructions. *Journal of Business and Technical Communication*, 20(2), 177-199. doi:10.1177/1050651905284404
- MacLean, A., Young, R. M., Bellotti, V. M. E., & Moran, T. P. (1991a). Design space analysis: Bridging from theory to practice via design rationale. *Proceedings of Esprit*, 91, 720-730.
- MacLean, A., Young, R. M., Bellotti, V. M. E., & Moran, T. P. (1991b). Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6(3-4), 201-250.
- Mangen, A., Walgermo, B. R., & Brønnick, K. (2013). Reading linear texts on paper versus computer screen: Effects on reading comprehension. *International Journal of Educational Research*, 58(0), 61-68. doi:<http://dx.doi.org/10.1016/j.ijer.2012.12.002>
- Manojlović, S., & Nikolić-Popović, J. (2002). Cognitive Complexity of Schizophrenic Patients. *Acta Medica Medianae*, 41(2), 19-25.
- Marchionini, G. (1995). *Information seeking in electronic environments*. Cambridge: Cambridge University Press.
- Marchionini, G. (2010). *Information Concepts: From Books to Cyberspace Identities* (Vol. 16): Morgan & Claypool.
- Margolin, S. J., Driscoll, C., Toland, M. J., & Kegler, J. L. (2013). E-readers, Computer Screens, or Paper: Does Reading Comprehension Change Across Media Platforms? *Applied Cognitive Psychology*, 27(4), 512-519.

- Marr, D. (1982). *Vision: a computational investigation into the human representation and processing of visual information*. San Francisco: W. H. Freeman and Company.
- Marsden, D., & Littler, D. (2000). Repertory grid technique: an interpretative research framework. *European Journal of Marketing*, 34(7), 816.
- Marshall, S. P. (1995). *Schemas in Problem Solving*. Cambridge: Cambridge University Press.
- Martin, A. P., Ivory, M. Y., Megraw, R., & Slabosky, B. (2005). *Exploring the persistent problem of user assistance* (IS-TR-2005-08-01). Retrieved from Washington DC: <http://hdl.handle.net/1773/2079>
- Mayes, D. K., Sims, V. K., & Koonce, J. M. (2001). Comprehension and workload differences for VDT and paper-based reading. *International Journal of Industrial Ergonomics*, 28(6), 367-378.
doi:[http://dx.doi.org/10.1016/S0169-8141\(01\)00043-9](http://dx.doi.org/10.1016/S0169-8141(01)00043-9)
- Mehlenbacher, B. (2003). Documentation: not yet implemented, but coming soon. In J. A. Jacko & A. Sears (Eds.), *The HCI Handbook: Fundamentals, Evolving Technologies, and Emerging Applications* (pp. 527-543). Mahwah, NJ: Lawrence Erlbaum.
- Mehlenbacher, B., Wogalter, M. S., & Laughery, K. R. (2002). *On the reading of product owner's manuals: Perceptions and product complexity*.
- Mendoza, V., & Novick, D. G. (2005). *Usability over time*. Paper presented at the Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information.
- Michaels, C. F. (2003). Affordances: Four Points of Debate. *Ecological Psychology*, 15(2), 135 - 148.
- Militello, L. G., & Hutton, R. J. (1998). Applied Cognitive Task Analysis (ACTA): A practitioner's toolkit for understanding cognitive task demands. *Ergonomics*, 41(11), 1618-1641.
- Miller, G. A. (1983). Informavores. In F. Machlup & U. Mansfield (Eds.), *The study of information: Interdisciplinary messages* (pp. 111-113). New York: John Wiley & Sons.
- Mirel, B. (1998a). "Applied Constructivism" for User Documentation. *Journal of Business and Technical Communication*, 12(1), 7-49.
- Mirel, B. (1998b). Minimalism for Complex Tasks. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp. 179-218). Cambridge, MA: The MIT Press.

