# ON DESIGNING INTERACTIVE SYSTEMS THAT SUPPORT CREATIVE PROBLEM SOLVING IN EXPERT DOMAINS

by

Christopher G. Jennings

BSc, McMaster University, 1999

MSc, McMaster University, 2002

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in the School
of
Computing Science


SIMON FRASER UNIVERSITY
Fall 2010

## APPROVAL

**Name:** Christopher G. Jennings

**Degree:** Doctor of Philosophy

**Title of Thesis:** On Designing Interactive Systems That Support Creative Problem Solving in Expert Domains

**Examining Committee:**

**Chair:** Dr. Torsten Möller
Associate Professor, Computing Science

---

Dr. Arthur E. Kirkpatrick
Senior Supervisor
Associate Professor, Computing Science

---

Dr. Robert F. Hadley
Supervisor
Professor, Computing Science

---

Dr. D. Kevin O'Neill
Supervisor
Associate Professor, Education and Technology

---

Dr. Robert F. Woodbury
Internal Examiner
Professor, Interactive Arts and Technology

---

Dr. Peter Johnson
External Examiner
Professor, Computing Science, University of Bath

**Date Defended/Approved:** December 17, 2010

# Declaration of
# Partial Copyright Licence

SFU SIMON FRASER UNIVERSITY
THINKING OF THE WORLD

# STATEMENT OF ETHICS APPROVAL

The author, whose name appears on the title page of this work, has obtained, for the research described in this work, either:

(a) Human research ethics approval from the Simon Fraser University Office of Research Ethics,

or

(b) Advance approval of the animal care protocol from the University Animal Care Committee of Simon Fraser University;

or has conducted the research

(c) as a co-investigator, collaborator or research assistant in a research project approved in advance,

or

(d) as a member of a course approved in advance for minimal risk human research, by the Office of Research Ethics.

A copy of the approval letter has been filed at the Theses Office of the University Library at the time of submission of this thesis or project.

The original application for approval and letter of approval are filed with the relevant offices. Inquiries may be directed to those authorities.

Simon Fraser University Library
Simon Fraser University
Burnaby, BC, Canada

Last update: Spring 2010

# Abstract

Finding creative solutions to ill-structured problems is integral to the work in many expert domains. A common flaw of software tools that support this kind of work is to support mainly the detailed specification of a selected solution. To extend this support to the other processes of ill-structured problem-solving, I propose ten design principles, synthesized from results in diverse fields of research. These processes emphasize generating and comparing many potential solutions. To evaluate the principles' effectiveness, I built two prototypes; quantitative and qualitative results from evaluations demonstrate benefits, including faster task completion and the consideration of a wider variety of solutions. As there is disagreement within human-computer interaction on how to conduct such broad-scoped research, I introduce a generic framework modelled on the legal system and Thagard's explanatory coherence theory to structure this evidence into a compelling argument for the principles' wider adoption.

**Keywords:** problem solving; creativity support; design cognition; design space exploration; interaction techniques; evaluation methods

*For my own bright-eyed Athena,*
*Miss Abigail Jennings*

*"You see what I have done?" he asked the ceiling, which seemed to flinch slightly at being yanked so suddenly into the conversation. "I have transformed the problem from an intractably difficult and possibly quite insoluble conundrum into a mere linguistic puzzle. Albeit," he muttered, after a long moment of silent pondering, "an intractably difficult and possibly insoluble one."*

Dirk Gently's Holistic Detective Agency
*Douglas Adams (1952–2001)*

# Acknowledgements

Here I am, writing the last few words of my dissertation. Friends, family, faculty—so many have helped me along this road. Yet there is no doubt about who should come first in a list of people to thank: my advisor, Dr. Ted Kirkpatrick. Working with him has been a deeply rewarding experience, infused with Ted's unique blend of brilliance, humour, and honesty.

Ted always pushes me to put forth my best effort—and then shows me how to make it better. Whether in my writing or my research, Ted's comments are unerringly painstaking, thought-provoking, and—usually—spot on. Many of the best ideas in this dissertation were born of those comments—though I did add a few em dashes for seasoning—and yet Ted has always made our conversations feel like a discourse between equals. It is for that respect, and the fundamental kindness and decency it springs from, that I appreciate him most. Although his odourless whiteboard markers are a close second.

On a practical level, I also owe Ted thanks for creating the statistical engine that my prototype (*XDS*) builds on. He wrote it in *Scheme*, because he has the hacker nature. Thanks, Ted. For everything. You truly are a gentleman scholar.

While Ted's fingerprints are on every page of this dissertation, each member of my committee left an indelible mark. Dr. Bob Hadley kept me from getting sloppy with my logic, and pointed me towards Thagard's work. Dr. Kevin O'Neill helped me to articulate and address the pitfalls of developing design principles and of validating claims through prototypes. Dr. Rob Woodbury was immensely helpful in setting the core topics for my depth examination, and introduced Ted and me to design space explorers. Dr. Peter Johnson did an amazingly thorough job as external examiner, and on short notice. Thank you all.

Thanks, also, to Dr. Fred Popowich. His suggestion that evaluation methods might become a contribution of my dissertation led me to expand that topic into a full chapter. And thanks to Dr. Bill Smyth for first taking a chance on me as a graduate student.

I also acknowledge the research funding provided by the National Sciences and Engineering Research Council of Canada and the Canada Foundation for Innovation.

I thank my parents and my grandmother for buying my first computer, a Commodore VIC-20, when I was seven. I became interested in computers after watching my older brother in the arcade. I was fascinated not so much with the games themselves as with understanding how they worked—and how I could make my own. For a while I tried simulating them by arranging LEGO blocks on a LEGO "screen", and I think it was when I recruited my parents as additional CPUs that they began casting about for a more practical way to encourage my burgeoning interest. The VIC-20 arrived one Christmas morn shortly thereafter; I taught myself to program and never looked back. I'm pretty sure my parents haven't touched a LEGO since. Oh, and strictly speaking I should also thank my siblings, as the VIC-20 was not wholly mine but was a communal gift. So—Shawn, Melissa—thanks for your total lack of interest in microcomputer programming.

Thanks go to my inestimable and constant friend, Mr. Brett Kilian. We bonded in our first year of university when our respective roommates experienced near-simultaneous psychotic breaks. If not for Brett's support back then, I might have dropped out—or flunked out—and never seen a second year, let alone graduate school. I also thank Mr. Chris Cooke for introducing me to the Amiga—my first GUI—and Mr. Jeff Debackere, for his appreciation of esoteric pursuits. All of you helped me discover the person I want to be.

Above all, I want to thank my wife, Dr. Sarah Jennings, my best critic and staunchest ally. When I needed to think aloud, she listened. When I needed to find the right words, she helped me look. And when I needed to write and write and write, she kept the world away. This whole adventure began when I asked Sarah if she'd like to move to British Columbia. She said "Yes" without a moment's pause—and she didn't change her mind when I told her why. In the time since, she has shown me more patience, love, and sacrifice than I deserve. She has also given me a beautiful daughter: Miss Abigail, you inspire me, you amaze me, and you help me to see wonder in the everyday. The big book is finally finished, peanut. The next time you bang on the study door and call for me, mommy won't have to say, "No, no, Abby. Daddy needs to work."

Finally, for those I have forgotten: sorry—and thanks.

# Contents

**Approval** **ii**

**Abstract** **iii**

**Dedication** **iv**

**Quotation** **v**

**Acknowledgements** **vi**

**Contents** **viii**

**List of Tables** **xiii**

**List of Figures** **xiv**

**Preface: The Modern Hephæstus** **xvi**

**1 Expert Work and Its Support** **1**

1.1 Introduction . . . . . 1

1.2 Expert Work . . . . . 2

1.3 Contemporary Support for Expert Work . . . . . 3

1.4 A Historical Overview of Expert Support Systems . . . . . 5

1.5 Improving Support for Expert Work . . . . . 8

**2 Divergence: Theoretical Background** **11**

2.1 Introduction . . . . . 11

2.2 Ill-structured Problems . . . . . 11

2.3 Factors That Influence Problem Solving . . . 13
2.4 Theoretical Approaches to Human Problem Solving . . . 19
2.4.1 Symbol Systems and Problem Spaces . . . 20
2.4.2 Situated Action . . . 21
2.5 Strategies Used by Professionals . . . 24
2.5.1 Problem Definition . . . 24
2.5.2 Solution Generation . . . 26
2.5.3 Use of Diagrams and Sketches . . . 27
2.6 Creativity Research . . . 31
2.7 Summary . . . 34

**3 Transformation: Deriving the Principles 35**
3.1 E Pluribus Decem . . . 35
3.2 The Principles . . . 35
3.2.1 Make Partial Solutions First-class Entities . . . 36
3.2.2 Support Problem and Solution Matching . . . 36
3.2.3 Allow Subjectivity and Ambiguity . . . 37
3.2.4 Prefer General, Flexible Actions and Representations . . . 38
3.2.5 Engage Multiple Ways of Doing and Thinking . . . 39
3.2.6 Support Forming and Testing Hypotheses . . . 40
3.2.7 Encourage Parallel Exploration of Breadth and Depth . . . 41
3.2.8 Provide Rich History Mechanisms . . . 41
3.2.9 Assist the Construction and Use of Repertoires . . . 42
3.2.10 Create an Effective Environment . . . 42
3.3 Next Steps . . . 43

**4 The Knot of Evaluation 44**
4.1 Introduction . . . 44
4.2 Evaluation in HCI . . . 45
4.3 Approaches to Generalization in HCI . . . 48
4.4 Outline of the Factual Matrix . . . 50

**5 A Broad Implementation of the Principles 54**
5.1 Introduction . . . 54

5.2 The Design of *XDS* . . . 55
5.3 The Domain of Experimental Design . . . 56
5.4 Examining Typical Design Processes . . . 58
5.5 Trade-offs and Consequences . . . 60
5.6 Design Space Explorers . . . 63
5.7 Other Related Work . . . 65
5.8 *XDS:* Representing the Design Space . . . 66
5.8.1 The Design Moves . . . 67
5.8.2 Three Ways to Apply Moves . . . 70
5.9 Experiment Design Representations . . . 72
5.10 *XDS* in Use . . . 78

**6 An Evaluation of *XDS*** **82**
6.1 Introduction . . . 82
6.2 *XDS* in Light of the Principles . . . 82
6.2.1 Make Partial Solutions First-class Entities (High) . . . 83
6.2.2 Support Problem and Solution Matching (Moderate) . . . 84
6.2.3 Allow Subjectivity and Ambiguity (Moderate) . . . 84
6.2.4 Prefer General, Flexible Actions and Representations (High) . . . 84
6.2.5 Engage Multiple Ways of Doing and Thinking (High) . . . 85
6.2.6 Support Forming and Testing Hypotheses (High) . . . 85
6.2.7 Encourage Parallel Exploration of Breadth and Depth (High) . . . 85
6.2.8 Provide Rich History Mechanisms (High) . . . 86
6.2.9 Assist the Construction and Use of Repertoires (Moderate) . . . 86
6.2.10 Create an Effective Environment (Low) . . . 86
6.3 A Qualitative Study . . . 87
6.3.1 Participants . . . 87
6.3.2 Methods . . . 88
6.3.3 Results . . . 89
6.4 Discussion . . . 91
6.5 Summary . . . 94

**7 Halo Menus** **95**
7.1 Introduction . . . 95
7.2 The Halo Menu Technique . . . 96
7.3 Related Work . . . 97
7.4 Implementation Choices . . . 100
7.5 Supporting Evidence . . . 103
7.6 Limitations of Halo Menus . . . 104
7.7 Summary . . . 105

**8 A Selective Implementation of the Principles** **106**
8.1 Introduction . . . 106
8.2 Games and Problem Spaces . . . 107
8.3 The Design of *Strange Eons* . . . 109
8.3.1 Document and Project Framework . . . 110
8.3.2 Plug-in Framework . . . 113
8.3.3 Game Component Framework . . . 114
8.3.4 Text Layout and Graphical Effects . . . 115
8.3.5 Deck and Board Editor . . . 117
8.4 A Specific Game: Arkham Horror . . . 120
8.5 The Design Space of Investigators . . . 122
8.6 The Investigator Editor . . . 124

**9 An Evaluation of *Strange Eons*** **130**
9.1 *Strange Eons* in Light of the Principles . . . 130
9.1.1 Make Partial Solutions First-class Entities (Low) . . . 130
9.1.2 Support Problem and Solution Matching (Moderate) . . . 132
9.1.3 Prefer General, Flexible Actions and Representations (Moderate) . . . 133
9.1.4 Engage Multiple Ways of Doing and Thinking (Moderate) . . . 133
9.1.5 Support Forming and Testing Hypotheses (Moderate) . . . 133
9.1.6 Assist the Construction and Use of Repertoires (Moderate) . . . 133
9.2 Two Studies . . . 134
9.2.1 Methods for Controlled Experiment . . . 135
9.2.2 Methods for Observational Study . . . 138
9.3 Results and Discussion . . . 139

9.4 Other Evidence . . . . . 144
9.5 Considering Individual Effects . . . . . 146

**10 Convergence: Conclusion and Future Work** **148**
10.1 Closing Arguments . . . . . 148
10.2 Limitations . . . . . 153
10.3 Open Problems . . . . . 158
10.4 Concluding Remarks . . . . . 163

**Bibliography** **165**

# List of Tables

6.1 Degrees of Support for the Design Principles in *XDS* . . . . . 83

7.1 KLM-GOMS Analyses of Halo and Linear Menu Command Selection . . . . . 99

9.1 Degree of Support for the Design Principles in the *Strange Eons* Investigator Editor . . . . . 132

9.2 Time Taken to Submit a Design After Downloading the Application . . . . . 141

9.3 Number of Inferred Design Moves Executed to Complete a Design . . . . . 141

9.4 Responses to the Consequences Group Follow-up Questions . . . . . 143

# List of Figures


1.1 A simulated animal coat pattern. . . . . 6
1.2 The *Memex* desktop. . . . . 7
1.3 The Xerox 8010 Star Information System. . . . . 7

2.1 A problem whose solution depends upon appropriate representation. . . . . 15
2.2 *In Puzzleland: Christopher Columbus Shows Some Egg Tricks*, Sam Loyd, published 1914. . . . . 15
2.3 *A Study for an Equestrian Monument*, Leonardo da Vinci, c. 1485–90. . . . . 29

4.1 The major components of the factual matrix. . . . . 50

5.1 A hypothetical document length consequence display. . . . . 62
5.2 The application of design moves in *XDS*. . . . . 71
5.3 The detail view of an experiment design. . . . . 74
5.4 An experiment design summary. . . . . 75
5.5 The editor for detectability mappings. . . . . 77
5.6 An example of an *XDS* design session. . . . . 79

7.1 Choosing a command in a halo menu. . . . . 96
7.2 A hierarchical halo menu. . . . . 102

8.1 Sample components created with *Strange Eons*. . . . . 111
8.2 Intermediate users work with a small number of files. . . . . 112
8.3 Developing a plug-in within the project system. . . . . 112
8.4 Text fitting accommodates unusually long rules with ease. . . . . 116
8.5 Text shaping can wrap text to fit arbitrary paths. . . . . 116
8.6 Tinting makes it easy to create variants of existing components. . . . . 118

8.7 Laying out a deck of cards for printing. . . . . 119
8.8 Carolyn Fern, one of the investigators included with *Arkham Horror*. . . . . 123
8.9 The investigator editor. . . . . 125

9.1 A back story that makes extensive use of symbolic tags. . . . . 131
9.2 Overall scores assigned by the expert reviewers to investigator designs from all groups. . . . . 140

10.1 The consequence display for *Arkham Horror* monsters. . . . . 162

# Preface: The Modern Hephæstus

Human beings are compelled to render their creative thoughts in physical form: to create art and record knowledge, to design useful artifacts, and to find solutions to the problems that face society. For some, this productive urge seems nearly as strong as the *re*productive one: creative people often say, for example, that they would practice their craft even if they could not earn a living doing so [60, 73, 134]. The connection between these two urges is not simple coincidence: producing art was a way for our ancestors to demonstrate their suitability as mates [60]. Moreover, these urges both fill a similar role in our lives. Both involve a struggle through difficulty to nurture something that is both unique and yet also a reflection of our own qualities—and both offer the hope of a limited kind of immortality to a species that recognizes the temporary quality of existence.

The primacy of the productive urge has been recognized since antiquity. The ancient Greeks—widely recognized as having an astonishing list of artistic, scientific, mathematical, social, and technical achievements [67, 77, 116, 237]—expressed a clear understanding of the urge in their myth of the birth of Athena, goddess of wisdom, warfare, and crafts. The Greeks told many versions of this myth [27], but it unfolds roughly as follows:

Zeus, king of the gods, had just married his first wife, the wise Titaness Mētis, when he was brought disturbing news. An oracle of Gæa prophesied that their second child would grow to usurp Zeus's throne—just as Zeus did to his own father, Cronus. Zeus, not known for restraint, was compelled to consummate the marriage anyway; but afterwards concern began to gnaw at him.

Zeus knew that his father Cronus had been the subject of a similar prophecy. Cronus decided to swallow his children to prevent their ascendancy—but Cronus had been tricked. In place of infant Zeus he was fed a swaddled stone: Zeus was hidden away, grew up in secret, and eventually fulfilled the prophecy and seized power. Not wanting to fall for a

similar ruse, Zeus decided that the best course of action was instead to swallow the would-be mother, thus preventing children altogether. He called his bride to his side and offered to play a favourite game, in which they took turns assuming different forms. But when Mētis changed herself into a buzzing fly, Zeus gobbled her up in a single mouthful. Surprisingly, Mētis did not resent this betrayal. She eventually lodged somewhere in Zeus's head, where she shared her thoughts with him.

Some time later,[1] Zeus began to suffer from terrible headaches. These got progressively worse until finally Zeus ordered Hephæstus (god of technology, among other things) to split open his head with a double-headed axe in order to relieve the pain. Hephæstus obeyed, and no sooner did he complete this impromptu craniectomy than out sprang wise and beautiful Athena from Zeus's open forehead, a grown adult wearing full battle dress and carrying a spear! But Zeus had nothing to fear from the *first* born of Mētis. Prophecy averted, bright-eyed Athena went on to become Zeus's favourite child: the only one allowed to don his aegis and wield the mighty thunderbolt [99, p. 29].

Now to understand this story as something other than a bizarre reversal of the black widow archetype, a further bit of information is required. In ancient Greek, mētis (μῆτις) means "crafty thought": a combination of cunning and wisdom. For the Greeks, this was a highly prized quality [112, p. 377]: the product of mētis could be found in many forms, from the sculptor's art to a brilliant military stratagem. Indeed, Odysseus's infamous perfidy of the Trojan horse (in which Greek soldiers gain enter the city of Troy by feigning withdrawal and then hiding within a statue that was apparently left to appease the gods), is the canonical example of mētis. Today this kind of cleverness is often described with the trite expression "thinking outside of the box".

With this understanding of mētis, an interpretation of the birth of Athena as a metaphor for the creative process emerges: Humankind, represented by Zeus, feels compelled to quench the productive urge. By internalizing mētis, the urge can be satisfied through artistry and craftsmanship [10,27], endeavours that draw thoughts from the mind into material existence (Athena, grown and armoured). The process, however, is painful: determination, stamina—and perhaps outside assistance—are required to see it through.

This description of the creative process is as relevant today as it was in antiquity. Indeed, the Athena story contains two themes that have particular relevance to this dissertation. The

[1] Presumably much longer than the usual ten month term of human pregnancy, since Hephæstus is present at Athena's birth but is the child of Zeus and a later wife, Hera.

first of these themes is the important role that technology, represented by Hephæstus's axe, can play in helping people to release their creativity productively. Much of that technology is little changed in its essential form—while the Greeks used styli to write on wax-coated tablets, today we use them to write on pressure-sensitive electronic displays. A notable difference is the invention of the stored-program computer.[2] The stored-program computer, with its ability to modify its own data and behaviour, has the potential to play the role of Hephæstus rather than the axe: software applications can be active assistants and not just passive tools.

The second, related, theme is the importance of the long and difficult process that comes before an artifact can appear in its finished form. The myth specifically describes the sudden insight that may follow a period of incubation, a phenomenon that modern scholars have been interested in since at least Poinecaré [212], although there is still no agreement as to its precise nature [55, 207, 238, 274]. But regardless of whether the products of creativity are the result of insight or not, there is a tendency to overlook the difficult early work and—like battle-ready Athena—ascribe the finished product to genius, luck, or divine inspiration. This error has been repeated, though perhaps only by accident of history, in the design of contemporary software applications that aim to support this kind of work. These applications are designed for the final task of implementing *the* solution rather than supporting the difficult prerequisite work of deciding *which* solution to pursue. This limitation is left over from a time when memory was scarce, compute cycles were expensive, and details had to be worked out offline. Today, computers are cheap and readily available. By shaking off this old mindset, support systems can be designed to support the entire process of productive work, and not just the final pangs. Who knows what progeny of Mētis may follow?

[2] Astonishingly, the Greeks themselves had at least one *fixed*-program analog computer, the *Antikythera Mechanism* [77].

# Chapter 1

# Expert Work and Its Support

## 1.1 Introduction

Computers are powerful tools for supporting and extending human ability. In moments they can calculate a table of values that would once have taken a lifetime of painstaking work. Given more time, they can tirelessly plug their way through millions of possibilities to find efficient shipping routes, schedule events, or crack enciphered messages. No human could possibly compete with a computer at completing tasks like these, because these problems play to the computer's strength: the ability to manipulate a sequence of symbols according to an unambiguous procedure, and to do so extremely quickly. But there are also many tasks that do not play to this strength: tasks where there is no one best answer, no set way to proceed, and where ambiguity, subjectivity, and nuance are defining features. Examples of such tasks abound: writing a stirring song, designing an energy-efficient home, preventing violent crime, stopping the spread of malaria in developing countries—or even just deciding what to cook for dinner.

Computers cannot complete tasks of this second kind, but they can play an important role by supporting people who do: a computer cannot choose the best plot for a novelist's next story, but the invention of word processing makes it possible to edit a manuscript as the plot evolves without labouriously retyping each draft. Support of this kind is tremendously valuable—and yet computers could still do more. A major limitation of most contemporary support tools is that they are restricted to *document editing*, that is, editing a structure that represents the final product of a task. This places the focus on the result of productive work instead of on supporting the work itself, which includes the larger processes of navigating

through the subjective choices that the computer cannot make. Since many important aspects of these larger processes are not directly represented in the end product [39,188,275], computer support systems will remain incomplete as long as this focus on final products is maintained. By shifting support to include the larger productive processes, support tools can play a meaningful role throughout the whole of productive work.

## 1.2 Expert Work

This dissertation proposes design principles for tools that include support for the larger productive processes. The focus is on support tools for a specific group, the group engaged in what I call creative problem solving in expert domains. By "expert domain", I do not mean that everyone working in the domain is an expert, but rather that the domain itself is strongly associated with specialized knowledge (expertise). For the sake of brevity, I refer to the tasks in these domains as expert work (or expert problem solving when emphasizing that aspect). Also for the sake of brevity, I will often refer to the people who work in these domains generally as "experts", but unless this is specifically contrasted with novices it should be understood that this refers to the type of work, not the worker.

Domains that can be described as expert work share at least these six characteristics:

1. The work is productive in the sense that it leads to the creation of artifacts.
2. The work requires specialized knowledge, typically obtained through formal education.
3. The work is intellectually challenging. It requires a broad range of intellectual skills and involves the gathering, organizing, analyzing, and synthesizing of information.
4. The work features creative problem solving as a central activity.
5. The skills used in the work are predominantly *non-recurrent* [181] (as opposed to the performance of well-practiced recurrent skills by, for example, pianists or tennis players).
6. The product of the work is pragmatic rather than universal; the immediate goal is not to identify an abstract truth but to answer the needs of a specific situation.

Some work categories that fit these criteria include design, engineering, writing, knowledge work, social planning, health care, education, experimental research, and law.

Other names have been used to describe the group I call expert workers (or essentially the same group). The term *designer*, interpreted liberally, is on the mark. However, for many people the mental image conjured by this term directs them away from the liberal interpretation and towards a more narrow one. Thus it is often counterproductive to describe expert work as design, unless in connection to a specific domain ("user interface design").

Schön uses *professional* where I have used expert [228]. This term captures many of the right qualities, but it can also imply ethical, legal, and remuneratory implications which do not apply equally to all of the domains I have in mind. As well, the term is also used in some domains, such as sport, that do not meet the criteria for expert work.

Two notable exclusions from the domains included in expert work are art and science. (That is, art and science in the large. There are certainly aspects that qualify as expert work; see Chapter 5 for one example.) This restriction is due mainly to the sixth of the criteria listed above. Expert work produces a specific statement about a specific (contextualized) question. In contrast, a work of art makes many different statements, all of which are acceptable so long as they come from valid interpretations of the artifact [17]; science produces general statements about specific observations.

Still, traditional views on the nature of art, science, and design can be questioned (e.g., [61, 63, 95]). The definition given here is meant as a practical guide, not as dogma. It is derived from the common features of the domains studied in the theoretical foundation of this dissertation. There will be domains that match this list yet are not amenable to algorithmic support for other reasons. This issue will be explored further in the coming pages. There are also domains (like art) that overlap significantly with expert work and yet are not a complete match for the list above. The extent to which the design principles I develop apply to such domains ultimately depends on how closely they follow the *processes* used in expert work (see Chapter 2), because it is those processes which are being supported.

## 1.3 Contemporary Support for Expert Work

Since the goal of this dissertation is to improve computer support for expert work, it is first necessary to understand the nature and limitations of current support. Contemporary experts in creative problem-solving domains have a number of powerful software tools at their disposal to support their work. Web browsers and search tools help them to discover

relevant information from an immense range of sources. Databases, spreadsheets and statistical packages support the tasks of managing and analyzing data. Word processors and presentation packages help present recommendations to others. Although these tools are of proven utility, in their current forms they share a flaw that limits their potential. All of these tools focus on end products rather than process. While the products are important, expert work consists of more than executing the detailed steps needed to produce a work product. The most challenging problems appear in the larger processes of choosing and sequencing those steps. Current software provides little support for these higher-level processes.

A useful taxonomy of expert work processes is provided by Jones [129], who describes three types of activities: divergence (the generation of possibilities, mainly to define the limits of the problem), transformation (the reuse or cross-pollination of partial solutions), and convergence (refining a solution into its final form). Contemporary applications typically lack good support for divergence and transformation. However, they do provide strong support for convergence in the form of an accumulation of small, precise edits to a representation of the end product combined with error-correction mechanisms such as undo and spelling checks.

However, the problems solved by experts are usually poorly defined, and for this reason divergence focuses on generating alternatives as a way of defining the bounds of a problem and identifying potential issues. The advent of the Web and powerful search tools has brought some support for divergence by providing a way for experts to generate seed alternatives. But experts also need to collect, compare, and combine these alternatives. Web browsers, which largely view documents as unrelated entities, provide little explicit support for these tasks. (They do provide some support for collection, by allowing the user to create bookmarks and open a link in a separate window—although the trend towards organizing documents as tabs with mutually exclusive visibility makes them more difficult to compare than the juxtaposition of windows.)

Like divergence, the activity of transformation centres around alternative solutions. But whereas the goal of divergence is to define the bounds of the solution space, the goal of transformation is to generate, compare, combine, and build on partial solutions within those bounds in order to find the form that the eventual solution will take. Managing and navigating this set of partial solutions, and finding ways to express elements of one partial solution in another, are both challenging aspects of expert work, and contemporary applications provide almost no support for these tasks because they only model a single solution.

For example, while undo/redo does allow some movement between solution states, it is not an effective tool for collecting, comparing and combining alternatives. One limitation is that it restricts movement to a single dimension of time: one cannot create a branch in the undo history without losing some alternatives. Another limitation is that, since only one alternative is accessible at a time, the burden for storing and retrieving the relevant points of comparison is left to the user. Finally, because it is intended for error correction, the granularity of the undo/redo command is based on edits and not partial solutions. It may take many applications of the command to move between two relevant states, and it is the user's responsibility to recognize when a relevant state is reached.

## 1.4 A Historical Overview of Expert Support Systems

Although current support does not routinely encompass all of the processes of expert work, the historical trend has been an expanding role for computer support. Indeed, supporting domain experts has always been a primary aim of computer technology (although recent years have also seen a surge of interest in facilitating social relationships). The earliest support systems targeted a limited set of domains: primarily science, mathematics, and military applications. The support they gave was equally limited, coming mainly in the form of the direct computation of specific results that would become part of a larger product.

This limited level of support persisted until the 1960s, due in large part to hardware limitations. For example, computing pioneer Alan Turing's last article [263], published in 1952, proposed[1] the reaction-diffusion model to describe how patterns like those seen in animal coats could arise naturally from homogenous conditions. As part of the paper, Turing used a Ferranti Mark I computer to solve the differential equations needed to generate an illustration of a sample coat pattern. Figure 1.1 shows such an illustration, but Turing's illustration was based on far fewer samples owing to memory constraints. With the modern equivalent of 1.25 KiB of primary storage, computers of this era were simply not sophisticated enough to provide broader support for expert work.

If the hardware was not up to the task, human imagination was, and some visionaries already foresaw the potential for an expanded role for computer support of expert work. In 1945, Vannevar Bush wrote a popular article in which he described a hypothetical device to

[1]It was finally verified in 2006 that Turing's reaction-diffusion morphogenesis actually occurs in nature [236].

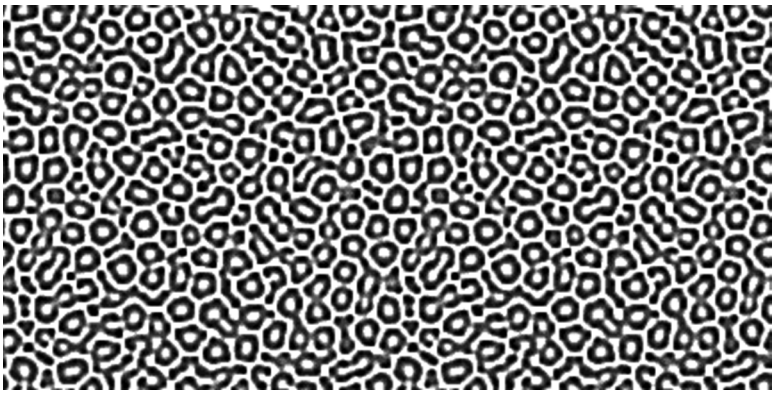


Figure 1.1: A simulated animal coat pattern. This was created using a modern implementation [122] of Turing's reaction-diffusion model of morphogenesis.

augment the human mind [33] (Figure 1.2). The device, called the *Memex* (thought to be a portmanteau of *memory index* [281]), was essentially a desktop-sized personal information processing system that could store books, personal communications, and other records, and organize them with "a mesh of associative trails" [33] analogous to what is now called hypertext. In 1960, J. C. R. Licklider [161] proposed that humans and computers combine their strengths in a real-time symbiotic partnership that would involve computers in every stage of expert work, not just data processing.

Important steps towards Licklider's vision were already being realized by 1963 in the form of Sutherland's *Sketchpad* [130,251] interactive drawing application. *Sketchpad* marked a transition from text-based interfaces to graphical ones, from batch processing to real-time interactive systems, and from tools that compute numerical results to tools that can represent the entire product of expert work. At about the same time, Englebart was establishing a research program to investigate ways that computers could support knowledge workers. This program culminated in the system known as *NLS*, which featured full-screen interactivity (as opposed to line-oriented—or none), Englebart's mouse as input device, and the first use of trees to represent the outline of a body of text [128].

While *Sketchpad* would eventually lead to modern drawing and drafting applications, the *Star Information System* [128] (see Figure 1.3) adapted the contributions of *Sketchpad* and *NLS* to produce a revolutionary system aimed at supporting knowledge workers generally. The *Star* was designed to be understood and used directly by domain experts without

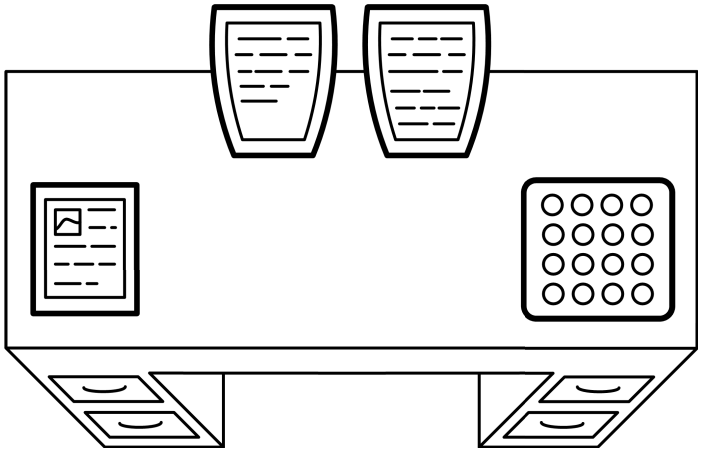


Figure 1.2: The *Memex* desktop. Notes and documents are microfilmed by placing them on a glass plate (left). Material is filed and recalled using code numbers entered with the control pad (right). Slanted viewers magnify stored microfilm records for display (top-centre). An associative link can be created between two documents displayed side-by-side. Additional microfilm reels are stored in drawers (bottom).

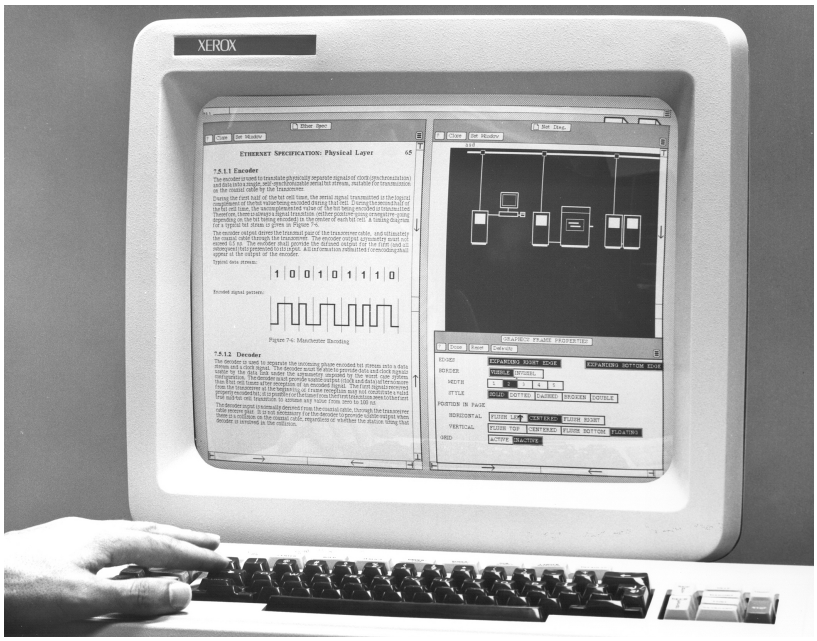



Figure 1.3: The Xerox 8010 Star Information System. *Note:* Courtesy of Xerox Corporation. Reprinted with permission.

requiring an intermediary or specialized training as a computer operator. Prominent support features include the use of domain-appropriate metaphors; generic commands; real-time interactivity; full-screen editing using the What You See Is What You Get (WYSIWYG) principle to allow the user to precisely visualize work products; integrated applications that support data sharing; and using distributed computing to support collaboration.

The *Star* had a pervasive and lasting impact on the computing industry generally and on the applications used by knowledge workers specifically. The most popular commercial word processing application today, *Microsoft Word*, is a direct descendant of the word processors of the *Star* and its experimental predecessor, the *Alto* [128]. However, while commercial applications have greatly expanded on the functionality of their predecessors in the *Star* and elsewhere, little has been done to extend the scope of their support beyond the stepwise implementation of a work product.

## 1.5 Improving Support for Expert Work

Although supporting the wider range of expert work processes has been largely neglected by commercial vendors, HCI researchers have taken steps to address this issue. To date, however, this has not produced a revolutionary shift similar to the one that marked the transition between computing specific results and constructing work products. Researchers have generally opted to limit their focus to the translation of a single concept or perspective from the literature on expert work processes into software support. For example, of the projects that have emphasized alternatives, some have focused on supporting the generation or discovery of alternatives [141,245], while others have focused on collecting and navigating alternatives [137], and still others have focused on comparing, selecting, and combining alternatives [254]. Although this piecemeal approach has been productive, it is hard to generalize from these individual results to the best overall strategies for the successful design of expert support systems.

The work presented here directly addresses this gap by considering a more holistic view of expert support. This begins with a survey of important perspectives on problem-solving and expert work in Chapter 2, including the psychology of problem cognition, strategies employed by experts in their work, and creativity research. An important purpose of this survey is to examine in more detail the processes of divergence and transformation which have traditionally received less attention than convergence. The results highlight the considerable

gulf between these two processes and the convergent process in terms of the cognitive styles used and the activities performed. That these should be distinct is not surprising considering that divergence and transformation involve primarily creative processes while convergence is largely analytical. These are qualitatively different modes of thought, although the popular media has exaggerated and mythologized some of the surrounding science, such as brain hemisphere differences [110, 223].

The review reveals several patterns and commonalities that could be exploited to support expert work, including the need to juggle many versions of a solution as it develops, the need to explore and experiment freely with ideas, the need to reconsider the problem as well as the solution, and the importance of context. In Chapter 3, these are weaved back together into a set of design principles to aid the construction of support systems. These principles should aid designers in thinking about, designing, and communicating systems that better support expert work. The rest of the dissertation is an investigation into how well the principles meet this goal. Chapter 4 considers the best way to evaluate this claim. In particular, it argues that the evaluation methods that are most readily acceptable to the HCI community are not an effective way to evaluate the kinds of broad claims made by this dissertation. Instead, it suggests an alternative framework based in part on how evidence is evaluated in legal proceedings.

Chapters 5 through 9 put this framework into practice, developing evidence to support the effectiveness of the principles. The argument is built around two systems designed in accordance with the principles. These two systems (*XDS* and *Strange Eons*) complement each other: they deal with different domains and implement the various principles to different degrees using different approaches. At the same time, the second system serves to strengthen and clarify some results from the evaluation of the first system by generalizing and reapplying one of that system's major features, *consequence displays*.

Applying the principles to the design of these systems led to the development of new interaction techniques for supporting expert work—notably halo menus and consequence displays. Moreover, small studies confirm that the systems yield benefits for users. In particular, users were found to explore more alternatives and to produce better overall outcomes as a result of using the systems. Taken together, the success of these two cases provides convincing evidence that the principles offer fruitful, effective guidance for the design of expert support systems, and that the resulting systems provide benefits to their users beyond those that could be achieved by more traditional designs.

Finally, Chapter 10 returns from particular implementations to the larger goal of helping system designers take the next step in the historic progression of supporting expert work. Some open problems are revisited, and some of the possible limitations of putting the principles into practice are discussed. Still, the overall conclusion is that designing support systems based on the principles developed here helps designers extend support for expert work from assembling a final product to helping them discover what that product will be.

# Chapter 2

# Divergence: Theoretical Background

## 2.1 Introduction

Many research areas touch on subjects related to expert work. This review focuses on work recommended by at least one of two factors: that it is a promising target for possible software support, or that it is relevant to supporting the processes of divergence and transformation.

The discussion is divided into three major topics: the psychology of problem-solving, the methods used by expert workers in practice, and results from the related field of creativity support. To a large degree, expert work involves creative problem-solving, and in particular with solving problems where both the criteria and the goals are poorly defined. An understanding of the psychology surrounding this kind of problem solving establishes limits and helps to identify areas where support may be most welcome. Looking at the methods already employed by experts provides insight into how these limitations can be worked within and around. Taken together, the result is a firm theoretical foundation for understanding and supporting expert work.

## 2.2 Ill-structured Problems

One of the most challenging aspects of expert work is solving problems that fall in the class known as *ill-structured* [219] or *wicked* [222]. Ill-structured problems are characterized by ambiguity, incomplete information, and multiple solutions. The other class of problems,

called *well-structured* or *tame*, is characterized by problems that are puzzle-like: they have well-defined goals and rules and they can be solved (at least in principle) by exhaustive search. Rittel and Webber [222] described ill-structured problems in the context of social planning. They identified four general features shared by problems in this class:

**1. There is no single goal state.**
Ill-structured problems are defined by both hard (non-negotiable) and soft (negotiable) constraints. While a problem such as a block puzzle has only one solution state, because of soft constraints ill-structured problems have multiple acceptable solutions. Soft constraints cannot be simultaneously optimal (one cannot have maximum freedom and maximum security). The task is therefore not to find the solution, but to compare solutions and choose the best compromise.

**2. There are no objective, universal metrics for the constraints.**
Stakeholders do not agree on the relative importance of competing constraints. Comparing solutions is necessary to make a selection, yet the attributes of interest cannot be objectively measured. How can we measure attributes such as happiness, quality, fairness or safety? If we could measure them, how would we agree upon how much is "enough"? The answer depends not just on objective facts, but on one's social values and the influence of others.

**3. The problem cannot be effectively isolated or decomposed.**
Ill-structured problems are never lonely: each can be considered the symptom of another ill-structured problem. For example, rather than design an experiment around a small budget, we might instead question the priorities of the budget itself. This problem-symptom duality is a consequence of soft constraints. A problem with only hard constraints has no ambiguity and is therefore well-structured: either all of the hard constraints can be met, or they cannot. Soft constraints, on the other hand, are necessarily subjective and ambiguous: choosing the best value for the constraint becomes its own ill-structured problem. The result is a web of problems that form complex feedback loops. Choosing a solution for problem A changes the constraints on its neighbours B and C. Adapting C to meet the new constraints has implications for its neighbours, including A. It also implies that ill-structured problems cannot be broken down into simpler (well-structured) subproblems. Any decomposition must include at least one ill-structured subproblem. Problem-solving strategies which rely on recursive decomposition into subproblems cannot work for ill-structured problems. Even

worse, by considering the problem in the context of its neighbours, it is possible to recursively grow the problem to include an arbitrary number of related problems.

**4. Each problem instance is unique.**
The resources available to a problem, and therefore the constraints placed upon it, vary with the context in which an ill-structured problem occurs, as do the states of the interconnected problems. Every ill-structured problem is therefore unique. Solutions are not repeatable. One can distinguish general patterns, and collect useful strategies [5] and concrete examples [228], but—unlike well-structured problems—ill-structured problems do not have a definitive answer.

Since ill-structured problems are so important to expert work, much of this dissertation is devoted to discussing them, either explicitly or implicitly. For example, the taxonomy of expert processes introduced in Chapter 1 can be used equally well as a classification scheme for ill structured problem-solving techniques [129]. This does not mean that well-structured problems do not occur in expert work, or that they are necessarily easy to solve. However, techniques for solving well-structured problems are already familiar to most computer scientists: any book on algorithm design will discuss them. Techniques for solving ill-structured problems are less likely to be familiar. Moreover, while typical techniques for solving well-structured problems are ineffective for solving ill-structured ones [222], techniques for solving ill-structured problems *can* be applied effectively to well-structured ones as well [149, 259, 265]. So while the focus may be on ill-structured problems, the presence of well-structured problems in expert work is neither forgotten nor excluded.

## 2.3 Factors That Influence Problem Solving

Both well- and ill-structured problem solving have been extensively studied by psychologists, and a number of factors that can affect the problem-solving process and its outcomes have been identified from their work. Many of these factors can be grouped together under the general category of problem representation. Both internal and external representations of a problem can dramatically affect its difficulty and the solutions that are discoverable [30, 139, 211]. This is illustrated by a well-known topographical puzzle dating back to at least the early 1900s [170, 261]. In its most general form, the puzzle consists of an $n \times n$ grid of points,

and the goal is to connect all of the points with as few straight lines as possible, without lifting the pencil. Figure 2.1 shows the puzzle and a solution for the most common $n = 3$ form. The puzzle's difficulty arises from the fact that most people incorrectly assume that the lines must intersect at one of the points, as if the points were squares on a game board. It is impossible to discover the minimal solution of $2n - 2$ lines using this representation of the problem.

It is unclear why people misrepresent the problem in this way, although a number of explanations have been offered. Some versions of the puzzle, such as the one from which the illustration in Figure 2.2 was taken, intentionally prime the solver with the wrong strategy, but this is unnecessary. Even without priming, most people are unable to solve the problem in less than ten minutes, if at all [32,272]. A common alternative to the priming explanation argues for a perceptual origin (the solver fixates on the boundary of the square induced by the law of prägnanz) [226]; evidence of a cognitive explanation has also been put forth [272].

While priming may not be be a factor in the difficulty of the nine points puzzle, there is a tendency for representations and strategies to persist from one situation to the next. Persistence effects can be both helpful and harmful. Appropriate transfer can make problem solving more efficient, but inappropriate transfer can make problem solving less efficient, lead to low quality solutions, or even prevent solving the problem at all.

This dual nature is illustrated by the phenomenon of *entrenchment*, also called *persistence of set* or the *Einstellung effect*, in which earlier problem solving sets the approach to future problems. Luchins studied this phenomenon extensively using problems that asked participants to measure a quantity $X$ of water using three jars of differing volumes $A$, $B$, and $C$ [173]. He broke participants into two groups: an experimental group that was given a series of "practice" problems whose only solution was $X = B - A - 2C$, and a control group that did not receive these problems. When participants in the experimental group were subsequently given a problem that allowed both $X = B - A - 2C$ and the simpler solution $X = A + C$, they quickly but mechanistically reproduced the more complex solution. In contrast, virtually all participants in the control group produced the simpler $A + C$ solution. The phenomenon appears to be easily manipulated. For example, Luchins found that when the experimental group was given the simple warning "don't be blind", it reduced the incidence of the suboptimal solution by more than half. Support tool designers might exploit this by making the user more aware of alternatives, either by reminding them of their possibility or by explicitly pointing them out, and thereby reduce the risk of entrenchment.

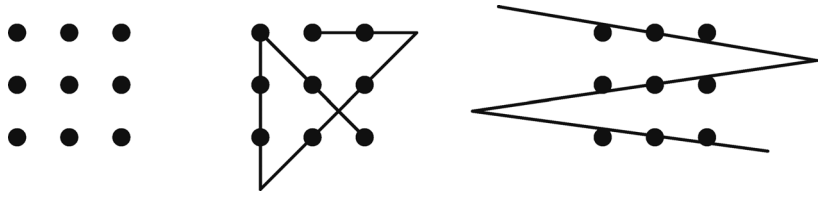


Figure 2.1: A problem whose solution depends upon appropriate representation. The nine points problem (left), and a solution (middle). A three line solution is also possible if the dots are treated as discs rather than points (right).


Figure 2.2: *In Puzzleland: Christopher Columbus Shows Some Egg Tricks*, Sam Loyd, published 1914. Black-and-white etching. This illustration accompanies Loyd's presentation of the nine points puzzle [170, p. 301]. It primes the reader with an incorrect representation in an effort to make the problem harder to solve. *Note:* From S. Loyd's *Cyclopedia of Puzzles* (The Lamb Publishing Company, 1914). Public domain.

Related to entrenchment is another source of interference due to prior experience called *functional fixedness* [59]. This refers to the tendency to fixate on one function of an object while ignoring less obvious uses. The result is an incomplete problem representation. For example, a number of verbal puzzles turn on the ambiguous use of words with multiple meanings where an incorrect meaning is more salient than the correct alternative. Here is a characteristic example (pause after reading it if you want to give it a try):

> A woman married ten different men from her home town. She has never been divorced, and all of the men are still living, yet she has never broken any laws. Explain how this is possible.

As is often the case with problems involving functional fixedness, this example is insoluble as long as the fixation holds (in this case, on the most salient meaning of *married*), but it is obvious once the fixation is broken.

Like entrenchment, functional fixedness is sensitive to context. In one set of experiments, simple changes to the verbal label used to describe an object (e.g., "tacks" versus "box of tacks") significantly increased the number of solutions discovered by participants when the label suggested relevant functions [85, 86]. This suggests that support systems should take care that the affordances offered by their objects are easily perceived [83, 84, 201] (see p. 22). This may be difficult, however, since ill-structured problem solving often involves finding unexpected applications for resources. Alternatively, designers might seek to avoid making restrictive affordances preferentially salient.

Another factor of problem difficulty related to representation is the ability to detect isomorphisms. Problems that are isomorphic have a common underlying structure but different representations. As a result, a solution for one problem can be adapted into a solution for any of its isomorphs if the common structure is recognized. For example, in computing science the members of the well-known class of *NP-complete* problems [81] are conjectured to be polynomial-time isomorphic to each other [18, 104]. While there is no practical (sub-exponential time) algorithm that can solve NP-complete problems, if a practical algorithm could be found for just one of them, the conjecture states that for any of the other problems there exists a polynomial-time conversion for the input that would allow the second problem to be solved using the more efficient algorithm.

Rigorous isomorphism is a characteristic of well-structured problems. However, ill-structured problems may have partial isomorphs, and the analogical transfer of solutions

between partial isomorphs can still be successful as long as the partial isomorphs have at least as much structural overlap as structural difference [227]. This is not that tall a hurdle, so it is not surprising that analogical transfer is also an important component of ill-structured problem solving [271].

Isomorphism and analogical transfer are of interest because even modest changes in the representation of a problem can dramatically change its difficulty [139]: difficult problems may be solved indirectly through easier (partial) isomorphs. In practice, though, it is also difficult to recognize the common structure linking isomorphic problems [216, 217], so the problem may be no easier to solve overall.

Papert [206] and Kay [130,131] suggest that the key to successful problem representation is to provide a context in which the most salient representations are also the most relevant. They respectively link problem representation to Piaget's [211] and Bruner's [29,30] theories of cognitive development.

Jean Piaget developed a comprehensive theory of human intelligence, but he is known primarily for his four-staged theory of cognitive development [211]: The *sensorimotor stage* lasts from birth to about eighteen months. In this stage, infants construct an understanding of the world by coordinating sensory input with physical action. By the end they understand that there are objects separate from themselves and that objects have permanence (that is, they continue to exist even when they cannot be seen). The sensorimotor stage is followed by the *preoperational stage*, which lasts until about age seven. The child begins speaking, and learns to represent objects using images and words. Rudimentary reasoning develops, but the child makes many logical errors, such as centration (incorrectly focusing on one aspect of a problem to the exclusion of others) and transductive reasoning (for example, fearing that because water is sucked down a bathtub drain, they will be as well). The next stage is the *concrete operational stage*, which lasts until about age 11. During this stage, the child learns to make appropriate use of logic, but can still only reason about concrete objects. Entry into the last stage, the *formal operational stage*, is marked by the acquisition of abstract reasoning.

For Papert, Piaget emphasizes that children can only construct new kinds of reasoning using the kinds of reasoning currently available to them [206, pp. 156–176]. The key to teaching a "difficult" concept is thus not to wait until a child has reached an appropriate developmental stage, but rather to represent the concept in a manner accessible to the child's *current* developmental stage. For example, Papert discusses how young children can

apply their understanding of their own body motion to draw a circle using turtle graphics in his *LOGO* programming language, and through this process learn about the differential equations which underlie calculus [206, pp. 63–66].

Kay, one of the researchers behind the *Star Information System* (see p. 6), also considers the link between difficulty, representation, and cognitive development, but he emphasizes the range of representations used by experts. For his starting point, Kay prefers Bruner over Piaget. Bruner confirmed and extended Piaget's cognitive development experiments, but in place of Piaget's stages he proposed three sequentially-developing modes of representation: *enactive* (action-based), *iconic* (image-based), and *symbolic* (language-based) [29]. Following Bruner, Kay emphasizes that moving into a new stage adds new modes of representation rather than replacing old ones [130], and he connects this to Hadamard's work on mathematical discovery.

Hadamard [97] surveyed famous mathematicians and physicists, asking them to introspect on the processes they employed in their work. Analyzing the results, Hadamard noted that very few reported working directly with symbolic structures such as sentences or equations. Instead, most worked primarily visually, while a few also reported using kinesthetic methods. Like Hadamard, Kay is particularly interested in Albert Einstein's reply [97, pp. 142–143], which alludes to all three of Bruner's representational modes:

> Words or the language, as they are written or spoken, do not seem to play any role in my mechanism of thought. The psychical entities which seem to serve as elements in thought are certain signs and more or less clear images which can be "voluntarily" reproduced and combined .... There is, of course, a certain connection between those elements and relevant logical concepts. It is also clear that the desire to arrive finally at logically connected concepts is the emotional basis of this rather vague play with the above mentioned elements. But ... this combinatory play seems to be the essential feature in productive thought—before there is any connection with logical construction ... which can be communicated to others .... The above mentioned elements are, in my case, of visual and some of muscular type. Conventional words or other signs have to be sought for laboriously only in a secondary stage, when the mentioned associative play is sufficiently established and can be reproduced at will .... In

> a stage when words intervene at all, they are, in my case, purely auditive, but they interfere only in a secondary stage as already mentioned.

From these sources, Kay concludes that computer applications should support working with all three representational modes [130]. The design of the *Star* put this into practice in the form of the mouse (kinesthetic), the graphical user interface (visual), and end-user programming (symbolic, and using Kay, Ingalls, and Goldberg's *Smalltalk* [87] programming language) [130,131]. Despite decades of tinkering, these three modalities are still represented in the modern graphical user interface in much the same way as in the *Star*: while the success of these interfaces cannot be ascribed to any one idea, this does suggest that Kay has hit upon a fundamental aspect of how people work.

Kay does not consider whether Hadamard's study of mathematicians and physicists generalizes to practitioners in other fields, but there is some evidence to support this. The extensive use of sketching by many experts can be explained in part by the need to externalize visual representations as an aid to memory (see Section 2.5.3 later in this chapter). And in author Stephen King's introduction to *Skeleton Crew* [135], King reports using primarily visual representations in the early stages of writing fiction. Written in King's conversational style, his introduction makes for a curious but congruent juxtaposition with Einstein's letter to Hadamard.

Although Kay focuses on modalities, it is also notable that Einstein's reply makes explicit reference to different work processes. His description of an early stage involving "combinatorial play" with "visual" and "muscular" signs followed by a secondary stage in which ideas are refined into communicable language is consistent with Jones's divergent and transformational processes being followed by a final convergent process (see p. 4). This suggests that not only are Bruner's representational modes significant to expert work, but that visual, and to a lesser extent kinesthetic, representations may be most important during divergence and transformation, while symbolic representations are more important during convergence.

## 2.4 Theoretical Approaches to Human Problem Solving

Besides studying individual factors that affect problem solving, a number of researchers have contributed to broader theories of the problem-solving process. Two approaches that have been particularly influential are Newell and Simon's symbolic cognition theory [194–197] and Suchman's situated action model [246]. There is a long-standing debate between

proponents of the symbolic view of cognition and proponents of situated action (e.g., [46,74, 93, 247, 268–270, 273]), and more generally with other dynamic systems explanations, such as connectionism (e.g., [6, 75, 175, 198]). While this debate is of critical importance when trying to explain or replicate human intelligence, here it is of secondary interest because both views have practical applications that do not rely on their strict correctness.

### 2.4.1 Symbol Systems and Problem Spaces

The first theory, Newell and Simon's, developed from laboratory experiments using well-structured problems, such as Édouard Lucas's Tower of Hanoi [171, 172]. It codeveloped with the classical view of artificial intelligence, which emphasizes symbol processing using list structures and the recursive subdivision of plans. Accordingly, both of these are central features of the theory.

Newell and Simon proposed two central hypotheses [197]. The first of these, the *physical symbol system hypothesis*, claims that symbolic processing is both necessary and sufficient for general intelligent action. Here, a symbol is a physical pattern that can act as a reference to some object, while a physical symbol system consists of symbols formed into structures using a set of relations [268]. Inputs to the system must be encoded as symbols, creating a separation layer between the processes of intelligence and the outside world [214,273]. Newell and Simon state [194,197] that physical symbol systems are universal Turing machines [209, 262] (and specifically compare them to interpreters for the *Lisp* [177,241] symbol-processing language [197]). The implication is that computers are capable of intelligent action, and that all intelligent action is the result of computational processes.

The second of Newell and Simon's hypotheses, called the *problem space hypothesis*, describes problem solving as a planning process in which an actor searches a space of problem states that are generated by applying operators. The selection of operators is guided by heuristics such as alternating between working backward from the goal state and forwards from the start state. The *forward chaining* heuristic allows experts to move directly towards the goal using knowledge gained through experience [145]. When stuck, the *generate-and-test* heuristic is used, meaning that moves are tried (essentially at random) and the results tested to see if the state has changed in a useful way. The most important of Newell and Simon's heuristics is *means-ends analysis*, in which actions are chosen by identifying differences between the current and goal states and then choosing actions that reduce those differences. If a chosen action cannot be performed from the current state,

then a subproblem is introduced with the goal of transforming the current state into one where the action does apply.

Although the physical symbol system hypothesis is the more significant—and certainly the more controversial—of the claims, it is the second which has had the most direct influence on expert problem solving. Problem solving as a search in a space of problem states has been widely adopted as a useful metaphor to visualize and discuss problem solving as a process. Moreover, an entire class of support systems, design space explorers, are distinguished by their use of explicit computational models of problem spaces [277]. (Design space explorers are discussed in more detail in Section 5.6; see p. 63.)

Although the metaphor is useful, the search mechanisms suggested by Newell and Simon reflect the theory's basis in well-structured problem solving. In particular, the reliance on means-ends analysis is not a good fit with ill-structured problems since it requires a specific solution state. Ill-structured problems do not have a single solution, or even a fixed number of solutions: there is no goal to climb towards or work back from. Solving ill-structured problems requires finding destinations as well as finding paths.

A more general problem for computer representations of problem spaces is that human problem solvers sometimes respond to a lack of progress by *reframing* the problem. This could mean choosing an easier partial isomorph, or attempting to solve a metaproblem such as proving that no solution is possible. (The latter technique is especially important to the resolution of paradoxes [111, pp. 196–7].) While the programs constructed by Newell and Simon were given a problem space definition *already known* to be sufficient to solve the problem, a general intelligent actor needs to be able to define its own problem spaces. However, creating programs that exhibit this kind of *transformational creativity* remains a challenge for researchers (see Section 2.6 later in this chapter).

### 2.4.2 Situated Action

Situated action offers an anthropological perspective on how humans deal with problem solving in an unpredictable and incompletely specified world. Suchman coined the term to emphasize that action and context are inseparable aspects of how human behaviour is coordinated; action is constructed dynamically through interaction with the environment and society [246]. In a sense, situated action views the entire notion of action as illusory; the actor and the environment are simply reacting to each other continuously and simultaneously. Within situated action, forming plans is accepted as a possible part of, not a

precursor to, taking action [246, p. 50]. However, while plans may guide future action, they are descriptive rather than prescriptive: subsequent action must respond to the actual situation as it unfolds.

Situated action is not viewed as a *kind* of human action; rather, the term situated describes the nature of all human action. Indeed, Clancey [46] argues that situatedness is a consequence of human neuropsychology. As one example of this, he points to the dynamic nature of memory. A large body of experimental work suggests that memory is not fixed during an experience, but that it is reconstructed when needed, and that these reconstructions are distorted by the context surrounding both formation and reconstruction [225]. As Clancey notes, Gero [82] observes that recall is therefore an essentially interpretive act. Gero describes such interpretations as dynamic systems arising from the interaction between two parallel processes: one which "pushes" the reconstructed memory and one which "pulls" the result to bias it towards the current situation.

Proponents of situated action often point to Gibson's concept of *affordances* [83, 84][1] to distinguish it from Newell and Simon's physical symbol systems [273]. For Gibson, affordance is a relation between an animal and its environment describing those things that the environment *offers* the animal. For example, if a rock is shaped such that it is physically possible for a person to sit on it, then the rock *affords* sitting for that person. Affordances link the animal and the environment into an inseparable pair [84, p. 127]: perception and action form a dynamic system in which perception guides action and the result of that action is new perceptual information [151, 152].

Gibson argues that affordances are perceived directly, without internal mediation. This does not mean that the input from perceptual receptors is not processed [84, p. 251], but rather that percepts are formed spontaneously from environmental cues without unconscious inference (such as symbolic processing). In problem-solving terms, affordances fill the same role as analogical transfer, but they do not necessarily require that the actor have a prior experience to transfer.

Whether or not situated action follows from biology, it remains important as a description of actual problem-solving behaviour. While Suchman focused on what might be called "everyday problem solving", this is precisely why situated action describes expert work.

[1] Most HCI practitioners are introduced to the term affordance through Norman [201, p. 9], but Gibson's original conception of affordances differs from Norman's in subtle ways [201, p. 219]. McGrenere and Ho [179] give a detailed comparison of both interpretations.

Everyday problems are ill-structured by their nature because they take place in the larger world, where the number of possible operations is huge and the rules far removed from the actual problem. How many times has the idea of tightening a screw with a coin, butter knife, folded paper, or similar object been rediscovered? Situatedness allows more formal approaches to problem-solving while at the same time explaining how animals are able to quickly deal with the incomplete knowledge and combinatorial complexity of less constrained situations. Where recursive planning systems might analyze deep trees of moves, planning several steps ahead to predict the best course of action, animals that delay action in order to process information for a significant length of time would be less likely to survive. Since the routine problems encountered by animals in the environment are ill-structured, solutions do not need to be precise or optimal, and many different approaches are likely to lead to an acceptable result. Intricate planning is less important than continuous action because one may hit on a successful approach in any number of ways, but a lengthy investigation of the wrong approach involves a potentially deadly delay.