- Moran, T. P. (1981). The Command Language Grammar: a representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies*, 15(1), 3-50. doi:10.1016/S0020-7373(81)80022-3
- Moran, T. P., & Carroll, J. M. (1996). *Design rationale: concepts, techniques, and use*: L. Erlbaum Associates Inc.
- Nadolski, R. J., Kirschner, P. A., van Merriënboer, J. G., & Wöretshofer, J. (2005). Development of an Instrument for Measuring the Complexity of Learning Tasks. [measuring_task_complexity.pdf]. *Educational Research and Evaluation*, 11(1), 1-27.
- Nardi, B. A. (Ed.) (1996). *Context and Consciousness: Activity Theory and Human-Computer Interaction*. Cambridge, MA: The MIT Press.
- Nguyen, F., & Woll, C. A. (2006). A practitioner's guide for designing performance support systems. *Performance Improvement*, 45(9), 37-45.
- Niebuhr, S., Kohler, K., & Graf, C. (2008). Engaging patterns: Challenges and means shown by an example. In J. Gulliksen, M. Borup Harning, P. Palanque, G. C. Van der Veer, & W. Janet (Eds.), *Engineering Interactive Systems* (pp. 586-600): Springer.
- Nistor, N., Schworm, S., & Werner, M. (2012). Online help-seeking in communities of practice: Modeling the acceptance of conceptual artifacts. *Computers & Education*, 59(2), 774-784. doi:10.1016/j.compedu.2012.03.017
- Norman, D. A. (1987). Some observations on mental models *Human-computer interaction: a multidisciplinary approach* (pp. 241-244): Morgan Kaufmann Publishers Inc.
- Norman, D. A. (1988). *The psychology of everyday things*. New York: Basic Books.
- Norman, D. A. (1999). *The Invisible Computer*. Cambridge, MA: The MIT Press.
- Norman, D. A. (2010). *Living with Complexity*. Cambridge, MA: The MIT Press.
- Novick, D. G., Elizalde, E., & Bean, N. (2007). *Toward a more accurate view of when and how people seek help with computer applications*. Paper presented at the SIGDOC 2007, El Paso, TX.
http://works.bepress.com/david_novick/16/
- Novick, D. G., & Ward, K. (2006). *What users say they want in documentation*. Paper presented at the Proceedings of the 24th annual ACM international conference on Design of communication, Myrtle Beach, SC, USA.
- Noyes, J. M., & Garland, K. J. (2003). VDT versus paper-based text: reply to Mayes, Sims and Koonce. *International Journal of Industrial Ergonomics*, 31(6), 411-423. doi:[http://dx.doi.org/10.1016/S0169-8141\(03\)00027-1](http://dx.doi.org/10.1016/S0169-8141(03)00027-1)