The combinatorial possibilities of action in the world are huge, but situated action, and particularly the notion of affordances, explains how actors avoid being overwhelmed by choices. Although a huge number of actions is possible, only the actions that are perceived to be afforded by the environment are relevant. The affordances perceived at any moment may not lead to a solution, but these change constantly. Behaviourally, this may appear very similar to the generate and test heuristic used by Newell and Simon. However, the selection of actions is neither random nor systematic, but is driven by the most salient perceived affordances at any given moment.

The situated action view is especially suited to ill-structured problem solving. It simultaneously deals with the additional complexity and uncertainty inherent in ill-structured problems, while also relying on the presence of multiple solutions when progress must be made quickly. Since ill-structured problem solving is also a core component of expert work, one would expect to see evidence of behaviours consistent with situated action in the problem-solving strategies used in expert work. As the next section demonstrates, the processes used by experts are consistent with situated action, perhaps most strikingly in the *dialectic process* observed by various researchers. (During this process, the expert poses a series of partial solutions, each of which becomes a resource in the new situation that their introduction creates.) While physical symbol systems and situated action may be at odds with each other as models of cognition, both are useful in guiding the design of expert support

systems: one as an aid to building computational models, and one for its description of human behaviour in practice.

## 2.5 Strategies Used by Professionals

While theoretical views provide an abstract account of problem-solving processes, this section considers how experts perform problem-solving in practice. Three aspects of professional practice are highlighted. The first two aspects are the processes of problem definition and the generation of solutions (essentially, Jones's divergence and transformation activities). The third aspect to be considered is the special role that pictorial representations play in many kinds of expert work.

### 2.5.1 Problem Definition

A dubious approach to problem solving is to start by gaining a complete understanding of the problem, then leveraging that knowledge to find a solution. Situated action argues that people tend not to behave this way in practice, but regardless of human behaviour a distinct problem-then-solution approach cannot be expected to work for ill-structured problems. Their ambiguity and interconnectedness makes it impractical to identify, let alone operationally define, every relevant variable.

Experimental results reflect this. In a study that compared the performance of first year and final year engineering students, Atman et al. found that students that spent a large proportion of their time on problem definition failed to produce high quality solutions [11]. These students tended to get trapped exploring the problem space and so failed to move on to generating solutions. A separate study indicates that students with an intermediate level of skill are most likely to fall into this trap [45]: less skilled students quickly run out of new criteria and move from a simple problem definition to a simple solution. The students who were the most successful were those that moved quickly to generating partial solutions.

By moving quickly to solution generation, problem-solvers do not have to identify every relevant aspect of the problem in order to find solutions. Rather, the understanding of the problem changes as the problem is investigated and new aspects come to light. Problem and solution are developed in tandem, by exploring the problem space and the resulting solution spaces together. Problem-solvers can learn to do this through education and experience [11, 45, 149, 154, 164]. This is good news not just for educators but also for designers

of support systems: the tools they build may play a positive role in shaping this education by emphasizing and supporting this approach.

Although experience is an important factor, there is also evidence that some people are predisposed to develop the problem and solution together rather than separately. In studies of children learning to program computers, Turkle [264, pp 106–115] observed that some children—mainly the males—prided themselves on mastering the details of the programming system. They were not satisfied until a program was both technically correct and elegant, suggesting an emphasis on searching for *the* optimal solution. A second group of children approached the computer as a negotiation partner. These children were not concerned that a program be technically correct in all cases, but that it work well enough to achieve the desired outcome. When errors were present that did not interfere with the overall goal, they tended to be allowed to the computer as personal idiosyncrasies, and even to be appreciated as such. When one student's program had to be told twice that the user wanted to quit before it would actually terminate, she left it in. Although she recognized that this was technically an error, she liked the fact that the computer exhibited quirky behaviour by "not [taking] no for an answer." Turkle emphasizes that the children in the second group worked with the computer through the prism of a personal relationship. What is also clear is that the children in this group had no qualms about changing the problem description to better suit a solution—even by redefining a typically hard constraint ("no bugs") as soft.

Experienced ill-structured problem solvers develop the problem and solution together in part because they use partial solutions to create anchor points from which to explore both the problem and the possible solutions [50, 164]. Cross and Dorst [50] use the nomenclature of problem spaces to describe this process as undertaken by industrial engineers:

1. The designers explore the problem space until they find or recognize a partial structure. Choosing this anchor also provides a partial structuring of the solution space.

2. Considering the implications of the partial structure in the solution space allows them to generate initial ideas about the form of the design, which extends the partial structure in the solution space.

3. This structure is transformed back into the problem space, where the implications are then used to extend the partial structure of the problem space.

4. The process is repeated until a matched pairing of problem and solution are found.

### 2.5.2 Solution Generation

Simply generating more solutions is no guarantee of finding a better result. If one starts from a bad idea, no number of incremental improvements is likely to beat a solution that was developed from a better starting point. Problem-solvers are more likely to find good starting points by exploring a breadth of approaches. Indeed, some people build celebrated careers on finding promising ideas in the unsampled regions of the space, championing those ideas when they are unpopular, and ultimately winning over detractors to establish the idea as a norm [242].

The possibility of creative breakthroughs notwithstanding, an overbroad exploration may be just as detrimental as one that is too narrow. Fricke [78] found that too much and too little breadth exploration were both suboptimal strategies in his studies of engineering students. While students that explored only one or two approaches were too focused on producing concrete results, those that expanded the solution space too broadly expended time and cognitive resources on organizing solutions rather than comparing and developing them. This suggests that support tools might be able to help experts effectively explore a broader range of alternatives by automating the management of generated alternatives.

Akin [3] found that architects learn to adopt a more balanced search strategy as they gain expertise: Novice architects tend to employ a depth-first search of the design space. They hit on an approach, and then follow through with it as far as they can. It is only when an approach reaches an impasse that they go back to look for alternatives. Over time, expert architects adopt a different strategy, neither breadth-first nor depth-first but rather a mixed strategy that considers alternative approaches and develops each in parallel, borrowing ideas from some to apply to others. So, while everyone could benefit from tools that *support* exploring a broader range of approaches, novices could benefit particularly from systems that *encourage* broader exploration.

Schön [228] describes the *process* of solution generation based on observations of expert practitioners from a range of disciplines. He identifies two key activities that are used together in a dialectic process. Both are responses to the fluid, experienced work that makes use of tacit knowledge—what Schön calls "knowledge-in-action"—being interrupted by *surprises*. When practitioners encounter a surprise, they enter one of two reflective processes: *reflection-in-action* or *reflection-on-action*.

During reflection-in-action, the practitioner is thinking about what is being done while

actually doing it, with the dual goals of better understanding the surprising phenomenon and influencing the outcome of the process. The practitioner's actions become a cycle of proposing and testing hypotheses about the nature of the phenomenon and how to change it in desired ways. Emerging states "talk back" to the practitioner, while the practitioner responds in turn by changing to a new state.

During reflection-on-action, practitioners stop and think back over what has been done and how it informs their own understanding of the problem and their practice. Reflection-on-action is a reflection on the already explored paths: unlike reflection-in-action, no new paths are explored. This leaves the practitioner free to explore other questions—for example, to consider their own behaviour and motivations, or to reconsider the nature of the problem being solved.

Schön sees the use of repertoires—collections of examples, actions, images and ideas—as essential to the reflection process. The practitioner's repertoire forms a basis of experience and knowledge through which new phenomena are understood [228, p. 138] and provides starting points to jumpstart the process of generating solutions when starting on a problem [36].

Nakakoji et al. present an approach to supporting reflective practice in linear domains such as writing, programming, and video editing [193]. Their *Amplifying Representational Talkback* (*ART*) prototype system assists writers by introducing a new interaction technique to traditional word processing. It allows breaking text up into rectangular chunks that can then be positioned in a two-dimensional space. (A one-dimensional view of the document is implied by the order of the chunks on the $y$-axis.) During the process of spatially positioning the chunks, the dynamically emerging representations of the text "talk back" to the author, engaging reflection-in-action [193]. When an arrangement is (at least momentarily) settled upon, the author can interpret the representation to obtain information about the state of the design and its rationale, thus enabling reflection-on-action [193, 279].

### 2.5.3 Use of Diagrams and Sketches

A common aid when solving problems is the use of external representations. External representations are a supplement to working memory [69, 146, 150, 218], but they have other advantages as well [31, 218, 283]. External representations allow the use of different knowledge and skills that are not available from internal representations (as when one writes down two potential spellings of a word to see which one "looks right"), and external representations

may use attention differently than internal ones [218].

The use of graphical representations is common in problem solving [146], but it is a particularly important practice in expert work [69,88,150,229,252]. The form of an external representation is significant, as it determines the information that is perceived and the mental processes that are activated by attending to the representation [283]. The pervasive use of graphical representations in expert work therefore suggests that these representations have properties that make them particularly well-suited to the needs of expert workers.

Larkin and Simon [146] compared the use of sentential representations and diagrammatic representations in well-structured problem-solving. They argued that diagrams are distinct from sentential representations because sentential representations are sequential while diagrammatic ones are indexed in two dimensions. By grouping related information, problem-solvers reduce search time, which leads to better performance overall. However, when Cheng [44] expanded one of Larkin and Simon's tasks to include a tabular representation, he found that the diagrammatic representation of the problem was up to six times easier to solve than the sentential representation, while the tabular representation (which also allows two-dimensional indexing) was not significantly better than the sentential one. Hence, locational indexing alone does not explain the value of diagrams in solving problems. Cheng suggests that the representational structure may cause the participants to use different problem-solving strategies.

While Larkin and Simon used well-structured problems in their work, other researchers have looked at the use of graphical representations in ill-structured problem solving. This work has focused on the use of sketches, a less rigorous representation form than the diagrams studied by Larkin and Simon. The imprecision found in sketches has long been recognized as beneficial to the early stages of design and engineering, where sketching is a common practice: Leonardo da Vinci advocated the use of intentional ambiguity and untidiness when developing compositions in order to stimulate visual creativity [69]. The preliminary sketches da Vinci prepared for monuments he was commissioned to build (e.g., [156, 157]) illustrate this technique well (see Figure 2.3).

Suwa, Gero, and Purcell [252] found that sketches can provide visual cues to non-visual information. They also note that creating an arrangement of items—whether intentionally or as an accident of the sketching process—hypothesizes relationships between those items, and that this helps to generate or refine ideas.

Sketches also play an important role in the dialectic process [69, 88, 229, 252]. Schön

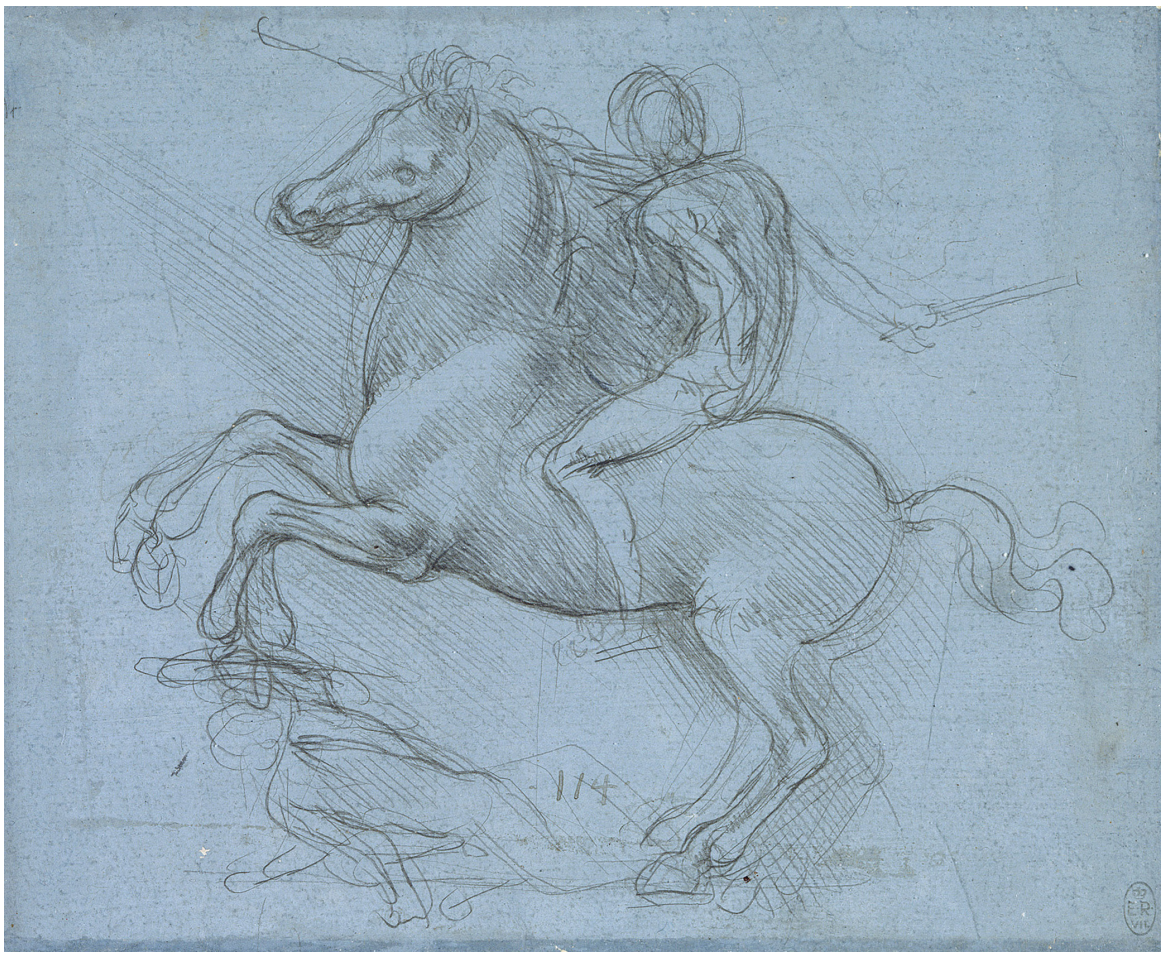


Figure 2.3: *A Study for an Equestrian Monument*, Leonardo da Vinci, c. 1485–90. Metalpoint on prepared paper (Royal Library, Windsor Castle, RL 12358r). This is one of many pages of sketches made by da Vinci while planning a monument to Francesco Sforza. This early version features a rearing pose supported by a cowering figure; by the final version, which was modelled in clay but not completed, da Vinci had switched to a simpler pose. Note the use of repeated lines, imprecision, and overlaid poses, all of which emphasize the tentativeness and ambiguity of the sketch. *Note:* Courtesy of The Royal Collection © 2010, Her Majesty Queen Elizabeth II. Reprinted with permission.

and Wiggins [229] describe sketches as the medium through which designers engage in a conversation with the materials through the cycle of sketching, reflecting on the sketches, and then using the result to drive the development of new sketches. Goldschmidt [88] argues that the ambiguity of sketches is a substantial component of creativity in the design process. In protocol studies, she observed that architects switch rapidly between two modes of reasoning while sketching. In the first mode, which she calls "seeing as", the architects used analogical reasoning to infer new meaning from the sketch. In the second mode, "seeing that", the architect recognizes the design consequences of the newly added meaning.

Fish and Scrivener [69] also discuss sketching as a cyclical process involving rapid mode shifts. Like Goldschmidt, they focus on the cognitive processes that are engaged by the sketching conversation. In sketches they see properties that mediate the translation between abstract information that is descriptive and propositional and more concrete information that is depictive and spatial. The sketcher begins with descriptive knowledge, creates a sketch, scans the sketch using attentional processes that extract new descriptive information, and use that to drive new depictions. They also view sketching as a tool for forseeing the results of an operation without actually performing it.

HCI researchers have developed some tools to support exploration through sketching. Silk [143] and Denim [163] both support the design of interactive systems through sketching screens and storyboards. Both systems are designed to increase the interactivity possible with hand-drawn interfaces, allowing sketches to serve as higher-fidelity prototypes than would normally be possible with traditional paper prototyping [178].

Working in the other direction, the *Napkin Look and Feel* [9] aims to *reduce* the apparent visual fidelity of interactive prototypes. Napkin Look and Feel is a pluggable extension for the *Java Swing* user interface toolkit that renders interface elements in a rough, sketch-like form. Lowering apparent visual fidelity can be useful when testing prototypes because prototypes with high visual fidelity tend to elicit comments about specific visual attributes rather than about fundamental usability issues [178].

Another research direction with potential is sketch-based image retrieval [76, 79, 119], which could be used to help generate additional interpretations of a sketch. An application of this work that may initially seem misguided is the *Magic Canvas* [232] system, which converts a sketch into a three-dimensional scene by matching parts of the sketch against a database of models and automatically determining an appropriate location and orientation. Automatically choosing a single model necessarily eliminates the ambiguity that is one of

the sketch's most beneficial features. However, this system focuses on domains such as animation where the rearrangement of a fixed set of objects is an important subproblem. The design of those objects is thus often a detail to be filled in during convergence rather than a problem feature whose final form is unknown.

## 2.6 Creativity Research

The need to generate and consider alternative approaches is vital to expert work processes. To effectively generate these alternatives requires *creativity*, the ability to produce ideas which are both novel and appropriate [243, p. 3]. A support system for expert work must therefore also be a support system for creativity. This section discusses creativity from two perspectives: studies of the creative process undertaken by psychologists, and studies of support tools taken from HCI research in the related field of creativity support.

Csíkszentmihályi [51, 52] introduces the term *flow* to describe the ideal mental state for creative work. The principal requirement for entering the state of flow is to balance a high level of challenge and a high level of skill. Other factors that contribute include a sense of control, an activity that the individual finds intrinsically rewarding, an environment conducive to intense concentration, and immediate feedback. When engaged in flow, an individual's cognitive resources are so completely devoted to the task at hand that their sense of time is altered and they lose awareness of their own consciousness. Thus, entering a state of flow also requires that an individual have facility with his or her tools. A consequence of this for experimental work is that users cannot be expected to enter flow when using prototype systems without being given extensive practice time.

A critical requirement for entering flow is finding the Goldilocks zone where skill and difficulty are both high and balanced. Because support tools can profoundly reduce the difficulty of subproblems that can be solved algorithmically, they can create situations where the users of the tool may be able to enter flow when it would otherwise be impossible given their skill level. Woodbury and Burrow [277] suggest one way that support tools might do this: by assuming responsibility for some or all of the convergence process and working out the mundane details of a solution automatically. This would allow experts to remain focused on the more engaging task of exploring alternatives, while at the same time alleviating novices of the need to be fully skilled in the domain. In either case, the net result is that it is more likely that the user will be able to enter flow.

De Bono [57] coined the term *lateral thinking* to describe a kind of problem-solving that emphasizes novel solutions. His writing presents a number of practical techniques designed to facilitate this mode of thought. The techniques are based on the following ideas:

- Acknowledging the views that dominate the perception of the problem, then searching for alternatives.
- Avoiding critical or dogmatic reactions when generating new ideas, for example, by interjecting the nonsense word *po* to suggest an alternative idea in a non-threatening way.
- The use of randomness to stimulate ideas, for example, by picking a random noun from a dictionary and associating it with the problem.

Many of de Bono's techniques are aimed at group settings. One way to apply these to support systems could be to give the system the role of group member, adapting the technique to suit the strengths of computing technology. For example, search tools might be used to identify both dominant and outlier views, as well as serving as a catalyst for generating new ideas based on ambiguous or random queries [141].

Although de Bono's ideas about enhancing creativity have gained broad acceptance culturally, it would be irresponsible to fail to point out that they have not been subjected to rigorous scientific testing. Sternberg and Lubart go so far as to dismiss de Bono's work as "equally damaging to the scientific study of creativity [as mystical beliefs]" [243, p. 5]. Whether or not such a strong reaction is justified, it is clear that overreliance on de Bono's work entails some risk.

Drawing on results from creativity and design research, Candy and Edmonds [36] argue that creativity support systems should enable users to:

- View problems holistically—for example, by providing multiple representations.
- Support parallel channels of exploration by supporting rapid, fluent task switching.
- Modify constraints.

Resnick et al. [220] similarly review existing work on creativity support tools and present a unified list of design principles. Their work combines and extends principles previously proposed by a number of scholars [108, 191, 221, 231, 234, 278]. Their first principle calls

for supporting and encouraging exploration by, for example, allowing any attribute of the problem state to be changed, supporting the comparison of results, and providing history manipulation (backtracking/undo) capabilities. In subsequent work, Shneiderman [235] recommends that history manipulation should include the logging, editing, and replaying of histories with different parameters.

Several of Resnick et al.'s principles emphasize the importance of selecting fundamental building units at an appropriate level of abstraction and complexity. They argue that by favouring simple elements and features that can be combined in a number of ways over more complex limited-purpose components, support systems can support a wider range and diversity of outcomes. They also point out that the level at which elements become black boxes determines the ideas that the system can be used to explore.

Resnick et al. also argue that creativity support systems should embody *epistemological pluralism* [265]. This is an appeal to treat all of the cognitive styles along the spectrum from *soft* (holistic, concrete, nonhierarchical, negotiational approach, focused on relations) to *hard* (abstract, hierarchical, focused on attributes) [264, 265] as equally valid. The soft and hard styles are biased for both profession [149,220,265] and gender [264,265]. However, a mixture of both approaches can be found in any group. (The magnitude and universality of gender differences especially has been exaggerated by popular media [65, 118].)

Finally, Resnick et al. point out that creative projects often require coordinating more than one tool. For this reason, they argue that tools should support data interchange features such as importing and exporting data as extensible markup language (XML), or the provision of plug-in architectures or remote procedure call mechanisms that allow third-parties to extend the tool's feature set.

Boden [21] distinguishes two classes of creativity based on whether they are applied within or upon the problem space. The first class, exploratory creativity, is limited to those points that can be reached by exploring within a space, that is, all of the valid combinations that can be generated from the initial state. The second class, transformational creativity, occurs when the creative act is performed upon the space itself, thereby creating a new space of possibilities.

Modeling transformational creativity has long proven elusive for artificial intelligence (AI) researchers [21]. This was seen earlier in the chapter during the discussion of the problem space hypothesis: reframing and considering the metaproblem are kinds of transformational creativity. For AI researchers, the dilemma is that a model is either limited to a

fixed set of configurable elements or else that it presupposes the existence of a creative agent that is able to invent new elements. Builders of support systems can assume the presence of the agent, but are left with the problem of representing that agent's transformations to the space. As the space representation is made more flexible, tools must become more abstract and general, while support strategies become less focused on the original domain.

Designers of support systems are probably better off using a fixed, widely-accepted problem space. Three arguments support this view: First, transformational creativity seems most likely when the practitioner is actively considering the metaspace, such as when reflecting on the shortcomings of a number of previously discovered solutions. Choosing a space is therefore best treated as a distinct but related problem. Second, working in a familiar space lets experts work more efficiently by enabling space-dependent techniques like *forward chaining* [145]. Because transformational creativity is rare, the productivity cost of losing access to familiar techniques is not justifiable. Finally, support for space transformations will either be limited, in which case it is unlikely to support the kinds of transformations that are useful in practice, or else it will be reduced to a programming environment that is decoupled from any particular problem domain.

## 2.7 Summary

Solving ill-structured problems is an integral component of expert work. Though many everyday problems are also ill-structured, the problems in expert work are generally harder, more complex, and will have broader and more lasting consequences. Solving these problems presents a particular kind of challenge as one must not only discover an acceptable solution, but also find ways to frame the question so that a solution can be discovered. The problem solver must negotiate—even embrace—the immense possibilities presented by the problem's ambiguity and from this extract a final, precise form. This requires extraordinary flexibility: with the assumed constraints, with the problem representation, and with the solver's own processes and approaches. Extending the support offered by typical applications to include these processes will demand just as much flexibility and creativity, but it will also require a common language and starting point; a framework for comparing approaches and evaluating outcomes. The next chapter provides this starting point in the form of design principles that synthesize and extend the background presented here into a more practical format for interface design work.

# Chapter 3

# Transformation: Deriving the Principles

## 3.1 E Pluribus Decem

The previous chapter demonstrated that many research perspectives are relevant to an understanding of expert work. While most HCI work on supporting experts has focused on only one such perspective at a time, there are many benefits of considering several perspectives together. One benefit is that independently discovered results that are common between perspectives have in effect been tested more rigorously. Noting observations that recur across many domains also suggests which kinds of support will be the most effective and will apply the most broadly. Noting differences between analogous results helps identify which aspects of successful expert practice are necessary, and which are coincidental. A broader understanding of a phenomenon may demonstrate multiple ways that a particular need can be supported. This in turn can lead to more elegant HCI designs since it may be possible to reduce the number of features required without impairing usability. Finally, by noting gaps in knowledge, discontinuities, or areas that seem to defy obvious transfer to software form, it is possible to identify open problems in need of further study.

## 3.2 The Principles

To reap the above benefits, and to convert the results of the previous chapter into a form that can be applied more readily to support tool design, I have derived the following ten design

principles from the broad theoretical background found in Chapter 2. The principles express the common requirements and processes of expert workers across a variety of domains and theoretical perspectives, and focus particularly on supporting the activities of divergence and transformation, and on helping the expert to generate, manage, and evaluate alternatives—the primary distinction between the task requirements of divergence and transformation and the contemporary emphasis on refining a single final product.

### 3.2.1 Make Partial Solutions First-class Entities

The early stages of ill-structured problem-solving consist of generating and comparing a variety of candidate solutions. These are not fully detailed, but incomplete, prototypical, abstract, and illustrative. Only later will one of these partial solutions be fully reified into a completed form. Yet typical applications present the user with a workspace that is limited to representing a single solution at a time. The tools available are geared towards entering the details of an already chosen solution, not towards brushing a number of solutions in broad strokes. While this is appropriate during convergence, when the user must finally commit to all of the details, it is at best awkward during the early stages of problem solving. Users of such applications end up repeatedly redoing work at a fine level of detail as new approaches occur to them, or else they may use ad hoc methods such as creating a separate file for each partial solution using the Save As command.

Support systems should treat solutions as first-class entities that can be composed, arranged, and transformed with an ease at least comparable to the manipulation of sentences in a word processor. This means making alternatives accessible, either explicitly (e.g., side-by-side windows) or implicitly (e.g., with parametric models). Support systems must represent a mixture of precision and ambiguity, completeness and incompleteness, placeholders and final forms, all within the same solution set. Internal representations and algorithms must handle missing or approximate data gracefully, without distracting error messages but also without misleading the user as to the solution's completeness.

### 3.2.2 Support Problem and Solution Matching

Finding a solution to an ill-structured problem requires that one also find the right problem. Experienced ill-structured problem solvers explore the problem and solution spaces simultaneously until they produce a matched pairing. In order to find this match, problem solvers

must be able to evaluate how well a given solution fits a given problem specification. An important aspect of the problem representation is the set of hard and soft constraints. The manipulation of perceived constraints helps establish the criteria used to evaluate candidate solutions and drive the exploration of the space [36].

Support systems should explicitly represent important aspects of the problem (including the set of constraints) and allow them to be manipulated. They should also assist problem solvers with the task of evaluating solutions against problems, for example, by computing and presenting metrics that describe the quality of the fit.

### 3.2.3 Allow Subjectivity and Ambiguity

Ill-structured problems are characterized by soft constraints that lack objective measures. Moreover, experienced ill-structured problem solvers often choose to treat hard constraints as if they were soft. Support systems should therefore allow fluid, subjective metrics and be flexible in how they present and evaluate constraints. One way to do this may be to support soft representations, such as choosing visualizations over (or in addition to) raw data. Visualizations help people to see unexpected relations, but they may also have an additional benefit here. People tend to work in the medium and mode that they are given. If they are presented with hard numbers, then they will probably make their judgements and set their limits and expectations in terms of hard numbers. Likewise, if they are presented with pictures, they will likely make their judgements and set their limits and expectations in terms of what "looks good".

Precision and concreteness may be necessary to bring a solution to its final form, but these qualities also project a sense of finality that is inappropriate until that final form is ready for expression. Countering precision is ambiguity, and one of the major benefits of using sketches is their ability to capture and project this quality. Like soft constraints, ambiguity opens a door to reinterpretation and the opportunity to find more creative approaches. Support systems should capture, embrace, and even amplify ambiguity. Since computers are inherently precise in their internal representations, support systems must find ways to hide that precision through visualization, clever presentation forms, the use of imprecise input methods, fuzzy systems [138, 280, 282], or other techniques.

### 3.2.4 Prefer General, Flexible Actions and Representations

In any kind of design there is a trade-off between the specificity of the task the tool is designed for, how directly that task is supported, and how this impacts other uses. It is harder to write a screenplay with a word processor than with an application designed specifically for that task. A professional screenplay writer will invest in learning the more specific tool because they must perform this task often. The front-end effort needed to learn to use a dedicated tool effectively is worth the future benefits to productivity. On the other hand, a Grade 5 teacher appointed to write this year's school play will probably make do with the word processor. It is familiar, readily available, and flexible enough to get the job done. At the same time, the increased flexibility also allows the freedom to create a script with an unorthodox format—such as one better suited to the needs of nervous grade schoolers than production studios.

Expert support systems face a similar dilemma. The domains that experts work in are so different that each tool is likely to be highly specialized. At the same time, within a domain the system must allow the user the freedom to access a broad space of possibilities. The fundamental uniqueness of ill-structured problems demands the ability to produce unorthodox solutions. Operations and representations that are too specific will be of use in only a narrow range of cases, limiting both the problems that can be tackled and the solutions that can be found. A smaller set of simpler tools or operations that apply generally can enable more creative outcomes than a larger set of powerful but less flexible ones.

This does not mean that more specific functionality must be entirely eschewed, but it should be a secondary focus and it should answer a distinct need. The core operations and representations must be chosen to enable a wide range of outcomes. This done, there are likely to be common cases that cannot be conveniently expressed with simpler operations. These are likely to benefit from dedicated support.

Alternatively, and especially if there is little commonality across different practitioners, it may be more effective to provide a general and flexible (but perhaps less accessible) core for others to build more specialized layers upon. In particular, this approach seems the most plausible route to supporting transformational creativity.

### 3.2.5 Engage Multiple Ways of Doing and Thinking

There is no recipe that can be followed step-by-step to solve an ill-structured problem. Individuals, too, are predisposed to approach ill-structured problem-solving using different mixtures of cognitive styles. Support systems should not support one style of cognition or problem-solving to the exclusion of others. In particular, they should not assume that either a soft or hard cognitive style is required by a problem. In fields that have traditionally favoured one approach over another, further study may be required to better understand the techniques employed by experts who use other approaches. This may be especially true in fields that traditionally employ the hard style, as soft practitioners in these fields may be pressured to drop or conceal use of the soft style [265].

Instead of being limited to one mode or approach, support systems should be able to engage different methods at different times. For some individuals, visual methods will be preferable during divergence and transformation, while symbolic methods may become necessary during convergence; others may work best when engaging the somatosensory system, and so on. Aside from matching a user's personal style, using different styles of reasoning can also change a given solution's discoverability, and can radically change a problem's difficulty [130].

Special efforts should be made to incorporate some of the benefits of sketching, such as arrangement and juxtaposition, surveying the explored space, acting as a catalyst for reification, and engaging the perceptual system to assist in problem solving. Although the benefits of sketching are not completely understood, it is clear that sketches play an important and complex role in the work of many experts. Additional research will help to expand and refine this understanding. In particular, experimental support systems could help us to better understand which benefits of sketching are the most important, which aspects might be automated without losing their potency, and which sketching benefits might be transferred to less visual domains.

Another aspect of this principle is that difficult or unusual problems may call for the use of a combination of tools, or for custom tailoring a tool to better suit the needs of a particular problem or user. Systems should therefore support features such as data interchange between tools and end-user development. (I use the term *development* in part because the idea that programming is too hard for non-specialists has taken root since the late 1980s [221]. Consequently, some have proposed replacing programming with more accessible alternatives,

such as programming by example [162]. Contrary to this trend, however, there is ample evidence that non-specialist adults and even children are capable of creating short but effective programs [130, 206, 221, 264]. The principle limitations on end-user programming are the choice of programming language, the length of the program—about one page for children, five pages for adults—and the need for basic computer literacy [130].)

### 3.2.6 Support Forming and Testing Hypotheses

Although an ill-structured problem is never completely understood, the user's understanding of a problem changes as they explore it. This can be seen clearly in the dialectic process, which can be viewed as a cycle of posing and immediately testing hypotheses about the nature of the problem space. The hypotheses are necessary because the user is manipulating the state of the problem, but wants to create outcomes that arise from interactions between elements of the state or between elements of the state and external factors: "I need to draw attention to this part. Maybe if I make it darker? Mmmm... yeah, that works... but now it doesn't feel as cheery." These interactions are typically complex, hard to predict, and incompletely understood. Pulling at one corner of a solution to better satisfy one constraint may be found to affect other constraints in unanticipated ways.

It is necessary to shape both the solution state *and* the resultant implications in order to satisfy the problem constraints, yet only the problem state may be immediately accessible. Other implications may be important to the outcome, yet the user may be oblivious to their significance. Therefore, support tools should:

- Help the user to identify relevant implications.
- Allow the user to temporarily suspend the current activity to try operations that might manipulate the solution in desired ways.
- Help the user interpret the results of those trials.

In cases where an implication is sufficiently understood and formalizable it may be possible to invert causality, allowing the user to manipulate the implication and let the computer solve for one or more variables in the state.

### 3.2.7 Encourage Parallel Exploration of Breadth and Depth

There is a sweet spot between exploring too many alternatives and exploring too few. A common pitfall for novices is moving to concrete solutions too quickly. More experienced hands will explore a breadth and depth of solutions in parallel, switching between the two and moving between different approaches.

The ideal amount of exploration is unknown. It may vary by domain or from problem to problem. Furthermore, since the main problem that was identified with the exploration of too many solutions was that too much time was spent managing solutions instead of refining one of them, the right computer support could significantly change the optimum value. Keeping aware of the possibility of unexplored alternatives is also important in preventing entrenchment and the mindless reproduction of old solutions. Until better information is available, designers of support systems should err on the side of more exploration. Since novices tend to be the ones that explore too little, they will gain the most from such encouragement; more experienced problem solvers will be better judges of when to ignore the support system and stop exploring.

Support tools might provide feedback on the number and variety of alternatives that have been considered, or organize and display alternatives as a tree or other linked structure to give a visual indication of the relative breadth and depth of exploration.

### 3.2.8 Provide Rich History Mechanisms

Problem-solvers must review and reflect on the history of actions taken so far. When they reach a dead end, they must be able to back up to a previous state. Since they explore multiple lines in parallel, a more elaborate structure than the linear stack of traditional undo or a Web browser history is called for. Extending a historical move should not cause subsequent events in the history to be forgotten.

Rich histories help the user reapply ideas. If one approach peters out, concepts may still be borrowed from it and reapplied. Histories must not only be revisitable, but editable and repeatable [235]. It should be possible to "go back in time" and insert new operations without manually recreating subsequent events. Support tools should support rapid switching between multiple solutions. They should help the user to keep the thread of how solutions are related, and help them to distinguish their current location on this thread.

### 3.2.9 Assist the Construction and Use of Repertoires

Repertoires are collections of examples that help the problem-solver to understand new situations and provide the initial jumping-off points needed to bootstrap a solution for a new problem. Repertoires have traditionally been collections of physical artifacts, although this might not be a necessary condition. For example, design patterns [5,80] are abstractions of previous designs that play a comparable role to repertoires in some domains.

Specialized search tools may be a way to establish a new repertoire [141] or augment an existing one. Another way to build a repertoire would be to extend a tool's history mechanisms to span multiple sessions. To be most effective, tools for annotating, classifying, filtering, and browsing records should be provided so that examples are readily available when needed.

As with sketches, a better empirical understanding of the subconscious processes that are involved in the use of repertoires would be useful to predict how support systems could best augment or replace them. For example, it seems that one role of repertoires may be to serve as "starting sketches." That is, like sketches, it seems that concrete aspects of the repertoire artifact prompt the expert to identify new considerations and relationships in the current situation. If this is the primary role of repertoires, then searches would seem to be a good software analogue for them. On the other hand, perhaps a more important aspect of repertoires is to cue the expert to recall previously internalized knowledge associated with the repertoire artifacts. In this case a repertoire that is built on demand from search results is unlikely to work as well as one that has been personally collected by the expert, as there is no previous association between the artifacts and the expert's existing knowledge.

### 3.2.10 Create an Effective Environment

Cognitive psychology is rife with experiments showing that various environmental factors can impact cognition. As one example, a recent study found that the presence of the colour blue enhances performance on cognitive tasks that require creativity, while red enhances performance on detail-oriented tasks [180]. Situated action, too, predicts that environment plays a crucial role in determining the outcome of problem-solving. It shapes not only how a given solution develops, but even which solutions are discoverable. Observations of experts at work also make it clear that their environment is an important part of their process. They surround themselves with sketches and other artifacts that they have collected, and they

generate and array before them new sketches and artifacts as they work on new problems.

Although support tools are unlikely to have much control over the wider environment in which they are used, they can maximize the control that they do have. The difficulty of expert work suggests that a more rigorous, even aggressive approach may be warranted than is usual for an application. For example, a number of studies have verified that randomly-timed interruptions interfere with task performance [16, 53, 147, 205]. We also know that problem-solving is most effective when the expert can engage the state of *flow*, which requires an environment that is free from distraction and interruption. Expert support tools might temporarily suspend screen savers, audio cues for email, software update notices, and similar non-essential interruptions that the computer normally generates. A more advanced implementation would recognize that interruptions may vary in importance: critical events may require immediate interruption, unimportant ones may be suppressed, and those in between might be scheduled for moments when mental workload is reduced [184]. Ideal interruption times could be estimated using a cognitive model that predicts subtask completion [15,184], or by measuring pupil dilation (which can be done with eye-tracking hardware) [15].

## 3.3 Next Steps

These ten principles capture the common requirements and processes of expert workers across a variety of domains, and from many theoretical perspectives. It is therefore reasonable to expect that systems that successfully embody these principles will better fit the expert's workflow than contemporary applications that do not. Yet beyond verifying this assumption, several questions remain: Are the principles of practical utility? Do they help expert support system designers think about, communicate about, and implement new interaction techniques or support systems? If the resulting techniques or systems alter the performance of the experts who use them, in what ways and how much? In particular, does implementing a given principle produce a predicted effect? Are there unintended consequences? To what extent are the principles independent? Is it necessary to implement all of them, or will a modest subset suffice? Which of the principles, if any, are the most effective? The following chapters will begin to answer these questions, but only after a brief detour to address an important metaproblem—namely, *how* to go about answering the questions in the first place.

# Chapter 4

# The Knot of Evaluation

## 4.1 Introduction

Before deciding on the value of this—or any—set of design principles, it is first necessary to decide what makes a set of design principles "good", or at least promising. It is clear that the primary determinant of good design principles is that we can apply them to the design of something, and the resulting design will produce positive outcomes (or avoid negative ones). This requirement actually poses two related tests: first, that the principles can be meaningfully applied to a problem, and second, that the application has a measurable effect that is explained by (predictable from) the principles.

If these two tests are the only criteria, then there would seem to be little value in deriving principles. After all, directly applying the unfiltered results (from Chapter 2) should produce similar effects if the principles correctly interpret the source material. But beyond simple effectiveness, design principles should have three immediate benefits compared to the original sources. First, derived principles simplify the body of work by combining repeated results and concepts and eliminating extraneous ones. This reduces the time and effort required to apply the principles to a design problem. Second, derived principles reorient the results from statements about facts and relationships to statements about the constraints on design problems and how to manipulate them. This renders them in a form more readily applicable to design, making it easier to evaluate their applicability and to actually apply them. Finally, derived principles provide a common language that facilitates thinking and communicating about designs, just as it is easier to discuss a problem involving multiplication if one has internalized the concept, so that it no longer needs to be expressed as repeated addition.

These benefits are immediate because they follow almost automatically from the process of deriving the principles: these benefits are essentially a description of what a thoughtful derivation process would entail: the harmonization of similar concepts, wording the descriptions for an audience of designers, and so on. In addition to these immediate benefits, there are additional, less tangible qualities that good design principles should exhibit. First, good design principles should open the door to further research and innovation. For example, they may lead to the development of new interaction techniques that can be reapplied in many design contexts. Second, good design principles should help to identify open problems and gaps in current understanding. Third, good design principles should be a solid framework for understanding the domain in which they apply. That is, a designer should be able to discuss and critique projects by comparison to the principles, and to understand and integrate new but related concepts in terms of analogy, contrast, or subsumption.

The minimal test of acceptability, along with the additional criteria above, suggest a basic approach for evaluating the design principles. This can be sketched out as follows (a more detailed outline can be found in Section 4.4 starting on page 50). The primary goal of the evaluation is to establish that the principles are at least promising (worthy of further study). This is done by furnishing an *existence proof:* building an expert support system in line with the principles and showing that it produces some benefits. Building on this foundation, additional evidence can then make a case for the stronger claim that there is not just one such system, but that using the principles to design expert support systems can in general be expected to lead to good designs. Adding further support, evidence for the three secondary criteria listed above will also be provided. Some examples of the first of these criteria (that the principles should inspire further innovation) will become major topics. As for the second criterion, a number of open problems and questions were already identified when the principles were defined.

## 4.2 Evaluation in HCI

With the overall goals and approach now understood, the actual form of the evaluation process can be addressed. Unfortunately, the question of how to best conduct and evaluate HCI research generally, let alone in the area of expert support systems, is an unsettled one [39, 92, 275]. A source of contention is that HCI practitioners come from a diverse range of backgrounds (primarily computing science, design, ergonomics, and psychology), each of

which has different methods and criteria for the acceptance of ideas. Despite this multicultural background, results grounded in the methods of psychology (especially usability evaluations [92]) have generally dominated publication venues. One reason for this is that psychologists played key roles in HCI's founding. The field's seminal works [37, 202] are strongly rooted in basic cognitive psychology research, and psychologists have had a correspondingly large influence on research methods in HCI. Another reason may be a desire on the part of the HCI community to be taken seriously as a discipline, as quantitative results provide a veneer of scientific rigour. Psychology went through a similar (though more profound) period during the first half of the 20th century [182, pp. 5–49].

Whatever the reasons, HCI publications exhibit undue preference for objective empirical results, with quantitative results further preferred over qualitative [92, 275]. Quantitative results have their place, but there is a trade-off between the precision of a result and its scope [109]. The bias persists even when it means settling for scientifically weak research questions that are narrow in scope and make few testable predictions [92].

Like a jar of cookies, weak questions can offer comfort but little substance. By fitting their work to what is publishable, researchers miss opportunities to discover valuable design innovations, which often fall outside the scope of the methods preferred by reviewers [92]. In response, several researchers have called on the HCI community to follow the example of other design disciplines—ones with a better record of innovation than HCI [92]—and place at least equal value on qualitative evidence, such as arguments from design rationale, expert opinion, and reflection [92, 188, 249, 250, 275]. These voices argue that quantitative data and the scientific method are not the only, nor necessarily the most appropriate, bases for evaluation in HCI [275].

If HCI is to be more receptive to evidence from a broader range of sources, it must address the need to combine evidence from different sources and with different standards of acceptance. One approach is suggested by Miller and Miller [183], who have similarly argued for accepting a broader range of evidence in medicine. Their concern was the growing import of *evidence-based medicine* in medical practice, which places primary importance on epidemiological (i.e., statistical) evidence.[1] They contrast the use of evidence in medicine with its use in the legal system. The legal system begins by setting an appropriate *standard*

[1]It appears that arguments like Miller and Miller's have been successful, as medical practice now emphasizes *evidence-informed medicine*, which uses epidemiological evidence as a guide to decision-making rather than an arbiter.

*of proof*, such as "more likely than not" or "beyond a reasonable doubt". Lawyers then try to meet (or deny) this standard by organizing evidence into a *factual matrix* where pieces of information corroborate each other to present a set of consistent facts. Different kinds of evidence may be weighted differently, and the reliability of both the information and its source are considered.

In legal proceedings, the standard of proof is based on risk [183]. For example, criminal proceedings use a higher standard than civil ones because of the correspondingly harsher penalties that would be unjustly applied after an incorrect verdict. At the other end of the risk spectrum, the *precautionary principle* allows lawmakers to enact restrictive legislation based on possible causes of harm without conclusive evidence that the risk of harm is real [7, 183]. The standards also acknowledge that false negatives are less damaging than false positives [20, p. 352], a principle that also guides the design of experiments [187, p. 35].

Interpreting this in an HCI context suggests that broad-scoped research should, at least initially, have a lighter standard of proof due to the risk of culling a promising line of inquiry before it can properly mature [92]. As the research matures, a factual matrix will gradually develop as different elements are investigated using a variety of methods—and hence, a variety of scopes, degrees of precision, and strengths. While legal cases have a fixed end, research can continue indefinitely, so its evidence continues to change over time.

What properties lend strength to a factual matrix, making it more convincing than alternative explanations? Philosophers describe the process by which one explanatory hypothesis wins out over competitors as *inference to the best explanation* [102]. Informally, the best explanation is often characterized as the one that is the simplest according to criteria such as Ockham's razor. Thagard's *explanatory coherence theory* [256] expands on and formalizes this notion. Thagard has implemented explanatory coherence as a computational model which can successfully predict the acceptance of explanations in a number of domains, including science, law, and medicine [256, 257]. Thagard argues that, over time, explanatory coherence yields progressive approximations of truth [258].

According to Thagard, explanations are selected primarily by two criteria, *consilience* and *simplicity*, of which consilience is the more important [255]. Consilience captures how completely a theory describes the phenomena being explained. One theory is more consilient than another if it explains more *classes of facts*. Simply explaining more facts is not as powerful as explaining facts from different domains. For example, a theory of light that explains one case of reflection and one case of refraction would be more convincing than one

that explains refraction in two different media. What constitutes a class of facts cannot be formally defined, but it is generally agreed upon within a field [255].

Thagard's simplicity acts as a constraint on arbitrarily increasing consilience through the simple accumulation of ad hoc hypotheses [255].[2] An ad hoc hypothesis is one introduced specifically to explain a special case that does not fit the main hypotheses (i.e., those that explain many facts). Simpler explanations require fewer ad hoc hypotheses, but simplicity also has a temporal aspect. Any hypothesis might be seen as ad hoc when it is first introduced. Therefore, it is important to observe whether that hypothesis is used to explain a growing body of facts over time.

For research like that presented here—broad-scoped research for which generalizability is a key goal—two points stand out from Thagard's theory. The first point is that attempts to demonstrate generalization by evaluating different prototypes or examples will be much more convincing if the examples are chosen to maximize consilience. The second point is that, like the application of risk in legal proceedings, the temporal aspect of simplicity argues against rejecting ideas too early.

## 4.3 Approaches to Generalization in HCI

A central goal of the evaluation is to argue that the principles will generalize to the design of all, or at least a large class, of expert support systems. A common approach in computing science, where empirical inquiry often begins with a concrete implementation [40, 197, 224], would be to build a system using the principles, and then show that it yields benefits. By itself this provides only an existence proof, showing that it is possible to build at least one good system based on the principles. At best, this provides weak evidence for causality or generalizability.

Some HCI researchers have argued for the use of *claims analysis* [39] as a basis for creating reusable knowledge from individual implementations [249, 250]. A claim describes the trade-offs and HCI knowledge that led to a particular design aspect of an artifact. By classifying and generalizing the knowledge in a set of claims associated with an artifact, their scope for reuse is generalized beyond that artifact [249, 250]. This promotes an existence proof into arguments and falsifiable hypotheses about the conditions under which a claim

[2] Ockham's razor is different from this definition of simplicity since it also applies to the number of postulates, which Thagard does not consider relevant [255].

generalizes to other situations, and what effects it should produce. While such arguments are, in a sense, already embodied in the artifact [39], making them explicit makes them more accessible for replication studies or incorporation into larger arguments.

A more widely known way to produce generalized HCI knowledge is through cognitive modelling. Cognitive modelling uses psychological theory and quantitative results to build predictive models of human cognitive performance. These models are a good example of the scope-precision trade-off. They are narrow in scope [249] (e.g., predicting postprocedural errors [34], menu label comprehension [136]), but within that scope they can have excellent predictive power. For example, a detailed GOMS (Goals, Operators, Methods, Selection rules) [37, 127, 133] analysis correctly predicted that a system designed to save a telephone company millions of dollars a year by cutting a few seconds from operator assisted call times would instead actually increase times by one second [91]. Cognitive modelling is undoubtedly useful when a design problem intersects with the specific variables predicted by the model. However, a narrow scope implies a small impact when evaluating broadly-scoped research.

Generalizability can also be demonstrated by extending an existence proof into an inductive argument by building and testing multiple systems. Positive test results offer additional support for the embodied claims, while negative results not otherwise explainable weaken or falsify them [92, 224, 250]. This process is comparable to, although not as well defined as, the use of meta-analysis to strengthen claims based on statistical evidence. How much successful replication is needed to be convincing? There is no definitive answer. As with all inductive arguments (including those made using the scientific method), no number of positive results can prove that a claim is true [213]. On the other hand, as already noted, a single test case can be sufficient in the early stages of promising work of broad scope, although this should ideally be followed with additional tests as the research develops.

As a notable example, it is worth considering Norman's set of interaction design principles, based upon his seven stages of action [201]. In his book, Norman considers dozens of everyday examples that break his principles, and argues that they can be improved by redesigning them accordingly. Norman has an important advantage: he can build an argument from any artifact he comes across, because he considers how well its design serves its primary function, whatever that happens to be. When, as in this dissertation, there are few or no suitable extant examples to draw upon, it becomes necessary to build suitable prototypes. Unless substantial resources are available, this will severely limit the number of examples that can reasonably be considered. In this case, it is all the more important to

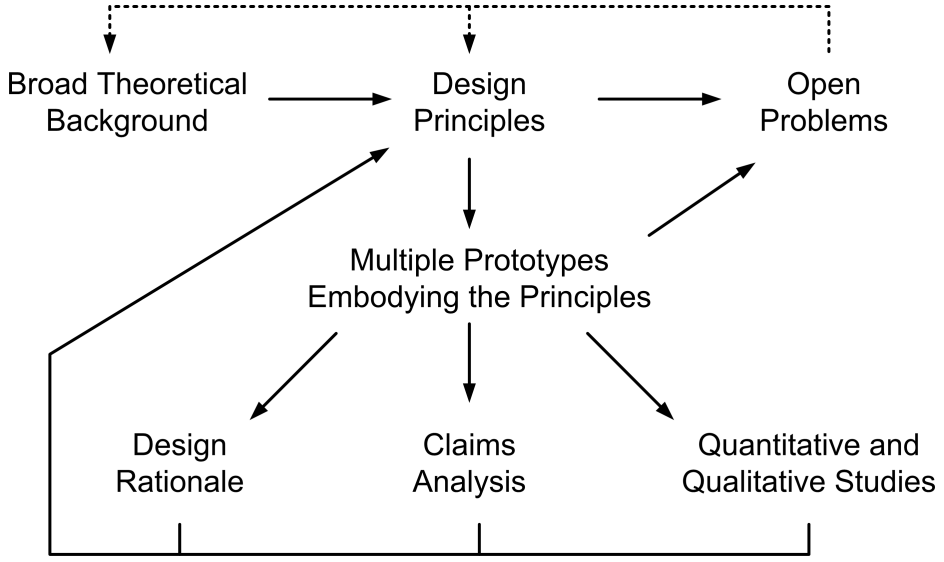



Figure 4.1: The major components of the factual matrix. The broad theoretical background informs the derivation of the design principles. These are embodied to varying degrees by multiple prototype systems. The prototypes are evaluated through a combination of design rationale, claims analysis, and quantitative and qualitative studies. This evaluation serves to reinforce confidence in the principles. Open problems may be identified anywhere, but this occurs primarily during the survey and derivation of principles. Investigating these open problems represents future work that will further expand the understanding of expert work (indicated with a dotted line).

choose prototypes that will increase the consilience of the argument.

## 4.4 Outline of the Factual Matrix

At the start of this chapter, three overall goals were presented for the evaluation of the design principles. Here they are presented as hypotheses:

1. There is at least one expert support system designed in accordance with the principles which has a positive impact on expert work.

2. In general, expert support systems designed in accordance with the principles will have positive impacts on expert work.

3. The design principles meet at least some of the additional criteria for "good" design principles listed on page 45.

Hypotheses (2) and (3) both depend upon the hypothesis (1), but are independent of each other. (Hypothesis (3) could in principle be satisfied even if hypothesis (1) was false, but such an achievement would have little value.)

Subsequent chapters will build an argument in favour of these hypotheses by marshalling the evidence into a factual matrix. The first element of this matrix is the broad theoretical and empirical background from which the principles are ultimately derived: the act of derivation implicitly appeals to their authority in support of hypotheses (1) and (2). The survey, along with the presentation of the principles, also provides instances of open problems and concept integration that are evidence for accepting hypothesis (3).

At the heart of the matrix are three prototype systems designed in accordance with the principles. I built and evaluated two of the prototypes; these are presented along with detailed design rationales. The third prototype was built and evaluated independently by a colleague and is presented in summary form in the context of other discussion.

The first system, *XDS* [124, 125], considers what is possible when designing a support system from scratch. *XDS* is a support tool for experimental design. It implements all of the principles to a degree, and provides strong support for many. Moreover, the principles are implemented mainly through a small set of tightly interwoven techniques, demonstrating that the principles can be implemented in an elegant way without sacrificing usability or aesthetics. *XDS* addresses not only the particular needs of the experimental design domain, but also the specific needs of the population that works in this domain. For example, most experiment designers have little formal training in ill-structured problem solving, so special attention is paid to coaxing these users out of less effective work habits. A study confirms that users of *XDS* try more alternatives, are able to improve on designs, and are encouraged to try a broader range of design choices.

*XDS* implements the principles as an elegant gestalt. It can serve as an inspiration to support system designers by providing a glimpse of what is possible with a diligent application. Although this generally supports hypothesis (3), it makes a thorough, controlled evaluation untenable due to the psychological overdetermination and interdependence of design elements. Instead, claims analysis is used to generalize specific XDS features, with a focusing on *halo menus* and *consequence displays*. The use of claims analysis strengthens evidence for the generality of these techniques, which supports hypothesis (3) while also

providing additional corroboration of the principles that these techniques embody.

In the case of halo menus, the claims analysis is done formally by presenting arguments about the technique's generalizabilty, its limitations and trade-offs, and a discussion of possible design variations. In the case of consequence displays, the claims analysis is informal and concrete: this technique provides the primary means of novel expert support in the second support tool, *Strange Eons*.

*Strange Eons* [121] supports content creation for board, card, role-playing, and other games. The gap between the domains is broad enough that very different strategies must be employed to implement the principles. ANOVA experimental design has enough formal structure that the entire design space can be modelled by the computer. Games, on the other hand, form individual subspaces through conventions and constraints created by existing components and the game's rules. The choice of *Strange Eons* as a second case greatly enhances the consilience of the factual matrix.

*Strange Eons* provides generic support for the creation of components (cards, tokens, boards and board spaces, booklets, fold-up tuck boxes, and so on). A degree of expert support is included for all component types through this generic support structure. However, primary support comes as consequence displays that are tailored to each component type's design space. For this reason, the evaluation of *Strange Eons* focuses on a particular component type for a particular game.

Unlike *XDS*, *Strange Eons* provides direct support for only a handful of the principles. This serves two purposes. First, it shows how the principles might be used to incrementally improve existing applications without the kind of radical design change posed by *XDS*. Second, it narrows the scope of support, enabling a more controlled evaluation. Two studies suggest that the consequence display in *Strange Eons* allows users to produce better designs in less time than they otherwise would. This result is made more compelling by the fact that the consequence displays in *Strange Eons* feature distinctly weaker consequences than those in *XDS*.

Since *Strange Eons* implements only a subset of the principles, it supports the generality of only that subset. This is addressed by briefly considering a third system, *Treesta* [125,186], which implements a largely complementary subset of the principles in yet another domain, lending further support to hypothesis (2). When combined with the other cases, *Treesta* also demonstrates that the principles have at least a minimal degree of independence.

Overall, the factual matrix outlined above—including the design rationale and evaluation for two novel systems and a combination of quantitative and qualitative studies and claims analysis—provides a much firmer basis for drawing conclusions about the quality of the design principles than a simple existence proof. Moreover, the use of a factual matrix (and other unusual methods, such as claims analysis) represents a test case that will advance the issue of appropriate research methods for HCI, a discussion with implications beyond the principles themselves.

# Chapter 5

# *XDS:* A Broad Implementation of the Principles

## 5.1 Introduction

A first step towards validating the principles is to build and evaluate a prototype system that embodies them. If the system produces benefits that can be explained by the effect of the principles, then the prototype constitutes an existence proof that indicates that further evaluation is warranted. On the other hand, if the system does not produce benefits, and if this failure cannot be explained by other factors, then this argues strongly against the effectiveness of the principles [224].

This chapter introduces such a prototype, *XDS*, an expert support system for planning experiments that use ANOVA statistical models. Designing ANOVA experiments is a challenging problem that is grappled with by experts in many domains. *XDS* supports this work using a small set of powerful, integrated techniques that together provide very broad coverage of the principles, making it an excellent candidate to establish an existence proof. This chapter introduces the problem and presents the design of *XDS* along with that design's rationale, which not only incorporates the principles but also demonstrates the importance of applying them alongside traditional HCI design knowledge. While this chapter presents the design of *XDS* in general terms, the next chapter completes the existence proof by connecting the design back to the principles and presenting a qualitative evaluation.

## 5.2 The Design of *XDS*

The design of *XDS* was especially driven by three goals that respond to weaknesses identified in the the processes used by typical experiment designers: First, many experiment designers use a flawed design process that tends not to consider a breadth or depth of alternative solutions. The inadequate exploration is explained by inexperience: for most researchers, experiment design is not their primary field of practice.

Second, experiment designers tend to construct new designs by retrofitting familiar designs that have worked in the past, rather than seeking a new design that is better fitted to the specific research question. Experimental design courses often emphasize this approach.

The third weakness arises from how experts evaluate alternatives, especially when working with ill-structured problems. Problem solving is a process of choosing options, and these options often involve making trade-offs between competing interests. The expert may choose to relax one constraint on the solution, but only at the cost of tightening others. The dialectic process that Schön and others observed in various fields (see page 26) is one method for navigating this process. It can be generalized to a two-step procedure in which a new candidate design is first derived and then evaluated in terms of project goals. This process is usually done informally, and the steps involved are not explicit. This informality allows problem solving to be more fluid, but it may also carry costs. One cost is that experts may begin to work from habit, making small modifications to stock solutions. This limits the region of the solution space that can be explored; the result is a faster problem-solving process, but a mediocre solution only loosely matches the problem. On a wider scale, this may lead to stagnation of the field.

To address these weaknesses, the design of *XDS* was driven by three goals (these are precursors to the current design principles; the principles and *XDS* were codeveloped and both have undergone multiple iterations):

1. Keep the user aware of available options for transforming the design. This makes it easier to try more alternatives, and therefore increases the likelihood of more exploration.

2. Explicitly represent the structure of the connections between alternative designs. This helps the user manage the alternatives that they generate, and makes it easier to jump from one alternative to another, which encourages a mixed exploration strategy.

3. Provide fast, accurate feedback on the consequences of design choices. Helping the user to predict important consequences of design choices neutralizes the benefit of working from habit, giving the user confidence in the reliability of newly explored alternatives.

## 5.3 The Domain of Experimental Design

*XDS* models the problem space of classical experimental design for ANOVA (Analysis of Variance) designs [70, 167]. A review of basic experimental and statistical methods will put this in context.

Researchers are interested in understanding the relationships between the variables in a model. This can be done by comparing different groups where the values of one set of variables (called *independent*) is fixed for the members of each group but different between the groups, and then observing how the groups differ in terms of another set of variables (called *dependent*). For example, by giving two different groups different levels of a Vitamin A supplement and then testing and comparing the eyesight of these groups, a researcher may hope to find a positive correlation between Vitamin A intake and visual acuity.

In any comparison of this kind, there will be some noise in the data due to experimenter error, limitations of the measurement instruments, and variations within the groups due to factors not considered by the experiment. Therefore, there is always some uncertainty as to whether an apparent difference between the groups (or lack thereof) is an accurate description of the effects of the independent variables, or if the true effects of the variables are being masked by noise. Statistical analysis allows researchers to quantify this uncertainty so that it can be constrained to fall within acceptable limits. This in turn allows the researcher to be reasonably confident in the accuracy of her conclusions.