- O'Malley, C., & Draper, S. (1992). Representation and Interaction: Are Mental Models all in the Mind? In Y. Rogers, A. Rutherford, & P. A. Bibby (Eds.), *Models in the Mind: Theory, Perspective and Application* (pp. 73-92). London: Academic Press.
- Ohlsson, S. (2005). Comparing Multiple Paths to Mastery: What is Learned? *Cognitive Science*, 29, 769-796.
- Olaverri-Monreal, C., Dlugosch, C., & Bengler, K. (2013). ManPro: Framework for the Generation and Assessment of Documentation for Nuclear Facilities. In Á. Rocha, A. M. Correia, T. D. Wilson, & K. A. Stroetmann (Eds.), *Advances in Information Systems and Technologies* (Vol. 206, pp. 849-859): Springer Berlin Heidelberg.
- Otter, M., & Johnson, H. (2000). Lost in hyperspace: metrics and mental models. *Interacting with Computers*, 13(1), 1-40.
doi:[http://dx.doi.org/10.1016/S0953-5438\(00\)00030-8](http://dx.doi.org/10.1016/S0953-5438(00)00030-8)
- Paas, F., Renkl, A., & Sweller, J. (2004). Cognitive Load Theory: Instructional Implications of the Interaction between Information Structures and Cognitive Architecture. *Instructional Science*, 32(1-2), 1-8.
doi:10.1023/B:TRUC.0000021806.17516.d0
- Payne, S. J. (1991). A descriptive study of mental models. *Behaviour and Information Technology*, 10, 3-12.
- Payne, S. J. (1992). On Mental Models and Cognitive Artefacts. In Y. Rogers, A. Rutherford, & P. A. Bibby (Eds.), *Models in the Mind: Theory, Perspective and Application*. London: Academic Press.
- Payne, S. J., & Green, T. R. (1986). Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction*, 2(2), 93-133.
- Payne, S. J., Squibb, H. R., & Howes, A. (1990). The Nature of Device Models: The Yoked State Space Hypothesis and Some Experiments With Text Editors. *Human-Computer Interaction*, 5, 415-444.
- Pirolli, P. (1997). *Computational models of information scent-following in a very large browsable text collection*. Paper presented at the Conference on Human Factors in Computing Systems, CHI '97, Atlanta, GA.
- Pirolli, P. (2006). The Use of Proximal Information Scent to Forage for Distal Content. In A. Kirlik (Ed.), *Human technology Interaction: Methods and Models for Cognitive Engineering and Human-Computer Interaction* (pp. 247-267). Oxford, NY: Oxford University Press, Inc.

- Pirolli, P., & Card, S. K. (1999). Information foraging. *Psychological Review*, 106, 643-675.
- Prinz, W. (1997). Perception and action planning. *European Journal of Cognitive Psychology*, 9(2), 129-154.
- Quinn, E. (1980). Creativity and Cognitive Complexity. *Social Behavior & Personality: an international journal*, 8(2), 213-215.
- Reason, J. (1990). *Human Error*. Cambridge: Cambridge University Press.
- Rettig, M. (1991). Nobody reads documentation. *Commun. ACM*, 34(7), 19-24.
doi:10.1145/105783.105788
- Rips, L. J. (1986). Mental muddles. In M. Brand & R. M. Harnish (Eds.), *The representation of knowledge and belief* (pp. 258-286): The University of Arizona Press.
- Rogers, Y., Rutherford, A., & Bibby, P. A. (Eds.). (1992). *Models in the Mind: Theory, Perspective and Application*. London: Academic Press.
- Rossett, A., & Gautier-Downes, J. (1991). *A Handbook of Job Aids*. San Diego, CA: Pfeiffer & Company.
- Rothkopf, E. Z. (1971). Incidental memory for location of information in text. *Journal of Verbal Learning and Verbal Behavior*, 10(6), 608-613.
doi:[http://dx.doi.org/10.1016/S0022-5371\(71\)80066-X](http://dx.doi.org/10.1016/S0022-5371(71)80066-X)
- Sanderson, P. M., & Harwood, K. (1988). The skills, rules and knowledge classification: a discussion of its emergence and nature. In L. P. Goodstein, H. B. Andersen, & S. E. Olsen (Eds.), *Tasks, Errors and Mental Models* (pp. 21-34). London, New York, Philadelphia: Taylor & Francis.
- Santhanam, R., & Sein, M. K. (1994). Improving End-user Proficiency: Effects of Conceptual Training and Nature of Interaction. *Information Systems Research*, 5(4), 378-400.
- Schobert, W., & Schümmer, T. (2006). *Supporting Pattern Language Visualization with CoPE*. Paper presented at the EuroPLOP, Irsee, Germany.
- Schriver, K. (1997). *Dynamics in document design*. New York: John Wiley & Sons.
- Schwamb, K. B. (1990). *Mental Models: A Survey*.
- Sebillotte, S. (1988). Hierarchical planning as method for task analysis: The example of office task analysis. *Behaviour & Information Technology*, 7(3), 275-293.