A commonly used statistical technique is the $t$-test [90]. This test is used when comparing the mean value of a dependent variable between two groups, given certain assumptions about the distributions and variances of the populations and the sampling method. The researcher decides upon the acceptable levels of uncertainty by choosing values for $\alpha$ (the probability that the conclusion is falsely positive) and $\beta$ (the probability that the conclusion is falsely negative). The researcher then performs the $t$-test on the collected data, and concludes either that the observed difference between the means of the dependent variable is *statistically significant*, or it is not. If it is statistically significant, then the researcher can be confident,

within the limits set by $\alpha$ and $\beta$, that the difference between the groups is genuinely due to the effect of the independent variable. If it is not statistically significant, then there is too much noise to conclude whether the difference is due to the effect of the independent variables or is a result of random chance.

The $t$-test is only applicable to experiments that compare two independent groups. This covers a large number of research scenarios, including the canonical experimental design that compares a *treatment* group with a *control* or *placebo* group, but there are also many important research questions that do not conform to this requirement. The introduction of ANOVA models lets researchers answer more of these questions by opening a space of experimental design possibilities. In its simplest form, called one-way ANOVA, ANOVA generalizes the $t$-test to more than two groups. Using more complex ANOVA models, it is possible to, for example, test more than one independent variable, or to construct designs that use repeated measures (in which the different treatment levels are applied to the same subjects).

The full array of ANOVA models greatly increases the flexibility with which researchers can pose a research question while also turning a fairly straightforward process for planning experiments into an ill-structured problem that must balance the the research question, resource costs, the generality of any conclusions, and other factors. While the presence of an ill-structured problem hints that ANOVA experimental design may be a candidate for expert support, five features particularly recommend it:

First, this domain has a substantial impact on expert work. ANOVA designs are widely used in the basic, behavioural and clinical sciences, as well as in industry and government.

Second, many practitioners of ANOVA design are amateurs in the sense that their primary training and background is in their particular research domain, not in statistics. Such amateur practitioners seem particularly likely to benefit from a tool that supports what is likely the weaker of their two skill sets. Less experienced practitioners are prone to certain common mistakes, such as engaging in a depth-first exploration strategy rather than considering alternative approaches (see page 26). A good expert support tool may help to more quickly wean novices off of suboptimal strategies.

Third, ANOVA experimental design is a difficult problem, both subtle and risky. To design an experiment correctly the expert must bridge the gap between the research domain and the statistical domain by identifying the statistically relevant features of the research question and the context in which the research is performed, and expressing these in terms

of the statistical domain. Small changes to a design can require surprisingly large changes to the analysis method, while evaluating designs requires the expert to make assumptions about the outcome of the experiment *before* the data can be collected. The wrong design may waste time and money conducting an experiment that does not accurately address the research question, and may result in false or misleading conclusions. Such mistakes endanger the reputation of the practitioner and may incur health, safety, or legal repercussions.

Fourth, the design consequences that must be balanced by the designer are generally difficult to compute by hand and hard to informally estimate. However, many can be effectively computed or estimated by machine.

Fifth, many practitioners are not aware of the full flexibility of ANOVA designs. The advent of a general model of ANOVA designs that allows it to be treated as a space of designs is relatively recent [167]. As a result, many researchers were taught to perform experimental design in terms of a small number of islands within the space that have come to be preferred largely through historical precedent and their suitability for manual calculation. Helping researchers to discover a broader design space could have a tremendous positive impact on their design process.

## 5.4 Examining Typical Design Processes

An informal interview with a statistical consultant helped establish typical design practice in experimental design. His personal experiences were later confirmed during detailed process interviews that I conducted as part of a user study of *XDS* (see Chapter 6). Observations from both initial and subsequent interviews depict experimental design processes that fit the general patterns for expert work processes described in Chapters 1 and 2.

Researchers tend not to consider many alternative designs for their experiments. Inexperienced designers reported considering very few designs: typically only one or perhaps two. However, even experienced designers considered at most a few more. The interviews revealed five principal reasons for this pattern:

First, many researchers neglect experimental design. They either leave it to the last minute, or even start collecting data without performing any design. The order of data collection is a critical component of an experiment's design, so collecting data effectively sets the design; little change is possible afterwards.

Second, as discussed previously, many researchers are trained to design experiments by choosing the closest match from a set list. Once this is done, all that remains is a final convergence of the design.

Third, even experienced designers prefer to use rules of thumb to guide design decisions, and to avoid comparing multiple designs. Substantial work is needed to accurately estimate the effects of design decisions, and few tools are available to help.

Fourth, in some domains, such as the physical sciences, the research context places so many constraints on the designer that only a few choices are available. Experts in these domains may benefit less from being able to explore a range of designs since the space in question has few dimensions. On the other hand, having that fine grained control over the restricted space available to them might better help them to make optimal use of the remaining options. In either case, like other researchers, they might benefit from more accurate estimates of the consequences of their design decisions. For example, cost estimates may help them to allocate a sufficient budget for the experiment.

Finally, although researchers usually have some training in the elements of experimental design, few are trained in design *process*. Instead, designs are developed largely using trial and error. Even experienced designers use a hodge-podge of techniques. A typical approach starts with a design that worked in the past and that has similar features, adjusts it based on rules of thumb, and then—sometimes—ends with an estimate of the statistical power. (Statistical power refers to $1-\beta$, the probability that a conclusion is *not* falsely negative. It is related to the sensitivity of the design, that is, the minimum magnitude of the difference between groups that a design can reliably detect.)

Although all of the experiment designers interviewed reported that they tend to consider a small number of total designs, more experienced designers considered more designs overall, as well as a broader variety of designs. This matches the patterns observed for novice and experienced designers in Chapter 2, and agrees with the prediction that many researchers are not expert experiment designers.

Overall, the interviews revealed four key areas where novices could most benefit from support. First, novice designers showed a lack of awareness of the full set of design options. Being more aware of these options might encourage them to try a broader variety of approaches. Second, like novices in other ill-structured domains, novice experiment designers tend to execute depth-first explorations of the problem space rather than consider a variety of approaches. This can waste time working on a dead end and lead to a lower

quality design. Third, the information available to designers when comparing designs or considering an alternative approach is limited because they rely upon informal estimates of the effects of design choices. More accurate estimates would allow designers to make more informed decisions, and to better determine whether developing additional designs is likely to be worthwhile. Fourth, the lack of a design process means designers may have no tools to fall back on when encountering novel or difficult situations outside of their experience, which may lead them to ignore potentially serious problems with a design. Providing a more structured approach would help them to consider the possibilities more thoroughly and to experience less stress in novel situations. Among other benefits, this could increase the likelihood of engaging flow.

Although these areas for support are derived in large part from the behaviours reported by novice designers, experts also reported considering a relatively small number of designs. This may be due to the lack of a design process, or due to the fact that the coarse information they glean is not useful enough to warrant more detailed exploration. In either of these cases, the same supports that are indicated for novices above would also benefit experts.

The areas outlined above represent the strongest needs for experiment designers. Once they are satisfied, others are likely to be revealed. For example, once the designers start exploring more of the design space, they will need to manage the history of that exploration. Thus, while the particular weaknesses discovered during the interviews should be vigorously supported, a good support system design must still consider all of the principles.

## 5.5 Trade-offs and Consequences

All ill-structured problems require striking a balance between opposing trade-offs. Understanding how a candidate solution balances the various trade-offs in a problem is critical to being able to meaningfully compare alternatives. Support system designers must consider carefully how the state of partial solutions will be represented. In a given representation, a trade-off may be explicitly represented in the representation's state, it may arise indirectly in the way aspects of the state interact with each other or the problem context, or it may be unrepresented. A trade-off that is explicitly represented can be directly manipulated by the user. If the trade-off is indirectly represented, the user can still estimate it by examining the externalized solution state provided by the support system. For unrepresented trade-offs, the burden for recognizing, evaluating, and comparing the trade-off falls on the user.

Because they arise from the problem state but are not directly described by it, I call indirect and unrepresented trade-offs *hidden consequences*. It is largely the need to manipulate hidden consequences (while being unable to do so directly) that gives rise to the hypothesis-testing aspect of the dialectic process: the problem-solver must manipulate the problem state in order to observe the effect on the hidden consequences.

While indirect trade-offs can in principle be evaluated from an examination of state information, it is often difficult or cumbersome to do so. This may discourage the problem-solver from considering hidden consequences with due care, or to consider fewer alternative solutions due to the work involved in properly evaluating them. A support system can help by computing the consequence estimates itself and displaying them to the user. Ideally, these should be available not only as raw values but in a graphical form that can access the benefits of pictorial representations. (Indeed, a well-designed visualization may help the user to evaluate even directly represented trade-offs more effectively. I therefore dispense with the *hidden* adjective from here on, although support system designers are likely to find that supporting hidden consequences will be both harder than, yet also more beneficial than, supporting visible ones.)

Although the consequences that are most relevant to a given situation will vary, it is often possible to identify consequences that are common across many problem instances. For example, authors are often limited to a certain page count. When preparing such documents for submission, the author must make decisions about what material to keep, which sections can be expanded upon, and so forth. Authors could benefit from a better understanding of how potential changes will impact document length, and document length is a consequence of the document state. It depends not just on the words used, but on their order (this affects line and page breaks), the location of chapters, sections, and paragraphs, the size and location of figures and tables, typographic style choices, and so on. A simple consequence display might show a bar that includes the current length, the page limit constraint, and an indicator of how the length would change after actions such as pasting the clipboard contents (see Figure 5.1). Other support features could greatly enhance the usefulness of such a display. For example, being able to to select multiple text blocks or (as in the *ART* system discussed on page 27) having a separate area that text blocks could be moved to nondestructively on a contingent basis.

Returning to the experimental design domain supported by *XDS*, there are six principal kinds of trade-offs:

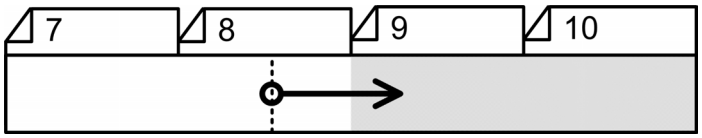



Figure 5.1: A hypothetical document length consequence display. The author is considering pasting text into the document, taking it from seven and a half pages to more than eight (eight is the page limit). If text was also selected within the document, additional indicators could show the effect of deleting the selection or replacing it with the clipboard content.

1. The risk of drawing an incorrect conclusion. This is represented in the statistical model by $\alpha$ and $\beta$.

2. The range of effect magnitudes which can be reliably detected (sensitivity). As the difference between two groups gets smaller, the designer must sacrifice other goals (such as low cost) to be able to find a statistically significant difference. Estimating the smallest effect magnitude that a design can reliably detect is called *power analysis* [47].

3. The generalizability of the results. The designer can make gains in other trade-offs by sacrificing how widely the results apply. Sometimes, the designer can argue separately that the results still generalize to a larger population using expert knowledge of the research domain.

4. The resource costs of running the experiment. This is a significant concern in most research domains. A typical application of power analysis is to determine the minimum sample size needed to find a difference of the expected magnitude. This ensures that resources are not wasted taking unneeded measurements.

5. The complexity of the experiment. Complex designs have implications beyond the cost of taking the measurements. For example, with a more complex structure, the probability of accidentally—or intentionally—violating the data gathering protocol increases.

6. How directly the phenomena tested by the design relate to the true phenomenon of interest.

In *XDS*, trade-off (1) is part of the design state, while trade-offs (2)–(6) are hidden consequences of the design. *XDS* provides visualizations for trade-offs (2)–(4) in each design's

summary representation (these are described in Section 5.9). Trade-off (5) is not presented directly, although it can be rapidly gauged by inspecting the design's detailed state representation. The interpretation of trade-off (6) is relative to the research question, so it cannot be computed automatically. However, designers can assess trade-off (6) by comparing the tests available from the design with their research question. In addition, a constraint system (also described in Section 5.9) allows designers to make statements about which tests are relevant to their research. Constraint satisfaction is also presented to the designer as a consequence display.

## 5.6 Design Space Explorers

*XDS* is in the family of applications known as *design space explorers* [277]. Design space explorers are derived from Newell and Simon's work on problem spaces (see p. 19) and Stiny's work on shape grammars [244]. They model the design process as the traversal of a space of designs, exploring possibilities by moving directly from one point (candidate design) to another. This point-to-point movement is accomplished by using design-level operators that directly transform one design into another design without moving through intermediate edit states.

From an HCI perspective, the design space explorer literature can be divided into two classes. The first class emphasizes the application of artificial intelligence and optimization methods to perform an autonomous exploration of the target space. Human participation in these systems is limited to supplying the input and applying the output of the exploration process. These systems are typically limited in scope, addressing one specific aspect of a larger problem and covering only a subset of the processes in Jones's taxonomy (see page 4). Indeed, they may fail to meet the above strict definition of a design space explorer, due to limitations in their model of the selected space. Example domains include machine code optimization [58], optimal system-on-a-chip configuration [105,160], adaptive hardware implementation [22], and aspects of aerospace engineering such as powerplant design [185]. Systems in this class are typically used in one of two ways. The first application is to automate part or all of the final convergence phase of a design: the system is presented with a high-level representation of the design which it completes by filling in implementation details. Systems of this kind are generally used to automate combinatorially vast but well-structured subproblems that are required to complete a design but tedious to execute

manually. These subproblems can be effectively tackled with standard algorithmic techniques. For example, given a description of the electronic systems in an aircraft design, the explorer might complete the design by finding a cost-effective but code-compliant way to wire the systems together. The second application of explorers in this class is to automate the transformation process by automatically exploring a subspace based on a fixed set of design variables. These systems use the model of the design space to simulate the effects of different combinations of the chosen variables. The results of this kind of exploration can be used, for example, to build a repertoire of useful starting points for more specific design problems, or to identify some of the broad alternatives in a problem and gain an understanding of the advantages and disadvantages. The statistical technique known as *response surface methodology* [24,25], which uses a series of experiments to build an approximate mathematical model of a black box system, could also be viewed as design space exploration in this vein.

The second class of design space explorer emphasizes a partnership between the user and the system. Systems in this class, including *XDS*, come from the tradition of Licklider [161], and of Fischer and Nakakoji [68]: they seek to empower users by leveraging the computer's complementary strengths rather than replacing users with autonomous actors. Examples in this class have previously focused on the design of objects in space, such as solid shapes [106] and room layouts [101]. This class of design space explorer is of particular interest as a basis for providing expert support, as the subjectivity found in ill-structured problems demands a high degree of human involvement. Of Jones's three processes, they provide the most direct support for transformation and convergence, but they can potentially support the user during all three processes. As well, their support for divergence can be bolstered with independent techniques, such as electronic repertoires or a representation of the problem space (in addition to the solution space).

Since previous design space explorers in this class have focused on the design of physical objects, their displays have featured the spatial arrangement of design elements. The abstract operators and constraints generating the design have typically remained implicit, without graphical representation. *XDS* extends previous design space explorer work by displaying the explored design space rather than the current design. It displays the track of all past designs, together with the design moves connecting them, and it arranges them according to their similarity to an initial design state. This approach explicitly presents the designer's task as one of trying alternate sequences of design moves, taken from an explicit,

finite set of such moves. The designs themselves are displayed, in full or summarized form, as points in the space. These summaries characterize important consequences of the design rather than providing a summary of the design state or, as in an approach used by Stump, Yukish, Simpson and Harris [245], abstract indicators of the design state.

Stump et. al. [245] proposed a *design by shopping* exploration model: the designer chooses designs that they like, and the system attempts to find optimal combinations of features. Optimization of this kind is in general not possible for experimental design. Changes to the design imply new assumptions about the research domain, and only an expert in that domain can determine which assumptions are valid for the research question. Similarly, other ill-structured problems are likely to have unique aspects that are important to a particular problem instance but which are not or cannot be represented in a computer model.

Previous design explorers emphasize reducing the relative cost of computing generative details, letting the designer focus on making broader changes [277]. How a circuit is laid out is unimportant so long as it is functionally correct and reasonably efficient. While *XDS* does compute generative details of this kind, the focus is on reducing the cost of another kind of design activity, namely that of balancing trade-offs through design evaluation and hypothesis testing in order to help the user explore more confidently and effectively.

## 5.7 Other Related Work

I have previously argued that the key drawback of most contemporary applications is a lack of support for divergence and transformation, two activities that revolve around the generation of alternatives. Other HCI researchers have considered the problem of supporting multiple alternatives and designer choice:

Terry, Mynatt, Nakakoji, and Yamamoto [254] present an alternative approach to supporting multiple simultaneous solutions to a design problem. Their system focuses on designs that cannot be well-described in terms of generative operators. Hence, they emphasize displaying multiple designs, which can be modified individually or as a group. In contrast, the design space in *XDS* has stronger structure, and I emphasize displaying the space of designs created so far.

Design rationale systems are related to design explorers in that both focus on the design space. These systems are used to capture the reasoning behind design decisions along with the design changes. Providing a design rationale establishes an argument for the design that

may be reviewed by critics or used as an aid in future design tasks [188]. In contrast, the purpose of design explorers is to support the current design task rather than future work. Although not a focus of this presentation, *XDS* allows the association of informal design rationale annotations with both individual designs and explored spaces.

Klemmer, Thomsen, Phelps-Goodman, Lee, and Landay [137] present the *Designers' Outpost*, a tool for designing Web sites. This system combines history capture with informal design rationale annotations. The design history is navigated by scrolling through a linearized tree of thumbnails of the design state. In contrast to the history state approach, the summaries in *XDS* feature consequences of the design state rather than the state itself. In *XDS*, there is no notion of an active or current moment in the design history—instead, it presents only the space of explored designs. This is supported by the design summaries, as they are often sufficient for decision-making, but the full design state is also accessed through this view when needed.

A common problem for systems that perform history capture is identifying which states are meaningful enough to be of historic importance [100]. This is less of an issue in design space explorers because they are defined in terms of higher-level operations (moves in the space) and so rely less on the aggregate effect of long sequences of operations. Consequently, a higher proportion of the products of operations are expected to be worth capturing. In practice, the designer will still wish to group a few moves into a single result at times. In *XDS*, this can be done easily as a side effect of the mechanism for inserting operations into the design history.

## 5.8 *XDS:* Representing the Design Space

Most applications present a single explicit representation that changes as it is edited. In *XDS*, the focus is shifted from a single incrementally developed product to a collection of alternatives related by the designer's choices. At all times, the interface displays the space of explored designs on a two-dimensional grid (see Figure 5.6). Initially, the explored space consists of only the Empty Design, which acts as an origin for the space and a starting point for the user. New designs are generated by performing moves on existing designs, and the resulting child designs are arranged automatically according to their similarity to their parent and the type of move performed. This arrangement explicitly represents the notion of design as exploring a design space. It also allows the designer to get a sense of

how designs relate to each other structurally by observing their spatial relationship.

In addition, parent designs are linked to their children by an arrow line, with children always placed close to parents. This creates a clear path for the designer to follow when reviewing the design history.

To apply design moves, *XDS* introduces an interaction technique called the *halo menu*. A halo menu encircles the design under the cursor with buttons that correspond to the available design moves. As the user moves the cursor around the space, designs will prompt the designer with design options, encouraging the designer to explore more alternatives. (Halo menus are a large topic; to avoid sidetracking this chapter, a thorough treatment will be postponed until Chapter 7.)

### 5.8.1 The Design Moves

The set of moves is symmetric: each move has an inverse, and the inverse move is always located opposite the move in the halo menu. In a few cases this distinction is somewhat artificial. For example, changing the $\alpha$ value is split into moves that increase and decrease the value. However, this organization maintains the metaphor of the interface as an explicit design space. The position of the halo menu button relative to the design determines where the new design will appear when the button's move is performed, so the distance between designs provides an approximate indicator of their similarity.

**Increase/Decrease Replications** Replications are reruns of the experiment; performing an experiment multiple times increases confidence in the results. Replication increases detectability, but also increases cost.

**Increase/Decrease $\alpha$** This allows the experimenter to declare the risk of a false positive that they are willing to accept. As this increases, the detectability also increases—the experimenter can show that a smaller difference is statistically significant, but this conclusion is more likely to be wrong.

**Increase/Decrease $\beta$** This allows the experimenter to declare the risk of a false negative. It is generally more acceptable to increase $\beta$ then to increase $\alpha$ because no conclusions are drawn when the experiment does not show a significant difference.

**Add/Remove Effect** Adds or removes an independent variable (main effect). Adding additional main effects may be necessary to accurately model the research question or

the limitations of the situation in which the experiment must be performed. However, it decreases detectability and makes the experiment more complex and expensive.

**Increase/Decrease Levels** Changes the number of levels of an effect. Suppose that the experimenter plans to test bread ovens at different temperatures in order to test if temperature affects the amount that the loaves rise. The number of levels of the temperature effect specifies the number of different temperatures that will be used in the experiment. Adding more levels adds additional cost, but it also increases detectability and, if the effect is fixed, increases the inference space (a measure of generalizability described in the next section). Control over the number of levels may be limited; a binary condition, for instance, can only have two levels.

**Restricted Randomization/Fully Randomized** Experimental results are most reliable when the measurements are taken in random order. Randomization minimizes the potential impact of any effects that are not included in the experiment design (either because the experimenter did not consider them, or because they are assumed not to significantly affect the dependent variable). Sometimes, complete randomization is either not possible or will make the experiment harder or more expensive to run. Building on the previous example, the experimenter knows that it takes time to change the temperature of the oven and that each such delay will cost the bread company money. The experimenter may therefore consider restricting the randomization of the temperature effect. This would mean that all of the loaves that will be tested at one temperature will be baked before switching to the next temperature. However, loaves will still be randomly assigned to a temperature, and the loaves within each temperature group will still be baked in random order. Restricting randomization decreases the inference space and implies assumptions about the research domain. When a restricted effect is significant, it is impossible to determine whether it is the effect itself or the error introduced by the restriction (or both) that is significant.

**Make Fixed/Make Random** Declaring an effect either fixed or random makes a statement about the generalizability of the results. When an effect is fixed, the results apply only to the specific levels that were tested; when random, the results generalize to all of the possible levels for that effect. If the bread company only ever bakes at the three temperatures they will test at, then the effect can be fixed. Making an effect fixed increases its detectability but decreases the inference space.

**Add/Remove Interaction Effect** How two or more effects interact may be significant. For example, there are mixed reports suggesting that sweeteners may have a synergistic effect, tasting sweeter when used together than the sum of their sweetness when used apart [267]. When an effect is added in *XDS*, all of the possible combinations of interaction effects with the existing effects are also added by default. This allows the most accurate possible interpretation of the results, but it also greatly decreases detectability if more than a few effects are present. Removing interaction effects will increase detectability, but it implies assumptions about the research domain.

**Nest/Unnest Effect** When one effect is nested inside another, it indicates that the levels of that effect will be different for each level of the effect that it is nested within. Effects can be nested in more than one effect, and can be nested more than level (effect A can be nested in effect B, which is nested in effect C). Nesting is needed to model certain properties of the research domain. For example, in a study about two different teaching methods, nesting participants within the methods (using a different set of participants for each method) would avoid the problem of learning effects. Nesting increases the cost of an experiment since it effectively multiplies the number of levels of the nested effect.

Not every design that can be generated as a sequence of the above moves is valid: some combinations of moves would visit points that are outside of the design space. (A valid design in this sense is one that can be expressed as an ANOVA model; it may or may not represent an appropriate solution to the problem at hand.) To be valid, a design must satisfy the following conditions:

1. A design may not contain interaction effects involving a main effect not present in the design. (This may happen as a result of removing a main effect.)

2. No interaction effect may involve both a nested effect and a main effect that it is directly or indirectly nested within. Since different levels of the nested effect will be paired with the levels of the effect it is nested in, there is no basis for drawing conclusions about their interaction.

3. The nesting of effects must be acyclic; this avoids the nonsensical possibility of an effect being nested (directly or indirectly) in itself.

When a move is performed that would lead to a design that breaks one of these rules, the result can be coerced into a valid design automatically. Essentially, the new design is snapped to the nearest valid design in the space. In the case of rules (1) and (2), this means removing the problematic interaction effects. In the case of rule (3), this amounts to performing a no-op. This coercion is always performed for rule (1). For rules (2) and (3), the interface will prevent the designer from breaking the rule directly, and will coerce designs as needed when a move is being applied deeply (see below).

### 5.8.2 Three Ways to Apply Moves

Once a move is selected, the designer can apply it by selecting the appropriate halo menu button. By using modifier keys, moves can be applied in one of three different ways, called *shallow*, *deep*, and *branching*. Figure 5.2 illustrates the different move types at a conceptual level by representing designs as compositions of design moves (indicated as variables).

Shallow moves are the most common. A shallow move creates a new design by cloning the target design and then changing it as needed to fulfil the operation. The result appears as a new point in the explored space, connected to its parent (the target design) by an arrow line. Shallow moves allow the rapid generation of alternatives while keeping related designs ready-to-hand for comparison, reflection, and further development.

Deep moves do not create a new point in the space, but instead modify a branch of the design history. Deep moves take place in two phases. First, as with a shallow move, a new design is created by cloning the target design and applying the selected operation. However, the result *replaces* the target design instead of becoming its child. Second, the changes to the target design are recursively propagated to its children using prototype-based inheritance [266] (the children inherit the new features that they do not themselves override). When a deep (or branch) move is applied to a target, the arrow leading to the target from its parent is changed from solid to dotted. This signifies visually that the design is the product of multiple moves, and only the combined result is shown.

Deep moves serve two principal purposes. The first role is non-destructive error correction. With traditional undo, when a user undoes some actions in order to insert a missing operation, all of the subsequent operations are lost. Deep moves let the designer modify the design history non-destructively to insert missing or corrected operations. The target design, and every design directly or indirectly derived from it, incorporates the correction as if the mistake had never been made. This is illustrated in diagram (d) of Figure 5.2: After

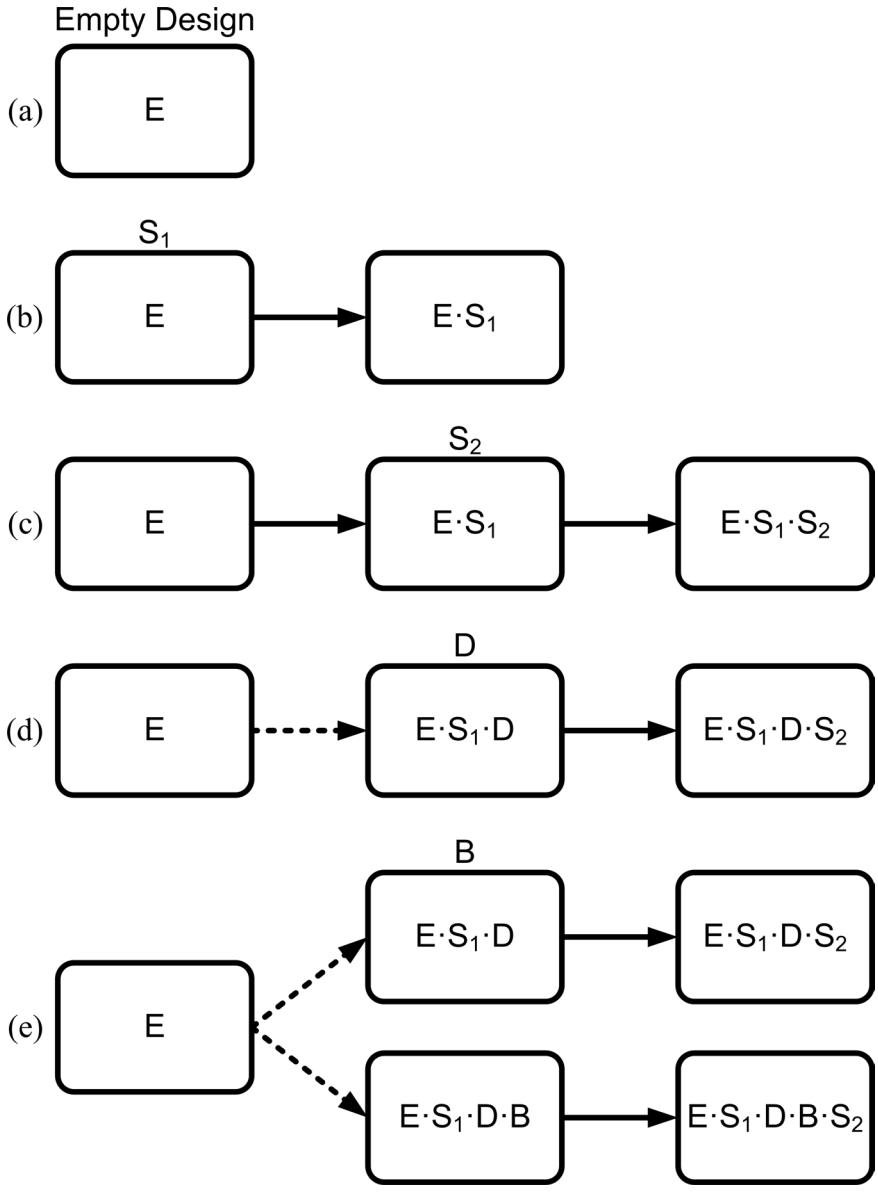



Figure 5.2: The application of design moves in *XDS*: (a) exploration begins with an empty design; (b, c) applying shallow moves ($S_1$, $S_2$) creates new children; (d) a deep move ($D$) is inserted into the history; (e) a branching move ($B$) is applied deeply to a clone of the target branch.

creating the state shown in diagram (c) using two shallow moves, the designer realizes that he or she forgot to apply a move $D$ after move $S_1$ and before move $S_2$. Performing $D$ as a deep move on the design $E \cdot S_1$ rectifies this, inserting $D$ into the design history as if it had actually been performed at the correct moment.

The second role of deep moves is allowing the designer to perform moves that do not generate visible points in the space, combining two or more moves as a single result. These "combining" deep moves are commonly used at the start of a session to set up a suitable base design, but they are uncommon once exploration begins in earnest. There are two reasons for this drop in frequency. First, shallow and deep moves are performed with equal ease; the choice of which to use is based on appropriateness rather than convenience. Second, the design space model ensures that moves generate complete, valid new designs. This means that moves are more likely to be worth recording (i.e., performing shallowly) than in a traditional tool, since they can be considered as potential solutions. (Contrast this with a word processor, in which the design moves in and out of syntactic correctness as the user types: the invalid states are unlikely to be of future interest.)

Branching moves enable the simultaneous generation of multiple designs. The target design is cloned along with all of its children in the design history, and the cloned branch is added as a new sibling of the target design. Then, the selected move is applied to the new branch as if it were a deep move. This allows the designer to quickly answer what-if questions that compare entire lines of exploration in a single step, without manually replaying moves.

## 5.9 Experiment Design Representations

The design explorer's interface is a zooming user interface (ZUI) [208] that uses two principal representations for designs at different zoom levels. When zoomed in, designs are presented in a table form that describes the current design state. When zoomed out, designs are presented as summaries of their consequences.

Figure 5.3 shows the detailed design state representation (zoomed-in view) of a simple design. In this representation, the design is split into a matrix of rectangular cells. The first cell contains a design summary, and the remaining $O(2^n)$ cells[1] (where $n$ is the number of main effects in the design) describe one main or interaction effect each.

[1]The set of effects that *might* be in a design is described by the power set of the set of main effects: the empty set represents the error "effect"; other non-singletons represent interaction effects.

Figure 5.4 shows a design summary representation (zoomed-out view). This representation consists of 7 elements, of which 3 are navigational or provide state information about the design and 4 represent design consequences. These are arranged within a square to maximize the number of designs that can fit on the display at any one time

Each design is assigned a serial number, labelled (b) in Figure 5.4. Serial numbers are assigned on a design's creation and do not change. They provide a fixed visual anchor to aid in navigation and visual search, as well as capturing the chronological ordering of explored designs. Designs also have a text label, labelled (g) in Figure 5.4. Initial labels are automatically assigned to designs as they are created. These describe the move(s) that were performed on the parent to generate the current design. The user can also edit labels to provide more descriptive names when desired. If left unedited, the default label will update itself to reflect the application of deep and branch moves.

When a short name is not enough to capture the information relevant to a design, it can be annotated with arbitrary text. The intended purpose is to record design rationale [188], but any comments deemed useful can be entered. Annotations are added through the same window that is used to edit the label. When an annotation is present, the label area will show a dogeared corner—see label (h) of Figure 5.4. To review an annotation, the designer can zoom in to the detail view or click the label to open the editing window.

The remaining elements represent design consequences: cost, detectability, the inference space, and constraint satisfaction. Each of these is described below:

For most research domains, the cost to run an experiment is important. Even when there are ample resources available, opportunity costs must be considered. The cost of running the experiment is estimated with a probabilistic model. To use the model, the designer must estimate two kinds of costs for each main effect: a fixed cost that must be paid for each experimental unit, and a change cost that is paid when one unit must be switched for another. For example, paying each participant $10 would be a fixed cost for the participant effect, while providing a new set of headphones every time participants change would be a change cost. This change cost can be minimized by blocking participants (running all trials with one participant before switching to the next participant). Displaying change costs shows one of the consequences of such randomization restrictions; there are other, statistical implications, which will be shown in the detectability values. If the designer does not bother specifying costs, default values ensure that the cost display is still a useful ordinal indicator of the required resources, though the absolute differences will be meaningless.

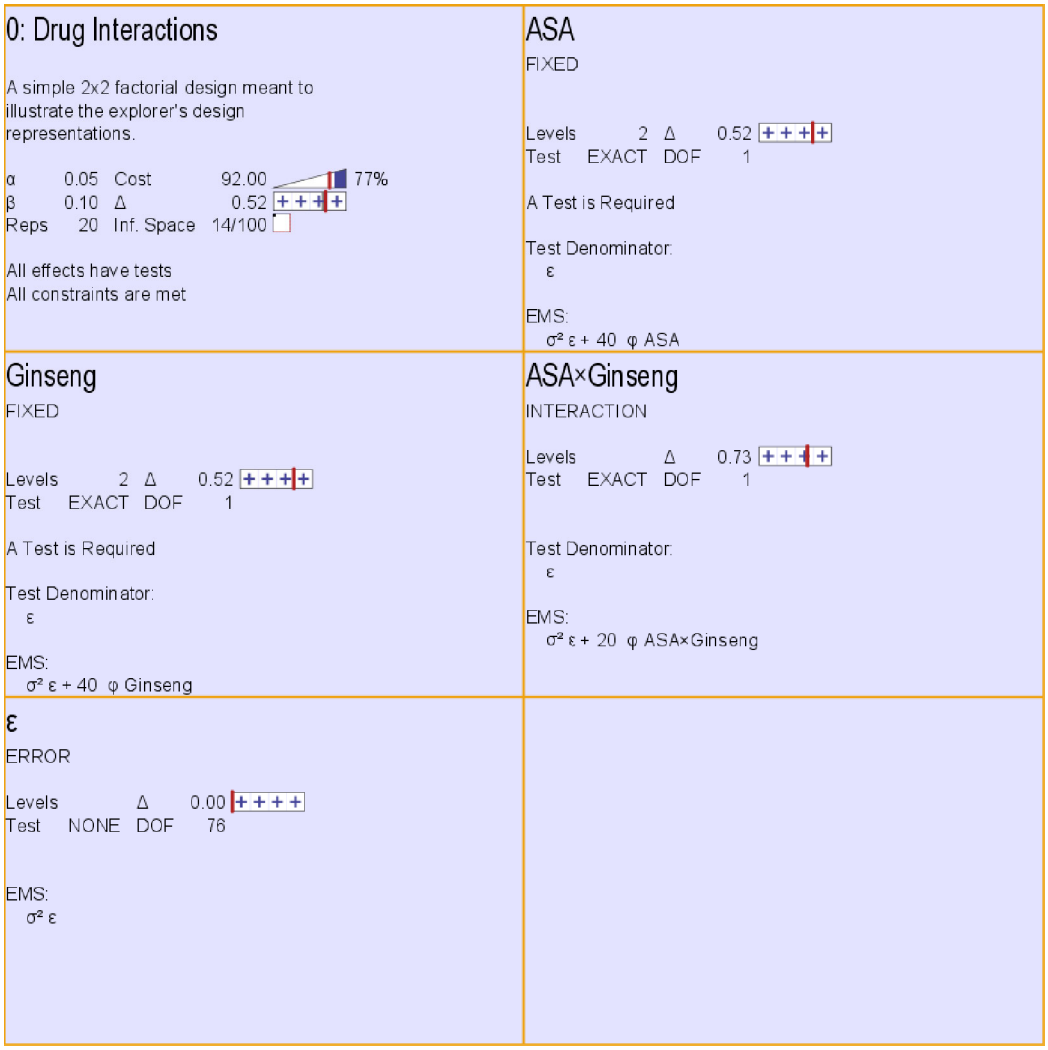



Figure 5.3: The detail view of an experiment design. The upper-left cell provides a more detailed form of the information available from the design summary, and also displays any design rationale annotations. The remaining cells each describe an effect in the design. While the design summary provides a constant complexity view of the design that is sufficient for most decision-making, the complexity of the detail view varies with the complexity of the design.

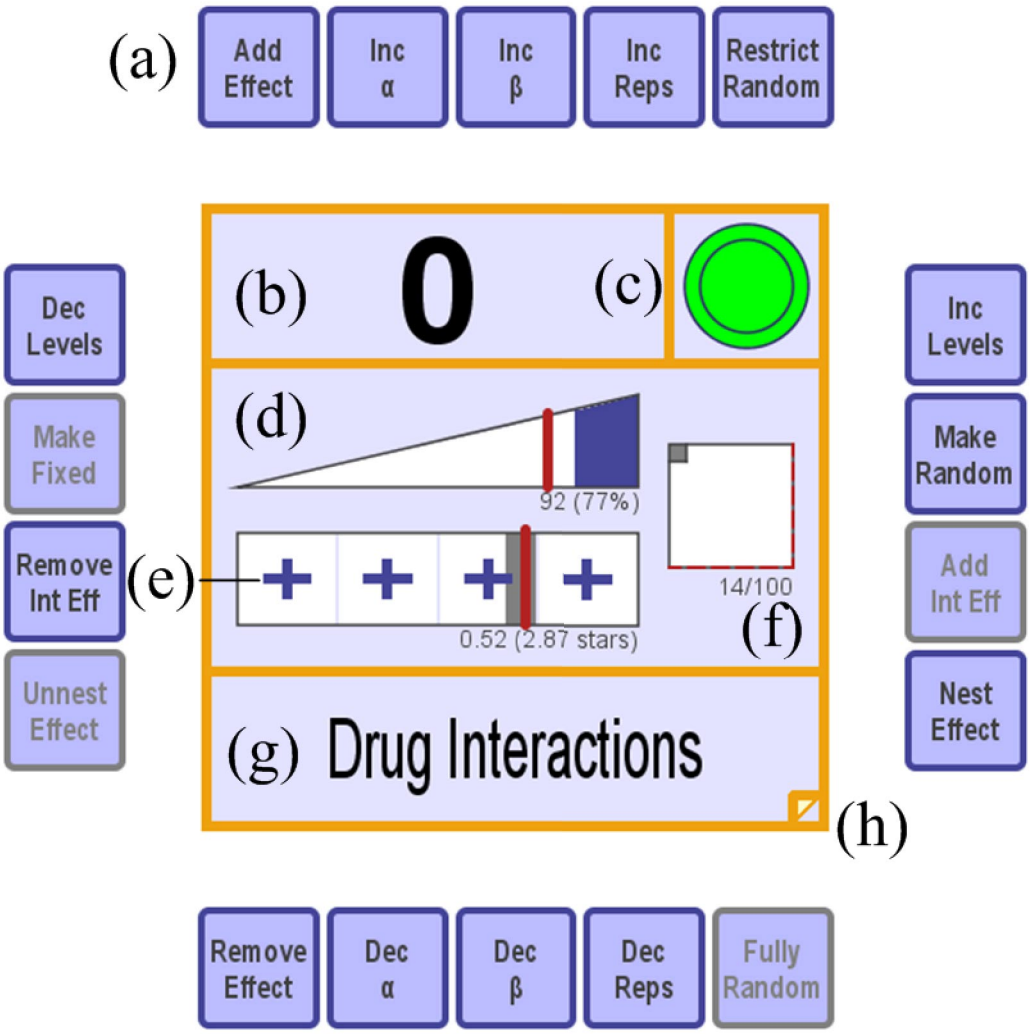



Figure 5.4: An experiment design summary: (a) the halo menu presenting the available moves; (b) design serial number; (c) constraint greenlight; (d) cost; (e) detectability; (f) inference space; (g) design name; (h) indicator for an attached design rationale annotation. This is the zoomed-out view of the design featured in Figure 5.3.

Cost is indicated using a wedge, labelled (d) in Figure 5.4. The taller end indicates higher cost. A red needle indicates the cost of the present design, relative to other explored designs. A blue zone indicates the area where the design is deemed too expensive, as determined by a user-defined constraint (described later).

A value called *detectability* [167] is used to estimate the smallest effect size that a design can reliably detect. This is the ratio of the smallest detectable effect size (for a design's $\alpha$ and $\beta$) to the standard deviation. Compared to the standard approach to power analysis, this has the advantage that the designer does not need estimates of the standard deviations of the effects to use the system. However, it has the disadvantage that the designer's qualitative judgements about the design's power are less precise [155]. Still, it suffices for many purposes, and standard power analysis values are easily obtained if the user can estimate the standard deviations.

Within a research domain, the interpretation of detectability values is non-linear. For example, all detectabilities above 1 standard deviation may be considered equally bad, while the difference between 0.5 and 0.75 standard deviations may be considered substantial. To assist the designer in making fast qualitative judgements, the designer can specify a function that maps raw detectabilities to an abstract rating from zero to four *stars*. An interactive function editor and a library of built-in domain-specific functions let the designer tailor this mapping to their research domain (see Figure 5.5). Alternatively, an arbitrary function can be defined using script code.

The abstract detectability is represented in the design summary as a rectangle containing four Greek crosses—see label (e) in Figure 5.4. (Using this simpler shape rather than actual star shapes makes it easier to compare mean ratings.) A needle indicates the mean star rating for all of the effects, while a gray region indicates the range over all effects. Individual detectability bars for each effect are available in the zoomed-in design representation.

The inference space of a design describes how widely the results can be generalized beyond the groups used in the experiment. A numerical estimate of the relative inference space of each effect is computed by considering its type and number of levels. The inference space of some effects includes every possible instance of that effect; these are assigned an extremely large value to stand in for infinity. The logarithms of the estimates of each effect are then combined to determine the estimate for the entire design. This measure provides a useful estimate of the effect of moves on the inference space, although most

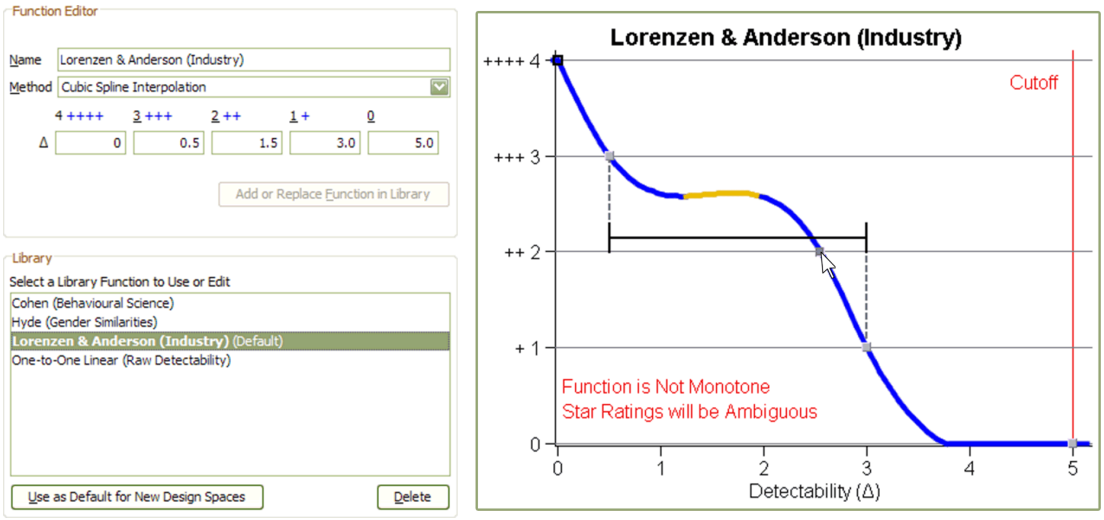



Figure 5.5: The editor for detectability mappings. The subjective interpretation of detectability values is changed by dragging points on the curve to define a function from raw detectability values to star ratings.

designs have additional considerations that the model cannot account for because they arise in the research domain.

Inference space is indicated using a stack of two squares sharing the same upper-left corner—see label (f) in Figure 5.4. The proportion of the bottom (white) square covered by the upper (gray) square indicates the inference space of the present design relative to the largest inference space seen. A dotted red line indicates the best inference space that the design could achieve without adding any more effect levels.

Some research domain-dependent consequences manifest in all research domains, yet vary in their specifics between domains. For example, the researcher may really only be interested in testing some of the effects in the design (with the rest necessary only to accurately model the phenomenon of interest). I provide support for these kinds of domain-dependent trade-offs by allowing the user to describe them as constraints for the designs to meet. A pair of concentric circles in the upper-right corner of the design summary provides continual feedback on how well designs are meeting the constraints—see label (c) in Figure 5.4. When no constraints are met, neither circle is lit. If some constraints are met, the outer circle is lit (drawn in green). Meeting all constraints lights both circles: such designs are said

to be *greenlit*, from the expression "give the green light" which refers to giving a proposal approval to proceed.

By summarizing key consequences of the design as a constant amount of information presented in a standard layout and format, the designer's cognitive workload will be greatly reduced compared to working with designs in the data-heavy and variably-sized detail view. Moreover, the design summaries provide accurate estimates of important consequences that inform much of the decision-making during experimental design. This combination supports the rapid creation and comparison of alternatives.

## 5.10 *XDS* in Use

To conclude this introduction to *XDS*, I present a small example to illustrate how the features fit together, and in particular to show how its representation of design structure is used to support exploration. The example is based on planning a small HCI experiment. The final explored space is shown in Figure 5.6. In addition, videos showing *XDS* in use are available online [123].

The experiment is intended to estimate the effects of integrality [120] and directness [117] for two interaction techniques. The designs all share three effects of interest: *technique*, *directness*, and *integrality*, each with two levels. There will also be an effect to represent the human *participants*, with 8 levels initially. Much of the design process will consist of varying the number and arrangement of the participants.

For the sake of brevity, the example uses only a subset of the available features of the system. The primary goal of the session is to strike an acceptable balance between the detectability of the design and its cost. The inference space will not change during this particular session. Only the participants effect has the potential to change the coverage of the inference space, since it is the only effect for which the number of levels is varied. However, the participants are considered to be representative of all people generally (i.e., participants is a random effect). For the purpose of estimating the inference space, this effect is treated as having an infinite number of levels regardless of the actual value.

To get started, we create a base design by adding the effects of interest. This is done by performing a sequence of four deep moves (one for each effect) on the initial Empty Design (Design 0). We add the effects using deep moves because we are setting up the space rather than exploring. Using deep moves replaces the Empty Design with our starting design.

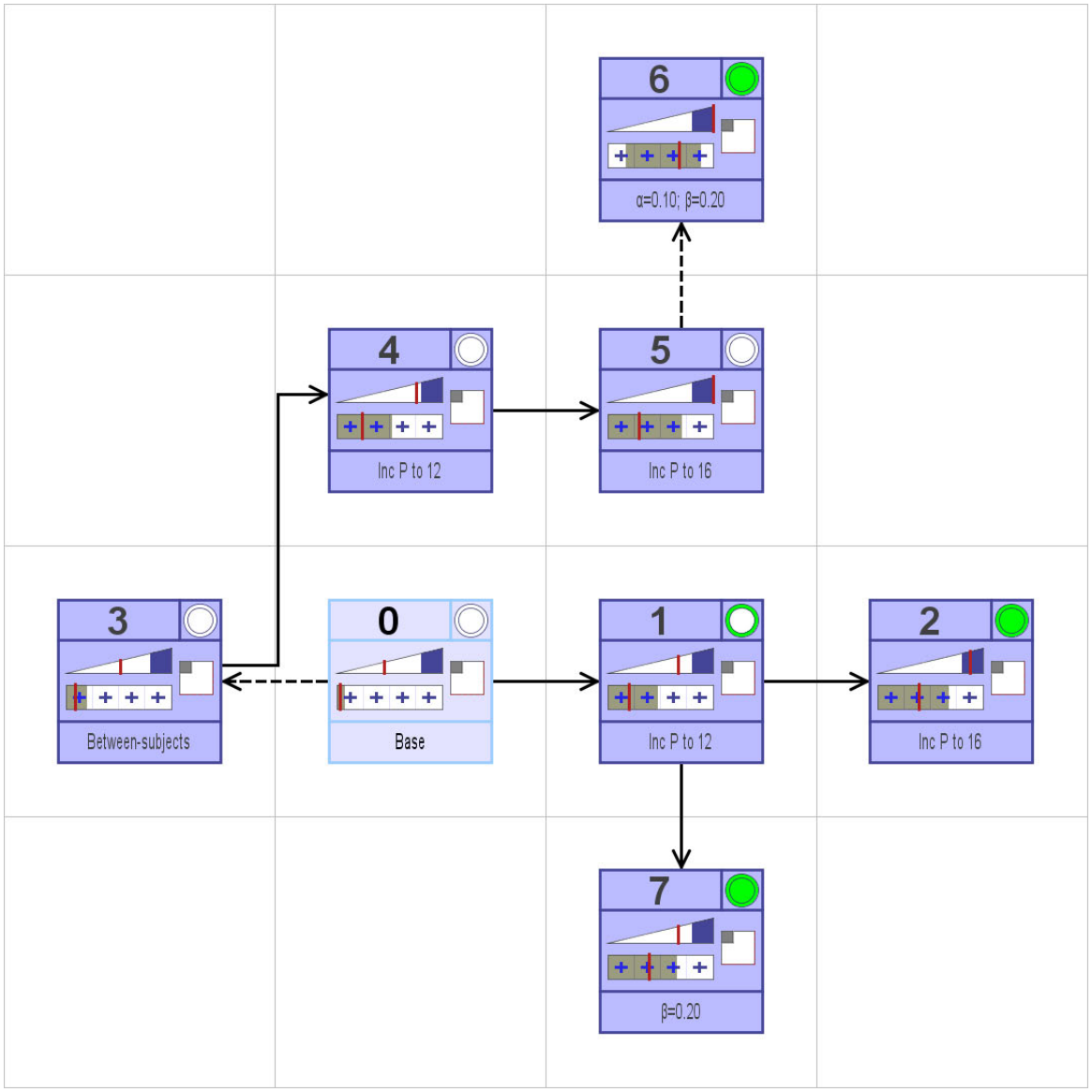



Figure 5.6: An example of an *XDS* design session. *Note:* Reproduced by permission of the Publishers from "Supporting Expert Work Processes", in *Electronic Performance Support: Using Digital Technology to Enhance Human Ability* eds. P. G. Barker and P. van Schaik (Aldershot etc.: Gower, 2010), p. 257. Copyright © 2010.

To perform a move, the mouse pointer is first moved over the target design, which causes the design to be surrounded by its halo menu of applicable moves. When a move is selected, a small dialogue window appears over the selected halo button in which the parameters of the move are specified. The move is then performed and the explored space is updated to display the result.

Once we have replaced the Empty Design with our starting design using deep moves, we are ready to begin exploring. Our main research interest is in differences between the techniques. We want the experiment to detect differences as small as 0.5 standard deviations, so we set a constraint that a design will only be acceptable if it can detect a value that small. We are not as interested in the other two effects, so we set their constraints slightly higher (0.6 standard deviations). We also set a constraint for the cost consequence that reflects our budget.

The starting design (Design 0) meets none of the constraints, so further refinements are necessary. We start by increasing the number of participants to 12. This is done using a standard shallow move application, so a new design branch is created which displays the result (Design 1).

At this point our attention is interrupted as we realize that we may have made a mistake while setting up the experiment by forgetting to set the number of levels for one of the effects. We momentarily zoom in to the detailed design state representation to verify this by checking the number of levels of the effect in question. To correct the mistake, we apply a deep move to Design 0 that changes the number of levels in the erroneous effect to the correct number. Unlike a traditional undo mechanism, performing a deep move on Design 0 propagates the correction to Design 1 since it is a child of Design 0 and does not override the number of effect levels. After verifying the correction, we zoom back out to the design summary representation to resume exploration.

Getting back to our new design branch, we find that increasing to 12 participants brings the detectability of directness and integrality to .51, so the outer rim of the green light is lit, but the centre remains off because the constraint for technique is unsatisfied. The new design is also more expensive, as we would expect—the cost bar clearly indicates the magnitude of the increase. Increasing participants to 16 (Design 2) improves the detectability (and increases cost) further. This design completely fills the green light, meaning that all of our constraints are met. We finally have the necessary sensitivity for the technique effect.

We wonder how this approach would compare to an alternative configuration that engaged some collaborators to perform each level of the directness effect at a different site, for a between-subjects design (Design 3). This requires making several concurrent changes to the design. We perform the first of these as a shallow move to start a new branch, then perform the remaining moves as deep moves on the new design. The dotted connecting line to Design 3 indicates that it is a product of multiple moves. (If we wanted an exact comparison against the existing alternatives, we could also have used a branching move.)

With only the 8 participants inherited from the base design, the detectability is substantially worse than any design so far, as shown by the low range of values in the detectability bar. We again increase the number of participants to 12 (Design 4) and then 16 (Design 5). Neither of them has sufficient power for any of the effects. We could increase the number of participants further, but at 16 the cost is already higher than we would like.

Instead, we decide to find out what we might gain if we accepted a higher risk of erroneous results. Increasing $\alpha$ to .10 and $\beta$ to .20, we find that the detectability for all effects finally falls within the desired range. Overall, the between-subjects design seems distinctly worse than the within-subjects design: even given 16 participants, we still have to accept a substantially higher level of risk to get our desired detectability. As a result, we will probably choose to abandon the between-subjects line of exploration.

Our attention returns to the original within-subjects line. The last design in that branch has the sensitivity we want, but it is rather expensive. We wonder if we can find a way to use the cheaper 12 participant version (Design 1). Pointing the mouse at this design, its move halo appears. On scanning the available moves, we are reminded that we just solved a similar problem in the between-subjects line by accepting a higher risk of erroneous results.

Operating on Design 1, we try relaxing just the $\beta$ value. (Increasing $\beta$ is generally less risky than increasing $\alpha$, because $\beta$ is the risk that we incorrectly conclude that there is no significant difference.) The result (Design 7) is observed to have the same detectability as the 16 participant design, but the same cost as its parent.

This analysis allows us to make an informed trade-off of cost and risk for this design. Whereas rules of thumb only give general directions (increasing participants always reduces the risk), we now have estimates of how much expense we will have to pay to reduce our level of risk a fixed amount.

# Chapter 6

# An Evaluation of *XDS*

## 6.1 Introduction

The design of *XDS* diverges widely from that of typical contemporary applications. Instead of editing a single design, it explicitly represents the explored alternatives within the larger problem space. Instead of leaving novice users to flounder about with an unstructured search process, it makes the user aware of ways to transform alternatives and provides feedback on how this exploration affects important consequences. This chapter continues the discussion of *XDS* by positioning it within the overall discussion of the design principles derived in Chapter 3. This is done first by relating the design back to those principles and then by presenting the results of a small user study that evaluated the effect of *XDS* on the experimental design process. As shown below, *XDS* provides some degree of support for all of the design principles. During the study, *XDS* users considered more alternatives, and a wider variety of alternatives, than in their normal practice. Taken together, these facts form the initial existence proof needed to argue for the effectiveness of the principles, and thus form an important supporting column of the factual matrix outlined in Chapter 4.

## 6.2 *XDS* in Light of the Principles

*XDS* supports the ten design principles presented in Chapter 3. This section recalls those principles and briefly discusses how they are supported. For reference and comparison, Table 6.1 summarizes this information by ranking the support in *XDS* for each of the principles as one of *none*, *low*, *moderate*, or *high*. (Here, none means "nothing more than

Table 6.1: Degrees of Support for the Design Principles in *XDS*

| Design Principle | Support |
|---|---|
| 1. Make Partial Solutions First-class Entities | high |
| 2. Support Problem and Solution Matching | moderate |
| 3. Allow Subjectivity and Ambiguity | moderate |
| 4. Prefer General, Flexible Actions and Representations | high |
| 5. Engage Multiple Ways of Doing and Thinking | high |
| 6. Support Forming and Testing Hypotheses | high |
| 7. Encourage Parallel Exploration of Breadth and Depth | high |
| 8. Provide Rich History Mechanisms | high |
| 9. Assist the Construction and Use of Repertoires | moderate |
| 10. Create an Effective Environment | low |

that found in typical contemporary applications". Since *XDS* supports all of the principles to some degree, Table 6.1 makes no use of the *none* ranking. It is mentioned for completeness since it will be needed later by Table 9.1. Note that these categories are not intended to be formal; the table simply provides a useful summary of the running text.)

Although *XDS* does implement all of the principles to some degree, it does not implement them all equally strongly. This is in part because the focus was on meeting the three subgoals identified for experts in this particular domain, but more generally it reflects the fact that interaction design is an ill-structured problem. The principles act as soft constraints on the design, and it is unlikely that they can be simultaneously optimal. Nonetheless, *XDS* provides a high degree of support for most of the principles, and a low degree of support for only one. Moreover, it provides this level of support primarily through just a few carefully integrated techniques. This is a clear indicator of the principles' practicality when applied to real design problems.

### 6.2.1 Make Partial Solutions First-class Entities (High)

*XDS* explicitly presents the space of explored designs rather than the traditional document view. The normal application of operations creates new solutions instead of replacing a single state. This keeps solutions available so that they can be combined, borrowed from, and transformed at will, without the need to recreate them or to explicitly mark them for future use (as with Save As). Although the design space exploration model constrains

each design to a valid point in the space, it does not constrain those designs to completely represent the design problem: for example, each space starts with the Empty Design, which is technically valid but has no practical value.

### 6.2.2 Support Problem and Solution Matching (Moderate)

This is supported mainly by allowing the user to set and change constraints to describe the problem. The support is not as complete as with solution generation, because the constraints apply globally: the user cannot explore the problem space in the same way as the solution space. Still, the explicit space representation does assist the user in matching the spaces. The tremendous freedom in the solution space makes it easier to bring the problem and solution space together than if movement was equally restricted in both spaces.

*XDS* only provides explicit support for aspects of the problem that can be expressed using the provided constraints. Although these are fairly flexible (for example, the cost system can be used to model the fatigue of human participants), there may be aspects of the research domain that are too domain-specific to be adequately represented. In this case, the only support in-program is to document the issues through the annotation system. A possible extension of the system would allow the definition of scripted constraints, although these too would be of limited use unless the script writer can also attach arbitrary data to designs in order to encode unrepresented problem features.

### 6.2.3 Allow Subjectivity and Ambiguity (Moderate)

*XDS* allows subjective interpretation for the consequences that it explicitly supports by presenting them visually, using indirect computations, instead of by presenting raw data (although this is also available). The star ranking system of detectability values is particularly flexible, as the user can freely define the mapping of these values.

Although the use of numeric constraints sets hard limits, these are only directly enforced by the constraint indicator. For example, if cost exceeds its constraint, the user can judge visually whether the magnitude of the excess is small enough to ignore.

### 6.2.4 Prefer General, Flexible Actions and Representations (High)

The use of a design space explorer model ensures the completeness and generality of the representation and operations used for designs. With respect to the user, the ZUI supports a

range of representations along a spectrum from a primarily structural/chronological overview (low zoom), to a primarily visual interpretation of designs (moderate zoom, design summary view), to a primarily symbolic interpretation of designs (high zoom, design state view).

### 6.2.5 Engage Multiple Ways of Doing and Thinking (High)

*XDS* makes few assumptions about methods, other than which consequences will be most generally applicable. It simply provides a space and a set of operations. While emphasizing visual exploration through design summaries, it does not exclude other approaches. For example: The two design views allow the user to engage in problem solving with an emphasis on visual or symbolic methods. The connecting arrows between designs show how the designs are related (favours soft style); displaying the entire explored space helps the user to explore methodically (favours hard style).

*XDS* also supports the technical sense of this principle. Designs can be exported to XML, and a scripting language is available to extend the application's functionality. For example, one script allows the user to invert the detectability consequence by stating a desired star rating and then automatically varying the number of levels of a chosen effect (typically representing participants) to achieve that rating.

### 6.2.6 Support Forming and Testing Hypotheses (High)

*XDS* supports forming and testing hypotheses for three key consequences in experimental design, as well as the extent to which a design fulfils user-defined constraints. Because of the explicit design space, the user can suspend their current activity at any time to test a hypothesis based on the current or any other design, and obtain immediate feedback on the result. The design summary view, and to a lesser extent the design detail view, allow the user to quickly interpret the outcome of such tests.

### 6.2.7 Encourage Parallel Exploration of Breadth and Depth (High)

In *XDS*, encouraging exploration starts by removing the barriers in typical applications that serve to *discourage* exploration. In most applications, editing is easy but exploring takes extra work: one must save copies of the work if exploration is to be non-destructive, and significant effort is required to manage and compare alternatives. In *XDS*, the commands that would normally be used to edit are instead used to explore. There are no extra steps

or safety precautions, and outcomes can be immediately compared side-by-side; This allows the user to rapidly, fearlessly, and effortlessly create alternatives.