- Seel, N. M. (2006). Mental Models and Complex Problem Solving: Instructional Effects. In J. Elen & R. E. Clark (Eds.), *Handling Complexity in Learning Environments: Theory and Research* (pp. 43-66): Elsevier.
- Seely Brown, J. S., Collins, A., & Duguid, P. (1989). Situated Cognition and the Culture of Learning. *Educational Researcher*, 18(1), 32-42.
- Segerståhl, K., & Jokela, T. (2006). *Usability of interaction patterns*. Paper presented at the CHI '06 Extended Abstracts on Human Factors in Computing Systems, Montreal, Quebec, Canada.
- Sein, M. K., & Bostrom, R. P. (1989). Individual differences and conceptual models in training novice users. *Human-Computer Interaction*, 4(3), 197-229.
- Selber, S. A. (2010). A Rhetoric of Electronic Instruction Sets. *Technical Communication Quarterly*, 19(2), 95-117.
doi:10.1080/10572250903559340
- Sellen, A. J., & Harper, R. H. (2002). *The myth of the paperless office*: MIT press.
- Shanteau, J. (1992). The Psychology of Experts: An Alternative View. In G. Wright & F. Bolger (Eds.), *Expertise and Decision Support* (pp. 11-23). New York: Plenum Press.
- Sharp, H., Manns, M. L., & Eckstein, J. (2003). Evolving pedagogical patterns: The work of the pedagogical patterns project. *Computer Science Education*, 13(4), 315-330.
- Shayo, C., & Olfman, L. (1998). The Role of Conceptual Models in Formal Software Training. In R. Argawal (Ed.), *Proceedings of the 1998 ACM SIGCPR Conference* (pp. 242-253). New York: ACM Press.
- Shulz, T. R., Katz, J. A., & Lepper, M. R. (2001). *Clinging to Beliefs: A Constraint-satisfaction Model*. Paper presented at the Twenty-Third Annual Conference of the Cognitive Science Society, Edinburgh, UK. CogSci01.pdf
- Simon, H. A. (1955). A Behavioral Model of Rational Choice. *The Quarterly Journal of Economics*, 69(February 1955), 99-118.
- Sloutsky, V. M., & Yaras, A. S. (2000). *Problem Representation in Experts and Novices: Part 2. Underlying Processing Mechanisms*. Paper presented at the CogSci2000.
- Small, D. L. (1999). *Rethinking the book*. Massachusetts Institute of Technology.
- Smart, K. L., DeTienne, K. B., & Whiting, M. E. (1998). *Customers' use of documentation: the enduring legacy of print*. Paper presented at the Proceedings of the 16th annual international conference on Computer documentation.

- Smart, K. L., Whiting, M. E., & DeTienne, K. B. (2001). Assessing the need for printed and online documentation: A study of customer preference and use. *Journal of Business Communication*, 38(3), 285-314.
- Smith, P. A. (1996). Towards a practical measure of hypertext usability. *Interacting with Computers*, 8(4), 365-381. doi:[http://dx.doi.org/10.1016/S0953-5438\(97\)83779-4](http://dx.doi.org/10.1016/S0953-5438(97)83779-4)
- Spannagel, C., Girwidz, R., Löthe, H., Zendler, A., & Schroeder, U. (2008). Animated demonstrations and training wheels interfaces in a complex learning environment. *Interacting with Computers*, 20(1), 97-111. doi:10.1016/j.intcom.2007.08.002
- Spink, A. (2010). *Information behavior: An evolutionary instinct*. Dordrecht: Springer Verlag.
- Spink, A., & Cole, C. (2006). Human information behavior: Integrating diverse approaches and information use. *Journal of the American Society for Information Science and Technology*, 57(1), 25-35.
- Spiro, R. J., Coulson, R. L., Feltovich, P. J., & Anderson, D. K. (1988). *Cognitive Flexibility Theory: Advanced Knowledge Acquisition in Ill-Structured Domains*. Retrieved from
- Spiro, R. J., Feltovich, P. J., Jacobson, M. J., & Coulson, R. L. (1992). Cognitive flexibility, constructivism, and hypertext: Random access instruction for advanced knowledge acquisition in ill-structured domains. In T. M. Duffy & D. H. Jonassen (Eds.), *Constructivism and the technology of instruction: A conversation* (pp. 57-76). Hillsdale, NJ: Erlbaum.
- Steehouder, M. F. (1997). *Author and reader in instructions for use. The effectiveness of instructions* Paper presented at the Crossroads in communication, Piscataway, NJ.
- Steehouder, M. F. (2002). *Beyond technical documentation: users helping each other*. Paper presented at the IEEE International Professional Communication Conference, IPCC 2002, Portland, Oregon, USA. <http://doc.utwente.nl/55875/>
- Strassman, P. A. (1990). *The business value of computers: an executive's guide*. New Canaan, CT: Information Economics Press.
- Sweller, J., van Merriënboer, J. J. G., & Paas, F. (1998). Cognitive Architecture and Instructional Design. *Educational Psychology Review*, 10(3), 251-296.
- Taatgen, N., Huss, D., Dickison, D., & Anderson, J. R. (2008). The Acquisition of Robust and Flexible Cognitive Skills. *Journal of Experimental Psychology: General*, 137(3), 548-565.