Presenting the designs as a space promotes the user's awareness of exploration as a deliberate process. Combining the explicit design space with halo menus creates an environment where each alternative continually prompts the user to consider further exploration. Because the set of moves is complete, the user is prompted not only to make small adjustments that explore a given solution more deeply, but to try out unfamiliar or more radical approaches—to think about using moves in new and innovative ways.

### 6.2.8 Provide Rich History Mechanisms (High)

An explicit design space and deep moves give the user the ability not just to revisit history, but to edit it at will. Branching moves allows the user to replay any part of the history tree with different parameters.

### 6.2.9 Assist the Construction and Use of Repertoires (Moderate)

Because it captures each design that is created, *XDS* can be used to rapidly build up a repertoire of designs. These designs can also be used easily as starting points, because of a command that can take any design and use it as the Empty Design in a new space. Additional commands make it easy to reuse designs in different contexts (for example, effects can be renamed to suit their new purpose). It is also possible to print designs or design spaces, which may be useful for designers that prefer to surround themselves with physical artifacts. However, *XDS* does not provide direct support for repertoires. For example, it does not come with a library of designs or directly support the collection, categorization, or searching of previous designs or spaces.

### 6.2.10 Create an Effective Environment (Low)

Performance on spatial tasks is improved by the use of physically large displays, in part due to the immersiveness such displays afford [253]. Although *XDS* cannot choose the display it runs on, it is designed to maximize its coverage of the available space by creating a window that is as large as possible and using nearly all of that area to display the explored space. This would increase the effective size of the display relative to other applications. In addition, the window content is at all times focused on the explored space; except for a

small menu strip and document selector across the top, there are no other panels to distract the user's attention. The only other windows are dialogues that pop up to gather additional information in response to user actions; the explored space is never completely occluded. The large window, the ZUI, and the space exploration metaphor may all contribute to increased immersion regardless of display size, though the effect would likely be enhanced by the use of large displays. The lack of potentially distracting windows or panels may also help the user enter and sustain flow. In short, *XDS* takes a few simple steps to get the most it can out of one environmental variable (display size), but it does not actively enforce any environmental conditions.

## 6.3 A Qualitative Study

I conducted a qualitative study to assess how *XDS* might affect the design process of experiment designers. Considering the broadness of the goal, there was significant danger of the instructions or the choice of activities creating bias. For example, if the instructions used suggestive terms such as "alternative designs," or if they focused too heavily on exploring alternatives, participants might be prompted to spend more time generating alternatives than they otherwise would have. To minimize this potential, I provided participants the opportunity to use *XDS* with minimal instruction in a loosely structured session using a think aloud protocol [159, 240].

Although any observed usability issues were noted, the focus during these sessions was not on usability evaluation but on observing the design process adopted by participants while using the prototype.

### 6.3.1 Participants

A total of five participants took part in the study. The participants represented a wide range of research domains and previous experimental design experience. The domains included academic HCI research, quality control and product testing in the food industry, pharmacoeconomics, and health care research. Because participants needed specialized knowledge, they were directly invited to take part based on recommendations from other expert workers.

Two participants were novices in experimental design using ANOVA experiments; though they had a basic statistical background and commonly designed experiments as part of their jobs, they had practical experience with only a small range of research problems and

experiment designs. Two participants were highly experienced (one taught a graduate-level experimental design course). The remaining participant was at a high-intermediate level, having constructed many designs within his own domain but lacking the breadth of experience of the experts.

### 6.3.2 Methods

Each participant took part in a single session of approximately 90 to 120 minutes length after being given a brief summary of the procedure and signing a consent form. The sessions consisted of three parts:

In the first part, participants were interviewed about their typical design process and the kinds of experiments they typically perform. Information about their typical design processes helped to establish a baseline for comparison. Information about typical experiments was later used to communicate *XDS* concepts using familiar language, and to build an example design. The length of the interview varied with each participant, while the remaining two parts were alloted approximately 30 minutes each.

In the second part, participants were given basic training with the tool. I worked with the participant to set up a basic design based on a typical problem in the participant's practice. During this process, I explained how to apply moves to change the design, how to use the ZUI, and what the consequence summaries represent. The initial design was set up using only deep moves. At the end, a single shallow move was performed to demonstrate how the software would let them create new designs without "forgetting" older ones.

In the third part, the tool was turned over to the participants who then worked with the system using the think aloud protocol. No particular course of action was suggested—rather, participants were simply instructed to "try the system out." No other details on the theory of the prototype's operation were provided, although additional program features were sometimes described in response to questions or actions from the participant.

Because multiple parallel designs are such a fundamental part of the design of *XDS*, it was important to observe the participants working with multiple designs. For this reason, I was prepared to inject a prompt such as, "what do you think would happen if you tried doing $x$ to this design?" if participants did not generate parallel alternatives on their own after a reasonable period of time. In practice, this was unnecessary, as all of the participants explored the space enthusiastically without prompting.

### 6.3.3 Results

Confirming the core hypothesis, the participants created many alternative designs. Three participants extended the co-developed example for the entire session. The other two developed fresh designs, starting with an empty design space. Of the second group, one developed a design based on an experiment they would soon be planning, while the other created an abstract design that was used to experiment by trying various moves to discover the effects.

All of the participants developed and tested hypotheses about design consequences at some point. Some of the typical comments made during exploration that indicated this hypothesis testing process include: "Let's see how much this will hurt the power."—"I like the power on this one, but I want to make the cost go down."—"OK, so now if I change $\alpha$, will that affect the cost?"—"How can I get this effect to have a test?"—"It doesn't seem to make much difference to make something random if you only have two effects."

All participants but one used the consequence visualizations to work on a realistic design problem. All four of these developed a design that was closer to the trade-offs they wanted.

Once they were comfortable using the basic design moves, all but one participant asked the interviewer to explain some of the moves that had not been demonstrated in the second part of the session. The participant that did not ask was one of the experts, and so may have already grasped their purpose. Typically, after asking about one of these "advanced" moves, the participant would immediately try to find a way to apply it.

All of the participants tried most of the moves available at some point in the session, and everyone tried at least one of the "advanced" moves. The major exception was the move to nest an effect, which only one person tried but three people asked about. However, the nesting move is arguably the most difficult to understand and the least widely applicable.

All but one of the participants learned to make appropriate use of deep moves to hide interim design changes. One participant (an expert) chose shallow moves almost exclusively, occasionally commenting, "I know I could use a deep move here, but I'm interested to see what happens."

All of the participants used the design consequence summaries. Only two people were observed to refer to the inference space visualization, and two people asked the interviewer to explain the concept again during the session. In contrast, no one asked for the detectability or cost visualizations to be explained again, although many people asked technical questions about how they are computed. This difference is not surprising, as the inference space is

an abstract concept which most of the participants had never considered before. Given additional practice, I expect that most designers could learn to use it effectively.

Although the focus of the study was on the use of moves and the consequence visualizations rather than the layout of the design space, I did notice that out of about 150 designs that the participants collectively generated, there were only 3 instances where they appeared visibly confused about a navigation task. I also observed several instances where the participant appeared to quickly navigate to far off branches by first visually searching back through the design history to locate their destination. The automatic spatial arrangement does not appear to significantly hinder performance.

Overall, I was pleased by how many designs participants explored, and by how readily they not only experimented with new moves, but also sought to apply the designs and moves to actual problems of interest. Most importantly, they did not restrict themselves to a narrow range of options, trying both a wider range of approaches and more designs overall than they reported using in normal practice. This confirms the value of the design principles for promoting the consideration of additional designs, enabling the discovery of more creative designs overall.

To temper this with some caution, I note that the participants were using a novel interface to work on hypothetical designs, and there were no consequences if they made a mistake. They might revert to more familiar methods if they were working on real designs. However, even within the restricted context of one hour of working with the explorer, users learned techniques that they said might change their future work. Two users spontaneously reported that they left the session with a better understanding of the consequences of their design choices. In addition, one of the expert users reported that in the past he had designed with fixed effects instead of random effects because he believed fixed effects gave better sensitivity, although at reduced generality of the results. However, after comparing the two approaches with the design explorer he concluded that the differences were not as big as he thought (for designs in his domain) and that he would likely try using random effects in his next design. These statements suggest that even this brief exposure to *XDS* changed how some participants will design experiments in the future. It is likely that more pronounced changes would occur if they were actually using the explorer when designing their next studies.

## 6.4 Discussion

*XDS* implements all of the design principles to at least some degree. Its design was based on observations that experiment designers could benefit particularly from support that encouraged more exploration and helped them to understand important design consequences, and consequently it emphasizes three features within those principles: make the designer aware of the design options; provide a structured context for design; and provide fast, accurate feedback on the design-level consequences of decisions. A study verifies that actual designers using *XDS* did consider more designs, and that they were able to move design consequences in desired directions, and improve their designs. There are two principal ways to generalize this work: by considering its application to actual experimental design, and by applying the principles to other domains. I also discuss potential improvements and changes to the design based on lessons learned from the experiment and other use.

Although all but one of the participants worked on a realistic problem, it remains an open question whether using the explorer would lead to more cost-effective experimental designs. Generating more designs is neither necessary nor sufficient for producing better designs. Since no algorithm exists for finding better designs, there is no way to know with certainty whether a better design exists. Perhaps most people are already using good designs and would not be able to find ways to improve them. This seems unlikely, however, given that even an expert designer had his assumptions challenged by a brief encounter with *XDS*.

The interviews and the experimental results both indicate that experiment designers try relatively few alternatives, and that the main reasons for this are lack of awareness of the alternatives and the difficulty of comparing them. *XDS* allows the designer to quickly generate and compare alternatives, and it focuses on presenting the kinds of measures that the designer needs to judge the quality of the design. The reasonable conclusion is therefore that if *XDS* encourages designers to make a thoughtful exploration of the design space, they are far more likely to find a design that is better tailored to the specific problem if one exists, or to confidently conclude that finding a significantly improved design is unlikely.

One criticism that has been aimed at previous design explorers is that they only solve simplified toy problems [4]. By contrast, *XDS* is not a toy: it features a complete representation of a design domain which (as discussed in Chapter 5) has substantial real-world impact. That said, many interesting design domains are not well-suited to a design explorer implementation: they may be too ambiguous to construct an effective model, or else the

moves of interest may be too dependent on the specific task. Although *XDS* uses a design exploration model as a foundation for implementing the design principles, other tools might take another approach. Alternatively, support might be restricted to a well-defined subset of tasks. As a practical example of the latter, consider the refactoring tools provided by many software development environments. Although software design in general is not well-suited to a design space representation, refactoring tools are effective at performing a classic design explorer task: automating the generative details of a design change. They can do this because they make small-scale, concrete design changes to a low-level but sufficiently formal representation—they do not have or require a representation of the high-level design.

It also seems reasonable to assume that providing fast, accurate feedback on design consequences would be effective independent of the other two principles. The *Strange Eons* application, described in Chapters 8 and 9, takes this approach. Using consequences to motivate design choices could be practical in many design domains that lack the structure needed for an effective design space representation.

Finding good design consequences can present its own difficulties. Clearly, the appropriate consequences must be evaluated separately for each domain based on the factors that influence decision making—a tool that supports Web site design might use consequences such as download time, word count, the reading difficulty of the text, the complexity of the navigation graph, or its suitability for disabled persons using a screen reader.

For some domains, it may not be possible to evaluate the most useful consequences automatically: extracting a summary of the plot of a story, for example. Sometimes it is possible to approximate the consequence or to provide the consequence in a simplified form that misses some details but is still useful in guiding decisions; otherwise, those consequences must be abandoned. In the latter case, it should still be worth supporting second tier consequences: the result is still better guidance for making design decisions.

The arrangement of small, incomplete, visually-oriented design summaries together on a large white space is intentionally similar to sketching. Although the benefits of sketching are incompletely understood, *XDS* captures some of its features, including imprecision (the design summaries) and the juxtaposition of alternatives. The layout of designs is automatic, but even arbitrary arrangements have the benefit of suggesting new relationships [252]. Indeed, arbitrary arrangements could be *more* helpful than intentional ones if progress is being blocked by a mental set or fixation. And while unconstrained layout might offer advantages, they would come at a cost since the arrangement in *XDS* presents historical

and structural relationships between designs. This is an aid to navigation and visual search, and assists the user in understanding the implications of changes to the design history.

The layout of the space is currently based on the type of move (determining the axis of movement) and similarity to the Empty Design (determining whether the movement is in the positive or negative direction). The layout of the halo buttons (see Figure 5.4) indicates the direction of the associated move. In addition to free layout, another possibility would be to organize the designs according to the history tree, moving in the same direction and creating parallel lines for branch moves. This would bias the layout towards a long, straight line rather than the fairly even coverage that tends to occur now. That could help the user better understand the history tree, but it would be more fatiguing to navigate over many designs, and may further inhibit the discovery of relationships.

One difficulty with using a sketchpad metaphor for the the design space is the difficulty of representing large sheets of paper on relatively small displays. In *XDS*, I found it difficult to compare designs that are far apart. Either the display must be zoomed out so far to get both designs on the display simultaneously that there is insufficient detail to compare the designs, or else the memory trace of the first design decays before the second can be reached by scrolling. To address this, I added an explicit comparison command. When the comparison button is held, it shows a selected design as a floating "paper scrap" next to the design under the cursor. Because exploration is performed through halo menus, this is the only exploration command that relies on the presence of a modal design selection. Alternatives were dragging one design to another (too cumbersome and tiring for large spaces) or creating a window that floats over the space until dismissed. The quasimodal gesture was selected over opening a window—which later requires closing—because comparisons are quick, fleeting operations when design summaries are used, and because this method allows the comparison scrap to be aligned precisely with each target, for more accurate comparison.

Another change to the design has been to add a specific command for switching between within-subjects and between-subjects designs. Although it was possible to do this with the existing command set, the required sequence of moves could be obscure and error-prone. This action could have been supported through the scripting system, but given its importance for experiments in the behavioural sciences, a dedicated command is justified.

## 6.5 Summary

*XDS* represents a broad implementation of the design principles presented in Chapter 3. The initial design placed special emphasis on three particular design goals based upon the user audience (make the designer aware of the design options; provide a structured context for design; provide fast, accurate feedback on the design-level consequences of decisions), and this is reflected in which principles received the strongest support. *XDS* focuses on these goals using a persistent "halo" of moves (described in detail in the next chapter) that directly transform one design into another, explicit traversal of the design space, and summarizing design states as visualizations of their consequences.

*XDS* represents an existence proof for the design principles. A support tool based on them can encourage the designer to consider more designs, and a wider variety of designs overall. This was identified as the greatest need for users in the domain of experimental design, but there are probably many domains with a similar exploration pattern, and it is also characteristic of novice performance. *XDS* also allowed me to explore the design space created by the principles, and to develop some useful techniques for reapplication. Some of these would likely work best in spaces that permit a design space explorer model. Others are more easily generalized. Chapter 7 looks at one of these, halo menus, and uses claims analysis [250] and other techniques to explore it in more detail. Chapters 8 and 9 provide a deeper investigation of the potential benefits of consequence displays, while also considering the value of applying a limited subset of the principles.

# Chapter 7

# Halo Menus

## 7.1 Introduction

*XDS* introduces a new interaction technique, the halo menu, which is designed to encourage users to explore alternatives. This chapter provides a separate detailed presentation of halo menus, including a discussion of how they relate to existing techniques and options to consider when adapting them to other applications.

When designing a command selection technique, there is a trade-off between the need to ensure that commands are easily discoverable and efficiently accessible and the need to ensure that they are not intrusive or distracting. Existing techniques emphasize one of these aspects at the expense of the other. They either keep commands visible (ready discovery) but separate from their target objects (reduced access—menus, tool palettes, toolbars), or they keep commands close to hand (ready access) but only visible when activated (reduced discovery—context-sensitive pop-up menus). Halo menus balance these competing goals by making a command set available around every active object but only displaying commands when the user attends to one of these objects.

Previous work on command selection techniques has focused on improving their efficiency [35, 142], and extending or specializing them for alternative input devices [96, 142]. In contrast, halo menus focus on increasing the user's *awareness of command choices* and not the efficiency with which they execute a choice once it is made. One benefit of this is to improve the learnability of the interface. However, the primary goal of the technique is to couple objects and actions in a way that encourages users to explore and experiment with the available objects. In expert support systems, this can prompt the user to generate more

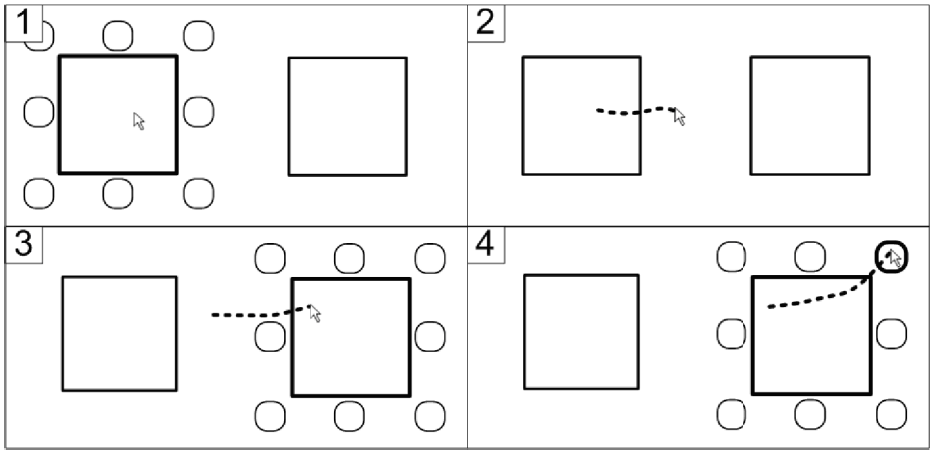



Figure 7.1: Choosing a command in a halo menu. As the cursor leaves the object it was positioned over (1), the halo menu over that object becomes invisible (2). When the cursor enters the target object, its halo menu appears (3), allowing the user to target the desired command (4).

alternative solutions than they might otherwise—in other words, using halo menus supports Principle 7 (Encourage Parallel Exploration of Breadth and Depth). The usefulness of halo menus is not limited to expert support applications, however. Any tool that is most effective when the user is motivated to try different options and alternative paths is likely to benefit: educational tools, children's software, games, puzzles, museum displays, interactive artwork, and creativity support tools are all examples.

## 7.2 The Halo Menu Technique

A halo menu consists of a set of command buttons arrayed along a bounding shape (typically a rectangle or circle) of an object in an application's edit space. A given halo menu is unique to one object; other objects in the edit space each have their own menu. As the user moves the cursor around the edit space, the halo menu of the object under the cursor is made visible, while the halo menus of all other objects are made invisible (see Figure 7.1).

Selecting a command occurs in three steps. First, the user positions the cursor over the target object and then over the button in the halo menu which corresponds to the desired action. Finally, selection is triggered using one of any number of secondary gestures, such as

clicking on the button, crossing out of the button, or gaze fixation. If the buttons are arrayed on the inside edge of the target object's bounding shape, the user can select a command with a single positioning step once they have learnt the locations of the commands.

The design assumes that the cursor's position approximates the user's locus of attention. By making menus selectively visible as the pointer reaches objects, the user is continually presented with the commands applicable to the current object, without being distracted by commands applicable only to other objects. At the same time, by tying visibility to the fluid movement of the cursor, both the general capabilities of a system and the specific capabilities of particular objects can be rapidly discovered by exploring the edit space with the cursor. This level of discoverability is ideal for systems designed for an audience of mainly one-time users, such as public displays.

Because each object has its own halo menu, the buttons do not require the user to select an object. Rather, a single fluid gesture selects an object and an action simultaneously. By continually pairing the attended object with its set of applicable commands, users are encouraged to think in terms of object transformation. Instead of interrupting their work to ask "what next?", users can browse a ready list of suggested actions without changing their locus of attention. This creates a subtle but important distinction between halo menus and other menu techniques. Traditional interaction assumes a largely one-way flow of ideas: the user first decides on a course of action (an intention), and then seeks an action sequence that will implement it [201, pp. 45–53]. By contrast, the objects in halo menus continually prompt users with possibile action sequences, some of which may catch their interest, thus encouraging more active experimentation.

## 7.3 Related Work

Existing interaction techniques favour either easy discovery of commands or rapid access to them. The classic menu bar facilitates command discovery by listing all possible commands on any object, disabling commands that are not available in the current program state. Complex states are set through subsidiary dialogue boxes that display additional widgets. These techniques for supporting discovery have made computers usable by people with a wide range of abilities.

Other techniques have emphasized rapid command access, often by bringing command and object selection into the same locus of attention. Draggable menus and toolbars can

be moved near the object of attention, but they must be repositioned as the user's focus changes to other objects. To maintain efficient access to commands, the user must manage the widget locations. In addition, draggable widgets do not indicate which commands apply to an object. The user may place any available set of commands near an object, even inapplicable ones, and the command set does not change.

Context-sensitive popup menus (context menus) are displayed under the cursor when activated. Context menus take three forms. Linear context menus, like halo menus, require the user to place the cursor within a specific area in order to select a command. Pie, or radial, menus are more efficient than linear menus as they only require the user to drag the cursor in the direction of a command [35]. Marking menus [142] extend pie menus by recognizing the cursor's path rather than its direction, allowing gesture interpretation to switch smoothly from menu activation to other activities, and further increasing efficiency.

The design of context menus commits to ease of access at the cost of making discovery more difficult. Because context menus are invisible until they are activated, the user can only determine that an object can be manipulated by trying to activate a menu. To discover available objects and their actions, the user must attempt to activate menus at various locations in a process of trial and error.

A third approach to juxtaposing objects and commands is the use of click-through tools [19]. These bring the menu to the object for every command invocation, with a button click selecting both object and command. These techniques replace the temporal modality of separate object and action selection with a spatial modality.

Like other techniques for rapid command access, halo menus place commands close to their associated object, but they do not require repositioning (or even allow it). Like click-through tools, halo menus simultaneously select an object and a command, avoiding a temporal mode. Halo menus also eliminate the spatial modes of click-through interfaces by assigning each object-action button a unique location in the edit space.

Halo menus support the discoverability of the available commands. In particular, the technique assists users who do not have a preconceived action in mind. The user can discover the active objects and commands by simply moving the cursor around the display. Available commands are displayed in context with the objects to which they apply, unlike menus that display commands separated from any objects. The process is far more fluid and efficient than the corresponding procedure using context menus. Yet once an active object is discovered, its available actions can be accessed as readily as with context menus.

Table 7.1: KLM-GOMS Analyses of Halo and Linear Menu Command Selection

| Technique | Gesture sequence[a] | Time (s) |
|---|---|---|
| Linear context menu | $H \cdot M \cdot P \cdot K \cdot M \cdot P \cdot K \cdot R$ | 5.7 |
| Halo menu | | |
| novice | $H \cdot M \cdot P \cdot M \cdot P \cdot K \cdot R$ | 5.5 |
| experienced | $H \cdot M \cdot P \cdot K \cdot R$ | 3.0 |

Legend (see [37] for details)

| | |
|---|---|
| $H$ | Time to move hand to pointing device (0.4 s). |
| $M$ | Time to mentally prepare for next step (1.35 s). |
| $P$ | Time to point at a position on the display (1.1 s). |
| $K$ | Time to press and release a key (0.2 s). |
| $R$ | Computer response time; treated as 0 s here. |

[a]These sequences assume that the target object is visible but not selected or under the cursor, and that the user's hand does not start on pointing device.

The goal of halo menus is to encourage exploration and not to maximize selection speed. The selection time of halo menus is therefore not a concern unless it will be much longer than with standard techniques. To ascertain the likelihood of this, I performed a simple KLM-GOMS [37] analysis. The analysis predicts that command selection time will be similar for both the halo and linear context menu techniques (see Table 7.1, Halo menu—novice). Unrepresented in the analysis is the fact that the distance to the target menu item is typically somewhat larger for halo menus. As a result, linear menus may gain a slight edge in practice due to Fitts's law [71]. However, this will be dampened by the fact that halo menu buttons will typically be taller than linear menu items, and thus they will count as larger targets [174] (also an advantage under Fitts's law).

If halo menu command selection speed is similar to that of linear menus, it will be significantly slower than using a pie menu or marking menu [35, 142]. While halo menus do not maximize the efficiency gains of practiced use by using directional gestures, they also lack the restrictions on number and placement of items that other radial arrangements require for maximum effectiveness [114].

Experienced halo menu users may be able to improve markedly on selection time given certain conditions. If the user can predict the placement of halo menu items (for example,

in *XDS* objects are all the same size and use the same command configuration), then they can move towards the target object and the target command in a single gesture. In this case KLM-GOMS analysis predicts that selection time for experienced users will be nearly half that of novice users (Table 7.1, Halo menu—experienced).

Overall, this analysis shows that command selection speed is comparable to that of linear context menus—and may in some cases be close to that of fast radial selection techniques. This prediction matches my practical experience with the halo menus in *XDS*. Command selection time is therefore not a barrier to the use of halo menus in appropriate contexts.

## 7.4 Implementation Choices

The basic halo menu design is agnostic with respect to many design decisions; they might better be thought of as an approach to designing interaction techniques rather than one specific technique. This section discusses some of the possible variations.

One important design decision is whether command buttons should overlap an object or surround it. If buttons overlap the object, relevant information may be obscured. If they surround it, selecting a command may require entering a neighbouring object. In *XDS*, the menu overlaps its object, but the objects include a margin that leaves space for the menu to appear. The margin is shared with the lines that connect designs, but the lines are not interactive so they do not compete for the cursor. In applications where this is impractical, the gradual activation strategy described below can allow the user time to select a command without activating neighbours, although occasional mode errors would be inevitable. Another alternative is to arrange objects so that their edges do not intersect.

*XDS* takes advantage of the menu margin in another way as well. When the view is zoomed out to a size much smaller than usual, the user is normally either moving from one part of the space to another, or else trying to survey the explored space. At low zoom levels, the efficiency of command selection becomes unimportant, while more legible design representations aid navigation and comparison. To take advantage of this, the size of the margin relative to the size of the design representation is gradually reduced as the user zooms out further and further. While halo buttons become harder to target, the design details retain more legibility—precisely matching the changes in the user's needs.

Some menu techniques require a specific gesture to select a command. Classic menus require a click. When used with a stylus, pie menus and marking menus combine object and

action selection as bookends of a single extended gesture (a temporal mode). Click-through tools follow one or more set up gestures with a single tap or click that instantaneously combines object and action (a spatial mode). Crossing interfaces [2] select commands by moving across a goal line. Because halo menus are modeless, they are neutral with respect to these styles. The designer can choose an approach which suits the rest of the design. *XDS* uses click-based selection because the commands call up dialogue boxes of traditional click-based widgets to gather command parameters.

Halo menus can be used seamlessly with traditional techniques that require explicit selection of a "current object". Nothing about the design prevents explicitly selecting objects, or from making explicit the otherwise implicit selection of choosing a halo menu command. This allows the designer to keep the number of commands in the halo reasonably low. *XDS* uses this approach to restrict the halo menu to primary commands representing moves in the design space; these commands are most likely to prompt the user with new ideas for alternative solutions. Secondary commands that apply to objects but do not change their semantics were relegated to a pull-down menu. Although the user may fail to discover these secondary commands, they do not restrict the user's range of design choices—the consequences of missing such commands are therefore substantially lower.

Should the halo menu only present commands that are available in the object's current state, or should the menu include all commands that might be applied in any state of the object? *XDS* includes all possible commands, on the grounds that it helps the user understand the full life cycle of each object, and also facilitates habituation by presenting each command in a consistent location. The choice may be less obvious in an application where the objects are not as homogeneous as in *XDS*.

Although hierarchical halo menus were not ultimately used in *XDS*, halo submenu techniques were designed and tested as part of its implementation. The final design combined goal crossing and the thoughtful arrangement of the submenu buttons to create a consistent and efficient selection mechanism (see Figure 7.2). As each object has its own halo menu, objects in a halo menu system are comparable to the top-level choices of a traditional pull-down menu. Since the root halo menu is activated by crossing the edge of an object, for consistency submenus are likewise activated by crossing the submenu's parent button. The submenu should remain visible until the user selects a submenu command or leaves the bounding rectangle of the union of the parent button and all of the children—plus a reasonable margin to allow for error. Crossing into a different parent button takes precedence

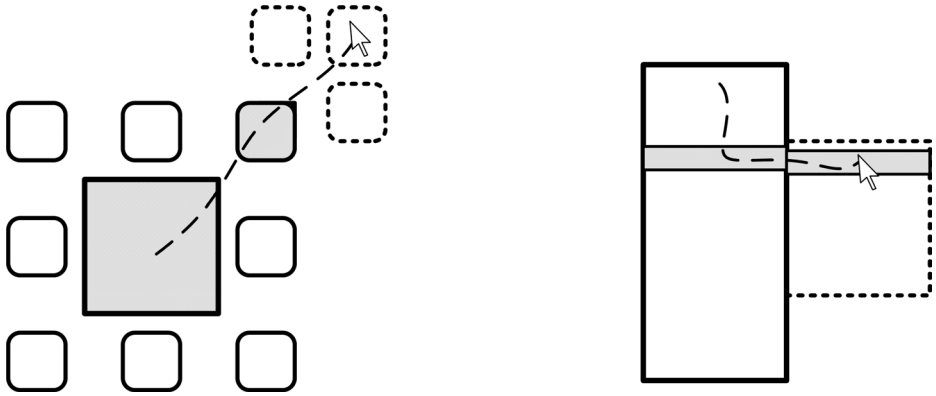


Figure 7.2: A hierarchical halo menu (left). A special button shape indicates the presence of a submenu. When the parent menu button edge is crossed, the submenu buttons (shown with dotted outlines) are displayed around the parent in a direction away from the centre of the target object. Although gesturally similar to traditional pop-up submenus (right), halo submenus are more efficient because there is no sharp turn or narrow tunnel to navigate with the cursor.

over this bounding rectangle, but ideally the menu should be designed to minimize overlap. In the test design, a modified button shape was used to indicate that a button displays a submenu; other methods, such as printing a special symbol on the button face, should work equally well. A parent's subitems are arranged around it in an arc or line to create a half halo around the parent. The centre point of the subitem halo is the furthest point from the centre of the target object that is on the parent item's halo boundary shape. This point can be adjusted for aesthetics; the key is that the submenu should appear along roughly the same line as the one from the target object to the parent item so that the cursor can pass through the parent item on the way to the target child with minimal steering. To maximize selection efficiency, the most-used commands should be placed closest to this centre point. This design is visually and procedurally analogous to the pop-up submenus used in traditional pull-down menus, but it is more efficient: the user is not required to steer through the narrow tunnel between the parent item and the submenu [1] or make a sharp turn.