- Taylor, R. S. (1962). The process of asking questions. *American Documentation*, 13(4), 391-396. doi:10.1002/asi.5090130405
- Tebeaux, E. (1997). *The Emergence of a Tradition*. Amityville, NY: Baywood Publishing Company.
- Tidwell, J. (2010). *Designing interfaces*: O'Reilly Media, Inc.
- Todd, E., Kemp, E., & Phillips, C. (2004). *What makes a good User Interface pattern language?* Paper presented at the Fifth conference on Australasian user interface.
- Tsai, W.-C., Rogers, W. A., & Lee, C.-F. (2012). *Older Adults' Motivations, Patterns, and Improvised Strategies of Using Product Manuals*.
- Ummelen, N. (1994). Procedural and declarative information: a closer look into the distinction. In M. F. Steehouder, C. Jansen, P. van der Poort, & R. Verheijen (Eds.), *Quality of technical documentation* (pp. 115-130). Amsterdam: Editions Rodopi.
- Vakkari, P. (1999). Task complexity, problem structure and information actions:: Integrating studies on information seeking and retrieval. *Information Processing & Management*, 35(6), 819-837.
- Vallacher, R. R., & Wegner, D. M. (1987). What Do People Think They're Doing? Action Identification and Human Behavior. *Psychological Review*, 94(1), 3-15.
- van der Meij, H. (1997). The ISTE Approach to Usability Testing. *IEEE Transactions on Professional Communication*, 40(3), 209-223.
- van der Meij, H. (2003). Minimalism revisited. *Document Design*, 4(3), 212-233.
- van der Meij, H., Blijleven, P., & Jansen, L. (2003). What makes up a procedure? In M. J. Albers & B. Mazur (Eds.), *Content & Complexity. Information design in software development and documentation* (pp. 129-186). Mahwah NJ: Erlbaum.
- van der Meij, H., & Carroll, J. M. (1998). Principles and Heuristics for Designing Minimalist Instruction. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp. 19-53). Cambridge, MA: The MIT Press.
- van der Meij, H., & Gellevij, M. (1998). Screen captures in software documentation. *Technical communication*, 45, 529-543.
- van der Meij, H., & Gellevij, M. (2004). The Four Components of a Procedure. *IEEE Transactions On Professional Communication*, 47(1), 5-14.

- van der Meij, H., Karreman, J., & Steehouder, M. F. (2009). Three decades of research and professional practice on printed software tutorials for novices. *Technical communication*, 56(3), 265-292.
- van der Veer, G. C., & Felt, M. A. M. (1988). Development of Mental Models of an Office System. In G. C. Van der Veer & B. Mulder (Eds.), *Human computer interactions—psychonomic aspect* (pp. 250-272). Heidelberg: Springer Verlag.
- van der Veer, G. C., Lenting, B. F., & Bergevoet, B. A. J. (1996). GTA: Groupware task analysis—Modeling complexity. *Acta Psychologica*(91), 297-322.
- van der Veer, G. C., Tauber, M. J., Waern, Y., & van Muylwijk, B. (1985). On the interaction between system and user characteristics. *Behaviour & Information Technology*, 4(4), 289-308.
doi:10.1080/01449298508901809
- van der Veer, G. C., & van Vliet, H. (2001). *The Human-Computer Interface is the System: A Plea for a Poor Man's HCI Component in Software Engineering Curricula*. Paper presented at the 14th Conference on Software Engineering Education and Training.
- van Gog, T. (2006). *Uncovering the Problem-Solving Process to Design Effective Worked Examples*. (Ph.D.), Open University of The Netherlands, Heerlen.
- van Loggem, B. E. (2007). *Paper-Based Support for Computer-Mediated Activity: a Cognitive Task Analysis*. (Master's Thesis), Twente University, Enschede. Retrieved from http://essay.utwente.nl/618/1/scriptie_van_Loggem.pdf
- van Loggem, B. E. (2012). *Assessing Use Complexity of Software: A Tool for Documentation Designers*. Paper presented at the 4th International Conference on Human-Centred Software Engineering (HCSE 2012), Toulouse, France. http://dx.doi.org/10.1007/978-3-642-34347-6_17
- van Loggem, B. E. (2013a). *Towards a Framework for Documentation Design: an Abstract Model of Computer-Mediated Activity*. Paper presented at the 31st European Conference on Cognitive Ergonomics (ECCE 2013), Toulouse, France. <http://dx.doi.org/10.1145/2501907.2501953>
- van Loggem, B. E. (2013b). *User Documentation: The Cinderella of Information Systems*. Paper presented at the 2013 World Conference on Information Systems and Technologies (WorldCIST'13), Olhão, Portugal.
http://dx.doi.org/10.1007/978-3-642-36981-0_16
- van Loggem, B. E. (2014a). *"Nobody Reads the Documentation"—True or Not?* Paper presented at the Information Behaviour Conference (ISIC 2014), Leeds, United Kingdom. www.informationr.net/ir/19-4/isic/isic03.html

- van Loggem, B. E. (2014b). *Software Documentation: a Standard for the 21st Century*. Paper presented at the International Conference on Information Systems and Design of Communication (ISDOC '14), Lisboa, Portugal. <http://dx.doi.org/10.1145/2618168.2618192>
- van Loggem, B. E. (2015). *Using the Repertory Grid Technique for mining design patterns*. Paper presented at the Proceedings of the 20th European Conference on Pattern Languages of Programs, Kaufbeuren, Germany.
- van Loggem, B. E. (2016). *What Makes a Design Pattern Language?—Value System and Organizing Principle*. Paper presented at the Proceedings of the 21st European Conference on Pattern Languages of Programs, Kaufbeuren, Germany.
- van Merriënboer, J. J. G. (1997). *Training complex cognitive skills: A four-component instructional design model for technical training*. Englewood Cliffs, NJ, USA: Educational Technology Publications.
- van Merriënboer, J. J. G., Kirschner, P. A., & Kester, L. (2003). Taking the Load Off a Learner's Mind: Instructional Design for Complex Learning. *Educational Psychologist*, 38(1), 5-13.
- Van Nimwegen, C. (2008). *The paradox of the guided user: assistance can be counter-effective*. (PhD), Universiteit Utrecht.
- van Welie, M. (2001). *Task-based User Interface Design*. (PhD), Vrije Universiteit, Amsterdam.
- Vicente, K. J., & Rasmussen, J. (1992). Ecological interface design: theoretical foundations *IEEE Transactions on Systems, Man and Cybernetics*, 22(4), 589-606.
- von Glasersfeld, E., & Massachusetts Univ, A. S. R. R. I. (1989). *An Exposition of Constructivism: Why Some Like It Radical*.
- Vromen, N., & Overduin, M. (2000). Handleiding: de Titanic onder de gebruikersondersteuning. *Tekst [blad]*, 6, 42-46.
- Vygotsky, L. (1978). *Mind in society: The development of higher psychological processes*. Cambridge, MA, USA: Harvard University Press.
- Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Cambridge, MA, USA: Harvard University Press.
- Waern, Y. (1993). Varieties of Learning to Use Computer Tools. *Computers in Human Behavior*, 9, 323-339.
- Wästlund, E., Reinikka, H., Norlander, T., & Archer, T. (2005). Effects of VDT and paper presentation on consumption and production of information:

- Psychological and physiological factors. *Computers in Human Behavior*, 21(2), 377-394. doi:<http://dx.doi.org/10.1016/j.chb.2004.02.007>
- Wegner, D. M., Vallacher, R. R., Macomber, G., Wood, R., & Arps, K. (1984). The emergence of action. *Journal of Personality and Social Psychology*, 46(2), 269.
- Wheeler, M. (2015). Martin Heidegger. *The Stanford Encyclopedia of Philosophy (Fall 2015 Edition)*. Retrieved from <http://plato.stanford.edu/archives/fall2015/entries/heidegger/>
- White, R. W., & Roth, R. A. (2008). *Exploratory Search: Beyond the Query-Response Paradigm* (Vol. 3): Morgan & Claypool Publishers.
- Wilson, T. D. (1999). Models in information behaviour research. *Journal of Documentation*, 55(3), 249-270.
- Winters, N., & Mor, Y. (2009). Dealing with abstraction: Case study generalisation as a method for eliciting design patterns. *Computers in Human Behavior*, 25(5), 1079-1088.
- Wright, D. (2008). Implicature, Pragmatics, and Documentation: A Comparative Study. *Journal of Technical Writing and Communication*, 38(1), 27-51.
- Wright, P. (1994). Quality or usability? Quality writing provokes quality reading. In M. F. Steehouder, C. Jansen, P. Van der Poort, & R. Verheijen (Eds.), *Quality of technical documentation* (pp. 7-38). Amsterdam: Rodopi.
- Wright, P. (1998). Printed instructions: Can research make a difference? In H. J. G. Zwaga, T. Boersema, & H. C. M. Hoonhout (Eds.), *Visual information for everyday use: Design and research perspectives* (pp. 45-66). London, New York, Philadelphia: Taylor & Francis.
- Wright, P., Creighton, P., & Threlfall, S. (1982). Some factors determining when instructions will be read. *Ergonomics*, 25(3), 225-237.
- Yeshno, T., & Ben-Ari, M. (2001). *Salvation for bricoleurs*. Paper presented at the Proceedings of the Thirteenth Annual Workshop of the Psychology of Programming Interest Group, Bournemouth, UK.

13. Samenvatting

Er bestaat nog geen samenhangend raamwerk waarbinnen discussie kan worden gevoerd over ontwerp en evaluatie van en onderzoek naar gebruikersdocumentatie van software. Zowel zij die de documentatie produceren als zij die hiernaar onderzoek doen hebben behoefte aan inzicht in het proces dat door documentatie wordt ondersteund, evenals hoe en onder welke voorwaarden voorgesteld documentatieproducten ingrijpen in dit proces. Dit proefschrift levert een dergelijk samenhangend kader, in de vorm van een model van computer-gemedieerde activiteit (in het Engels: Computer-Mediated Activity, afgekort tot CMA). Vervolgens wordt aangetoond dat het CMA-model een kader biedt waarbinnen ontwerpbeslissingen voor documentatieproducten zinvol geformuleerd kunnen worden, door middel van een zogeheten 'design pattern language': de Software Documentation Design Pattern Language of SDDPL.

Het eerste hoofdstuk van dit proefschrift geeft een overzicht van de inhoud. Vervolgens wordt in hoofdstuk 2 het probleem onderzocht waarvoor alle disciplines die zich bezig houden met gebruikersondersteuning een oplossing zoeken, te weten het fenomeen dat mensen er vaak niet in slagen een bepaald stuk software volledig te leren beheersen. Het ontwerp van gebruikersdocumentatie wordt neergezet als een relevant aandachtsgebied, zij het een waaraan relatief weinig aandacht wordt geschonken en waarover weinig academische literatuur beschikbaar is.

Hoofdstuk 3 bevat een literatuuronderzoek op basis waarvan de 'secundaire taak' in het ontwerp van gebruikersdocumentatie wordt beschreven: het refereren aan documentatie teneinde een bepaald programma te gebruiken om een bepaald doel te bereiken. In hoofdstuk 4 wordt dan de 'primaire taak' in detail beschreven: een model wordt gepresenteerd van de wijze waarop mensen door herhaald gebruik in de loop van de tijd leren omgaan (of niet!) met software. Dit model, het CMA-model, is samengesteld uit een aantal bestaande modellen en theorieën die menselijk gedrag beschrijven in naturalistische situaties.

De vraag dringt zich op hoe een dergelijk *generiek en theoretisch* model dat menselijke interactie met computers beschrijft, kan worden geconcretiseerd voor toepassing in *specifieke, praktische* ontwerp situaties. Waar hoofdstukken 3 en 4 waardevrije beschrijvingen gaven, poogt hoofdstuk 5 deze vraag te beantwoorden door de aandacht te verschuiven naar het doel dat documentatieproducten beogen te bereiken. De begrippen 'expertise' en 'beheersing' worden onderzocht. Niet alle software is even moeilijk te leren beheersen en termen als 'eenvoudig' en 'complex' zijn niet exact genoeg voor een bruikbare categorisatie. Een rigide categorisatie op drie onderscheiden niveaus blijkt het mogelijk te maken de complexiteit van een bepaald stuk software vast te stellen zodanig dat daaruit vereisten voor documentatie voortvloeien.

In hoofdstuk 6 wordt een mogelijke toepassing van het CMA-model gegeven, ter validatie van de bruikbaarheid van het model als een kader voor discussie van ontwerpbeslissingen op het gebied van softwaredocumentatie. Een zogeheten ‘design pattern language’ wordt gepresenteerd, waarin ontwerpen besproken kunnen worden onder verwijzing naar de onderliggende waarden uit eerdere hoofdstukken.

In de hoofdstukken 7, 8 en 9 wordt deze Software Documentation Design Pattern Language (SDDPL) in detail uitgewerkt. Design patterns voor softwaredocumentatie worden gegeven; sommige gebaseerd op voorstellen uit de bestaande academische literatuur, andere op benaderingen die in de praktijk worden aangetroffen.

Hoofdstuk 10, ten slotte, bevat een discussie van de voorafgaande hoofdstukken en trekt conclusies. Tevens wordt een agenda gepresenteerd voor verder onderzoek.

14. *Curriculum Vitae*

Brigitte Elisabeth (Brigit) van Loggem

August 11, 1957

Born in Amsterdam, the Netherlands

1979

Candidate's degree in Law (roughly equivalent to a bachelor's degree) from the University of Amsterdam (UvA), the Netherlands

1986

Licentiate's degree 'with distinction' in Technical Translation (roughly equivalent to a master's degree) from the University of Antwerp (RUCA/HIVT), Belgium

1986-2013

Independent technical writer, documentation designer, and documentation manager.

1995

Partial KHO in Computer Science (no equivalent; study load 1,000 hrs) from the Open University (OUNL), the Netherlands

2007

MSc 'cum laude' in Communication Studies from the University of Twente, the Netherlands (thesis titled *Paper-Based Support for Computer-Mediated Activity: a Cognitive Task Analysis*)

2013-now

Documentation developer at CGG GeoScience.

brigit@byteryte.nl

www.byteryte.nl

www.linkedin.com/in/brigitvanloggem