The animation of the halo's appearance has subtle consequences. In the original design for *XDS*, the visible set of controls switched immediately when the cursor moved over a different object. Since the controls are anchored to the objects they act upon, this led to a disorienting and distracting jitter effect when the cursor was moved between objects. As the

cursor was moved near the edge of an object, it would often momentarily pass into another object's scope. As a result, the halo buttons would disappear from the object that was the focus of the user's gesture, flash over the second object, and then reappear over the focused object. Users interpreted this as a single set of halo buttons that appeared to move away and then return for no apparent reason. This effect particularly frustrated users because the halo buttons are near the edge of an object. When pointing at a halo button, users would often overshoot the button and end up over a nearby object. As a result, the halo buttons appeared to jump away from the cursor just as it neared the target.

In response to this miscue, the design was changed so that an object's halo buttons fade in when the cursor enters the object's scope, and fade out again when it leaves. Under this gradual activation strategy, target buttons remain visible long enough for the user to complete a pointing action that overshoots the target, and the simultaneous visibility of both sets of buttons as they cross-fade clearly indicates that they are independent entities.

## 7.5 Supporting Evidence

Chapter 6 presents a qualitative study of *XDS*. The halo menu technique was not specifically evaluated, but several results suggested a link between the use of halo menus and a pattern of increased exploration and experimentation. A few of these are highlighted here rather than repeating the results in detail.

All of the participants experimented enthusiastically and tried many combinations of commands. All but one of the participants (a domain expert) asked the experimenter to explain one or more unfamiliar commands that they had seen in the halo menu. When these commands were explained, participants would typically roam over their previous designs looking for an opportunity to apply the new command realistically. All of the participants had tried most of the available commands before the end of their session.

Although the halo menu design is based on the assumption that the user's locus of attention is approximately represented by the cursor, the extent of this connection was surprising. Over the course of their session, the halo menus appeared to train users to synchronize the cursor with their locus of attention. By the end of the session looking and pointing had become nearly synonymous for all participants, and comments about designs nearly always referred to the design under the cursor.

## 7.6 Limitations of Halo Menus

There are several limitations to this technique. Because halo menus are always active, a halo will be visible whenever the user attends to an interactive object. When there are many such objects, this may lead to a cluttered display. If the user wishes to focus on objects rather than actions (perhaps due to engaging reflection-on-action), the user may be frustrated by the difficulty of hiding the menus if there are few dead zones or their location is not readily guessable.

Halo menus cannot be used for commands that do not work on individual objects. Some objects also defy placing a halo around them. This includes objects such as the desktop, which covers the entire display, as well as objects so small that they do not have enough room around them to display the buttons. In these cases, other techniques for entering commands have to be used.

Allowing overlapping objects is also problematic. For halo menus to be effective, the cursor location must unambiguously identify an object. If one object can completely occlude another, a mode must be introduced to ensure that both objects are selectable.

Although halo menus help users discover the actions possible with a given object, in comparison with fixed drop-down menus they may make it more difficult to form an understanding of the system-wide command set (all the things you might ever do to any object).

Halo menus are hidden until the user moves the cursor over an object. However, this is unlikely to be a significant detriment in practice. Users that expect the noun-verb command paradigm common in graphical interfaces will move the cursor over the target object as a preface to selecting it, thus making the menu visible. Based on similar reasoning, interaction designers at the search engine company *Google* recently changed the design of their main Web page to incorporate progressive disclosure. Noting that the vast majority of users visit this page to perform a search, the page was changed to initially display only the company logo and a search field. It is only when the user moves the cursor that links for other services and other elements are faded in. The designers reasoned that users wanting one of the less-accessed features would generally move the cursor anyway [176], in order to select the relevant link.

## 7.7 Summary

Halo menus are a novel interaction technique for context-aware modeless command selection. They feature self-promoting objects that continually but unobtrusively prime users with the actions possible at the user's locus of attention. This is useful in situations where the user may be unaware of the full spectrum of choices available to them, or may be reluctant to try unfamiliar techniques. A qualitative evaluation suggests that this technique may increase the range of distinct choices made by novice designers.

When used in an expert support system, halo menus are a simple but powerful way to support Principle 7 (Encourage Parallel Exploration of Breadth and Depth). They should be most effective when combined with a system that also implements Principle 4 (Prefer General, Flexible Actions and Representations), so that exploration can be performed with a reasonably small set of actions that fit comfortably in the halo configuration.

Halo menus make the functions of a system and its objects easier to discover and to deliberate; however, this may come with the cost of less efficient selection compared to techniques that focus on efficiency. This balance makes halo menus an ideal design choice in contexts where users will spend a greater proportion of their time exploring alternatives and considering possible actions than they will spend executing command sequences.

# Chapter 8

# *Strange Eons:* A Selective Implementation of the Principles

## 8.1 Introduction

The evaluation of *XDS* found that an expert support system designed in line with the principles can yield benefits for users. However, the expert domain supported by *XDS* has features that suggest it may be particularly well-suited to software-based support: the space of ANOVA designs is well-defined and can be explored using a small set of formalizable design moves. Evaluating a second prototype provides the opportunity to test whether the principles generalize to problem spaces where conditions are less favourable.

*Strange Eons*, a tool that supports end-user development of paper-based games, supports such a space. Game design has much less formal structure than ANOVA experimental design. Although games are constructed around rules, these only bind the player: the designer is restricted mainly by imagination and the soft constraints that the end result be fun, feel fair to the players, end in a reasonable amount of time, and not be too expensive to produce. In these respects, game design has much in common with domains like writing.

Rather than implement all of the design principles as *XDS* does, *Strange Eons* focuses on a narrow subset. This allows a more controlled evaluation, and so a more precise understanding of how these features affect expert work. In combination with other prototypes, it can also be used to start teasing apart the contributions of particular principles.

Some might argue that implementing a subset of the principles limits external validity since the selected principles may be those most likely to produce favourable outcomes. There are two responses to this charge. First, because the principles are essentially soft constraints, it is perfectly reasonable to emphasize those principles most likely to produce the best outcomes in a specific context. Second, the actual focus in *Strange Eons* is on adapting the notion of consequence displays (introduced in *XDS*, see Section 5.5 on page 60). Because the problem space is relatively unstructured, this is one of the harder problems that could have been selected. An easier choice would have been to implement a feature more independent of the problem space, such as rich history mechanisms. The result would probably be less interesting, however, as these have been widely explored (e.g., [42,100,124,125,137,233,248, 254,276]). In addition, intentionally choosing a support feature limited to providing weaker support makes a more compelling case if the system succeeds.

## 8.2 Games and Problem Spaces

Games can be classified in terms of four elements: goals, rules, actors, and resources. A goal defines what the players of the game must achieve, or avoid, in order to win; rules place constraints on how the goals can be accomplished. An actor is an agent that can change the game state in a manner intended to further progress towards goals. It need not be human—it could be programmed, for example, or it could arise from randomly selected events. Intelligent actors, or sometimes simulations thereof, are more commonly called players. Resources are used to represent the game state. While many traditional games use stones, pawns, or other simple markers, modern games often use a combination of markers, cards, and other objects. The play area (such as the board in a board game) can also be a resource, although in many games it serves only as an external memory on which to record the game state.

The game state can be a function of all of these elements: the actor (turn-taking), the number and configuration of resources, the current rules (for example, if a turn is broken down into steps or phases), and the current goals (for example, there may be a special tie-breaking goal). Game play occurs when the actors use the resources according to the rules to modify the game state in an attempt to satisfy the goals.

To qualify as a game, at least two actors must be in conflict, and at least one of these must be a player. Conflict can take many forms. The actors might be competing to reach

the same goal first, or perhaps one actor is is trying to prevent the others from reaching their goal. If no actors are in competition, then the result is better described as a puzzle than as a game: puzzles place passive obstacles (arising from the selection of rules and the initial configuration of resources) between the actors and the goals. If there are actors but no players, the result is a simulation.

Games are found in many forms: sports (games that feature recurrent skills [181] and players as explicit resources), board games, role-playing games, card games, video games, and more. Despite this variety, most games fall roughly into one of two clusters, which can be called *static* and *dynamic* in reference to the malleability of their rules. In the static cluster one finds abstract games that have fixed rules and no randomness. *Chess* is a classic example. Static games stress competition, skill, and an equal opportunity for all players to win. In these games, players generally begin the game with equivalent resources: either identical, symmetrical, or else asymmetric but balanced. An example of the latter is the traditional Nepalese game of *Bagh Chal* (बाघ चाल, "Moving Tigers"). In *Bagh Chal*, one player controls four tigers and the other controls twenty goats. However, the tiger player can capture goats, while the goats must be moved to hem in the tigers. The asymmetric number of resources is balanced by the asymmetric rules, and the game is considered fair to both sides.

Static games are primarily a competition between players. The goals are usually symmetric and mutually exclusive; for example, each player may need to eliminate the resources available to other players, so that the last player to retain resources wins. Since there is no randomness and the game is considered fair, credit for winning is ascribed mainly to the relative skill of the players.

In contrast, dynamic games feature more concrete designs, variable rules, and the use of randomness. These games emphasize simulation, discovery, socially-constructed stories, and novelty of experience. Games can still be competitive, but much of the fun in playing them is derived from these other factors. Dynamic games can also be played cooperatively. In a cooperative game, the players work together to achieve common goals and they win or lose as a group. Cooperative games are differentiated from puzzles in that there is an actor (sometimes a player) working against the protagonist players.

Role-playing games, which are cooperative, are different from dynamic board games mainly by their more extreme dynamism: they feature even more concrete, finely detailed simulation, more variable situations, and a more intense focus on storytelling. Play varies so

much that a fixed game board is no longer a practical aid for representing the game state. In these games, the opposing actor is viewed more as a referee, facilitator, and storyteller than as an opponent. This actor's task is to interpret the results of the other player's decisions rather than to actively seek their defeat.

A key feature of dynamic games is that the rules are not known completely in advance, and they change over the course of the game. This is done by mixing a fixed set of core rules with temporary rules that enter and leave play over time. Usually the temporary rules are attached to resources. It is generally intended that the player should not be able to know all of the possible rules beforehand: dynamic games aspire to be ill-structured problems (whether they actually are varies). Players tend to choose a style of play accordingly: Players of static games can select moves based on a deep analysis of the game tree because the principle unknown is the moves of the other players, which can be predicted. Players of dynamic games tend to act more in reaction to the situation as it unfolds (as described by situated action; see Section 2.4.2 starting on page 21), because there are too many unpredictable factors to perform a detailed game tree analysis.

Temporary rules can be introduced and withdrawn in a variety of ways. In role-playing games, it is primarily the referee's job to manage them. In board or other tabletop games, other strategies are employed. For example, special rules may be attached to resources that players can acquire. Randomization is also common: the players might draw new rules from a large deck each turn.

When a game is new, players will be unaware of some rules since they are distributed between the rule book and the game's resources. With repeated play, they will gradually gain knowledge of these rules. Play becomes more predictable over time, and therefore less interesting. One way board game designers combat this is to introduce rules via multiple mechanisms, so that the total number of possible combinations is vastly increased. Another is to follow the main game with expansions: optional add-ons that introduce new content to the base game. It is in this niche that *Strange Eons* fits most naturally.

## 8.3 The Design of *Strange Eons*

*Strange Eons* supports the design of components for paper-based games. Paper-based games include board, card, and role-playing games. Each of these is a large category. The U.S. market for board games alone was worth US$800 million in 2008 [230].

*Strange Eons* has been used by both amateur and professional game designers, and has been used both to extend existing games and to develop entirely new games. However, its primary purpose is to support the end-user development of new content for dynamic games. New content adds variety to thematic elements of the game, keeping play fresh by adding new discoveries, situations, and stories for the players to experience.

Adding content to a game is like writing a sequel to a story in that sequels must not only work within the constraints of the broader design space of storytelling, they must also work within the subspace defined by the original work. The sequel to a modern-day spy thriller is not likely to be set in a forest of magical talking trees. Likewise, the new rules and resources included with new game content must fit within the context of the original game: new cards should have a similar design theme and illustration style; new rules should integrate seamlessly with existing rules. Since each game implicitly defines its own design space within the larger space of all possible games, *Strange Eons* mirrors this structure with two distinct layers of support.

The first layer consists of generic tools and features that do not target particular games. This includes the plug-in framework and integrated plug-in development tools, document management, text layout, and generic support for creating, viewing, and printing components such as boards, cards, tokens, boxes, and booklets. Figure 8.1 illustrates some of these possibilities using examples created with the application.

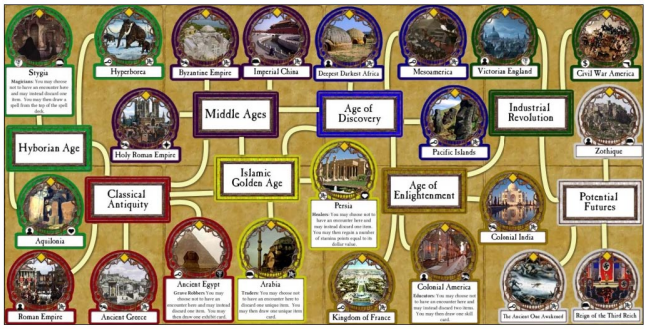
The second support layer, built atop the first, adds support for specific games. This consists mainly of adding new editors for that game's various components, but nearly every aspect of the application can be extended with specialized support.

The following subsections give brief overviews of the major components of the first layer in order to give a sense of the normal work flow and the scope and capabilities of the application, and to introduce features that will be needed later when discussing expert support. A complete survey of the features in *Strange Eons* would be too long to include here, but more information can be found by exploring the application itself or by consulting the online documentation [126].

### 8.3.1 Document and Project Framework

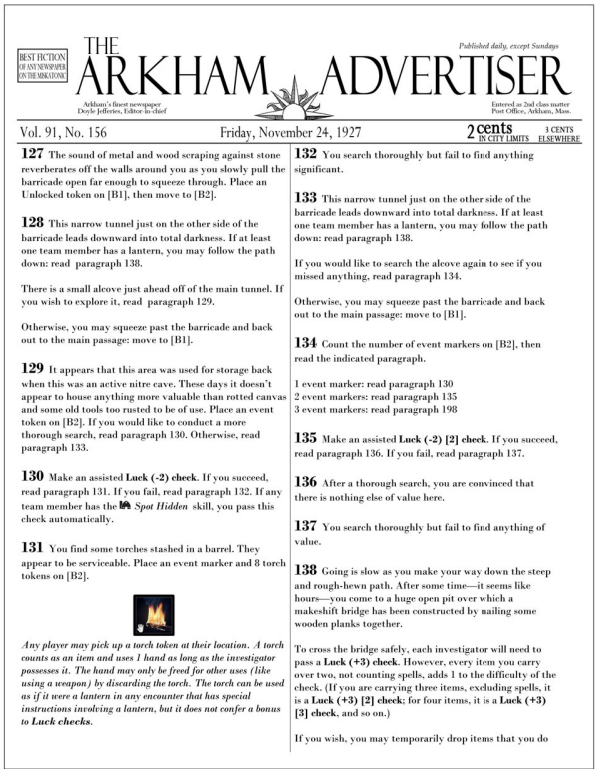
*Strange Eons* is a complex application. In order to help users grow into this complexity gradually, the interface design borrows from the the technique of *progressive disclosure* [41]—although in this case, *progressive discovery* is a more appropriate term. The stages of

(a)




(b)


(c)



BEST FICTION OF ANY NEWSPAPER ON THE NEWSTAND

THE ARKHAM ADVERTISER

Published daily, except Sundays

Arkham's finest newspaper
Doyle Jefferies, Editor-in-chief

Entered as 2nd class matter Post Office, Arkham, Mass.

Vol. 91, No. 156 Friday, November 24, 1927 2 cents IN CITY LIMITS 3 CENTS ELSEWHERE

**127** The sound of metal and wood scraping against stone reverberates off the walls around you as you slowly pull the barricade open far enough to squeeze through. Place an Unlocked token on [B1], then move to [B2].

**128** This narrow tunnel just on the other side of the barricade leads downward into total darkness. If at least one team member has a lantern, you may follow the path down: read paragraph 138.

There is a small alcove just ahead off of the main tunnel. If you wish to explore it, read paragraph 129.

Otherwise, you may squeeze past the barricade and back out to the main passage: move to [B1].

**129** It appears that this area was used for storage back when this was an active nitre cave. These days it doesn't appear to house anything more valuable than rotted canvas and some old tools too rusted to be of use. Place an event token on [B2]. If you would like to conduct a more thorough search, read paragraph 130. Otherwise, read paragraph 133.

**130** Make an assisted **Luck (-2) check**. If you succeed, read paragraph 131. If you fail, read paragraph 132. If any team member has the *Spot Hidden* skill, you pass this check automatically.

**131** You find some torches stashed in a barrel. They appear to be serviceable. Place an event marker and 8 torch tokens on [B2].

*Any player may pick up a torch token at their location. A torch counts as an item and uses 1 hand as long as the investigator possesses it. The hand may only be freed for other uses (like using a weapon) by discarding the torch. The torch can be used as if it were a lantern in any encounter that has special instructions involving a lantern, but it does not confer a bonus to* ***Luck checks****.*

**132** You search thoroughly but fail to find anything significant.

**133** This narrow tunnel just on the other side of the barricade leads downward into total darkness. If at least one team member has a lantern, you may follow the path down: read paragraph 138.

If you would like to search the alcove again to see if you missed anything, read paragraph 134.

Otherwise, you may squeeze past the barricade and back out to the main passage: move to [B1].

**134** Count the number of event markers on [B2], then read the indicated paragraph.

1 event marker: read paragraph 130
2 event markers: read paragraph 135
3 event markers: read paragraph 198

**135** Make an assisted **Luck (-2) [2] check**. If you succeed, read paragraph 136. If you fail, read paragraph 137.

**136** After a thorough search, you are convinced that there is nothing else of value here.

**137** You search thoroughly but fail to find anything of value.

**138** Going is slow as you make your way down the steep and rough-hewn path. After some time—it seems like hours—you come to a huge open pit over which a makeshift bridge has been constructed by nailing some wooden planks together.

To cross the bridge safely, each investigator will need to pass a **Luck (+3) check**. However, every item you carry over two, not counting spells, adds 1 to the difficulty of the check. (If you are carrying three items, excluding spells, it is a **Luck (+3) [2] check**; for four items, it is a **Luck (+3) [3] check**, and so on.)

If you wish, you may temporarily drop items that you do

(d)

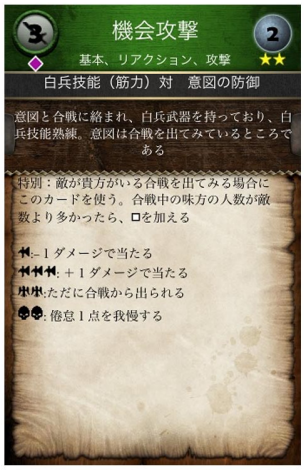



Figure 8.1: Sample components created with *Strange Eons*: (a) A game board (actual size is approximately 55 cm × 27 cm); (b) a round gate marker leading to a strange Other World in *Arkham Horror*; (c) a page from a case book [43], a type of multi-page document; (d) an "action card" for a popular role-playing game—the card's designer uses *Strange Eons* to allow him to play with Japanese friends because the game is not published in Japanese. *Notes:* (a) Board design by Max Mayer. Used with permission. (a), (b) Contain design elements from *Arkham Horror*. *Arkham Horror* is Copyright © Fantasy Flight Games. Used with permission for academic purposes. (d) Card text by Stuart Gilmour; graphic design by Ivelin Belchev. Used with permission.

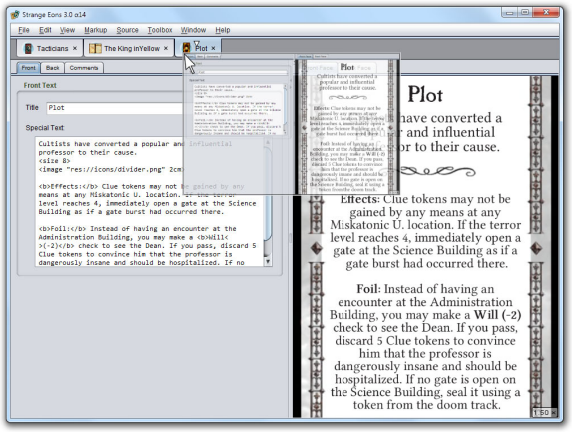



Figure 8.2: Intermediate users work with a small number of files. A simple tab strip suffices to manage work flow; in this example, a tab is being dragged to a different position on the strip.

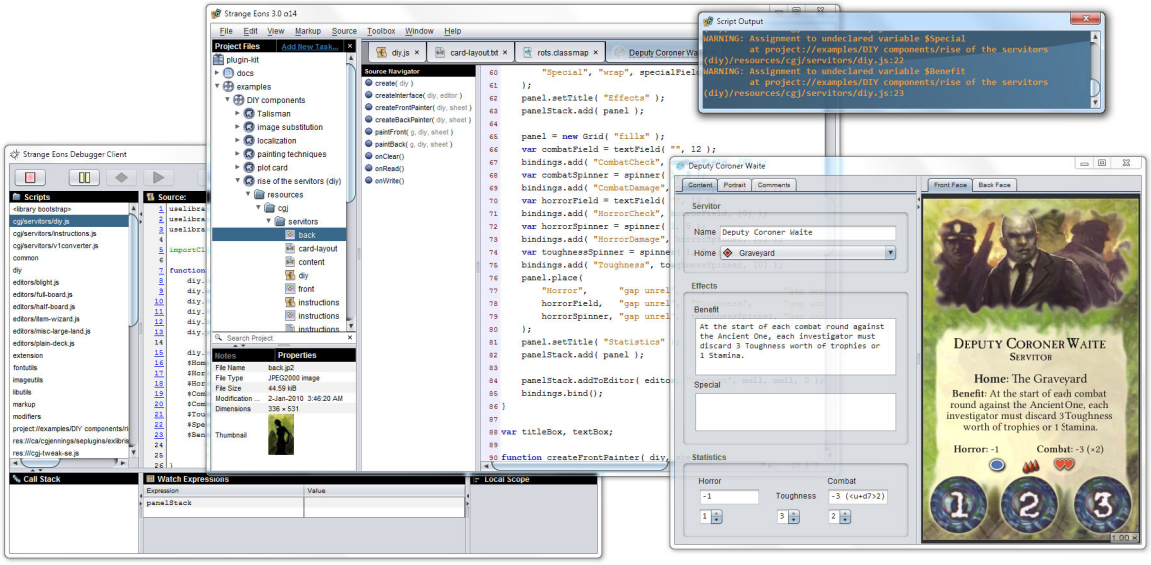



Figure 8.3: Developing a plug-in within the project system. The script for a new game component (center) is being tested. Running the script produces some console output (upper right), and an editor which has been detached from the tab strip (lower right) to compare against the script. The script debugger is ready for possible use in the background (lower left). *Note:* Some of the card graphics are adapted from *Arkham Horror*. *Arkham Horror* is Copyright © Fantasy Flight Games. Used with permission for academic purposes.

disclosure are not activated by explicit selection but arise coincidentally as the user works on larger projects and needs to manage more files simultaneously. (Each game component is stored in a separate file.)

When new users first start the program, a single dialogue is displayed that allows users to choose the kind of game component they wish to create. This gives novice users the opportunity to immediately try out one of the component editors for a few minutes and decide whether the application is likely to suit their needs.

As users become more comfortable with the application, they may eventually want to work with more than component at a time (see Figure 8.2). At this stage, the few needed documents can all be open simultaneously. The open documents are grouped together in a tab strip at the top of the application. This strip allows basic management of open files and also allows them to be "detached" into separate windows. (Detaching is useful when comparing two or more components.)

More ambitious users may eventually need to manage more files than will fit conveniently on the tab strip: for example, they may be making a large deck of cards. At this point, they are probably at least intermediate users and will be able to guess from the available commands (or will already have learned) that they should create a project. Projects are task-oriented collections of files that can be managed using a project pane on the left side of the window. (The project pane is hidden when no project is open.) Projects combine high-level file management and other generic commands with task-specific commands built around task folders.

At the deepest level of disclosure, users may decide to extend the application, perhaps to support new component types for a game that they have invented (see Figure 8.3). This requires managing a large number of disparate files: script code, configuration files, images, plug-in glue, and so on. These users will create a project containing a plug-in task folder. As plug-in task commands are performed, the application comes to resemble a modern integrated development environment, with source editing, build tools, output console, plug-in testing, full source-level debugger, and a host of other features.

### 8.3.2 Plug-in Framework

Plug-ins can be used to extend nearly every aspect of *Strange Eons*, including the addition of new commands and tools, support for new locales, shapes, images, fonts, or other resources,

and—most importantly—new games and game components. For ease of distribution, plug-ins are packaged into special archives called plug-in bundles. Plug-in bundles include support for versioning (e.g., automatic updates), dependency resolution (e.g., automatic installation of required bundles), and can be compressed using a variety of compression algorithms to reduce distribution costs.

Plug-ins are installed and updated from the application using a catalogue system. The catalogue dialogue downloads and displays a list of available plug-ins with descriptions; the user may then choose plug-ins to be downloaded and installed. Installed plug-ins are dynamically loaded at runtime, giving them full access to the *Java* classes and other resources that make up the application.[1]

Plug-ins may be based on compiled *Java* [89] code, or more commonly, *ECMAScript* [62] scripts (specifically, *JavaScript 1.7* [62, 189, 190]). Although *JavaScript* has a number of well-documented design flaws [49], it was chosen for several reasons. First, a good implementation, *Mozilla Rhino* [26], was readily available. Second, due to its prevalence in Web development, nonspecialists are more likely to have previous experience with it than with alternatives. Third, for the same reason, any number of online tutorials are available to help beginners learn the language. Fourth, it has excellent interoperability with *Java* (for example, scripts can dynamically subclass *Java* objects). And finally, it has features that dovetail with the design of the plug-in architecture; for example, its support for *closures* [144] provides a compact way to implement the event listeners and other interfaces needed to extend the application.

### 8.3.3 Game Component Framework

Game components (files) generally represent a single playing piece, such as a card, sheet, or token. Most game components have a fixed size and are rectangular in shape, although this is not required. For example, the token editor can create game tokens in a variety of shapes and sizes.

User interaction with game components is usually through an editor window, which groups the widgets needed to edit the component together on the left side and provides a live, zoomable preview of the component on the right. The different surfaces of a component (such as the front and back faces of a card) can be flipped through using small tab buttons.

[1] In *Java* parlance, the plug-in files are added to the classpath using a custom classloader.

The game component framework provides a host of functionality that custom components take advantage of automatically, such as serialization, printing, exporting as images, and support for publishing industry features such as fold marks, crop marks, and bleed margins. It also provides a standard *Spin Off* command that explicitly creates an alternative as a new document (without the interruption of choosing a new save file).

For developers, the framework's features include automatic management of rendering resolution (component drawing code is independent of resolution), caching mechanisms, simplified handling of user-supplied images (called *portraits*), and automated binding of widget events to variables in a game component.

### 8.3.4 Text Layout and Graphical Effects

Dynamic games are usually text-heavy, so *Strange Eons* includes a powerful text layout engine that can render formatted text from HTML-like markup text. The markup language is based on HTML in order to leverage any previous experience the user might have.

The layout engine supports standard text styling (font family, point size, weight, width, posture, tracking, underline, text colour, and so on), justification and alignment, ligature glyphs, inline graphics, bidirectional text, and other modern digital typesetting features. Of greater interest, though, are features aimed specifically at game component design:

**Text Fitting** Game designers must often squeeze complex descriptions onto tiny cards, so they learn to write economically. This is difficult for amateurs, so the layout engine can instead fit the available text into the desired space by automatically shrinking the line spacing and text size. (See Figure 8.4.)

**Text Shaping** Many components have complex layouts that make use of decorations, borders, icons, or other features that the text must wrap around. The layout engine can adjust the margins of each line to match arbitrary curves and shapes. (See Figure 8.5.)

**Variable and Conditional Text** The details of a game component are often revised many times before they are finalized. The markup parser supports variable tags that are dynamically replaced with information taken from the component. For example, the tag `<name>` inserts the component's name or title. (While some variable tags are predefined, end users can also define new variable tags using a macro facility.) The parser also supports conditional text composed of two or more alternatives, one of


Figure 8.4: Text fitting accommodates unusually long rules with ease. As the text gets longer (from left to right), the layout engine will first reduce line spacing and then scale the text size down. When the scale factor becomes extreme, a red warning box is displayed.

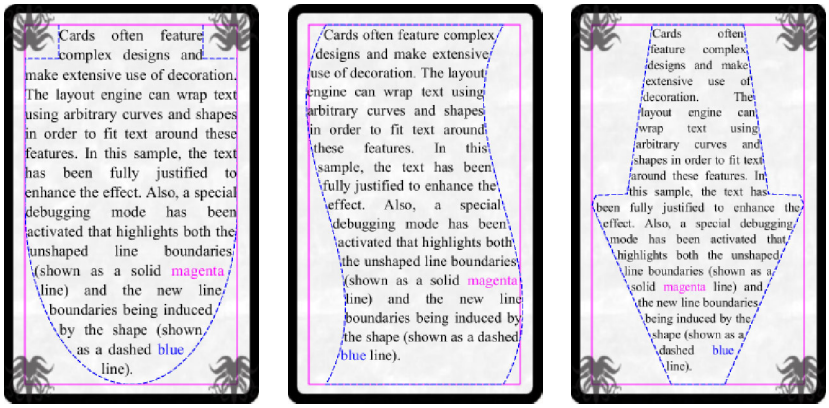



Figure 8.5: Text shaping can wrap text to fit arbitrary paths. This is necessary to match the complex overlapping arrangements of text and graphics used by some component designs.

which is selected based on the component state. The linked state can be anything, but is usually gender. For example, a game character might be given a special power called `Manipulat<or/rix>`, which would appear on the component as either *Manipulator* or *Manipulatrix*, changing automatically as the character's gender is altered.

In addition to text layout, *Strange Eons* also supports various graphical effects and filters commonly used in game component design. One such effect, *tinting*, specifically supports custom content creation (see Figure 8.6). Tinting allows the user to create simple variations of a standard component by dynamically recolouring selected parts of the component's graphics. The user simply chooses a target colour and the component is recoloured to match. Using tinting, it is easy to create a line of custom components that is both distinct from but clearly in the same design family as other components from the game.

### 8.3.5 Deck and Board Editor

To facilitate printing on home printers, *Strange Eons* provides the deck editor for laying out groups of objects together on a page. This editor has some similarity to vector-based diagramming software, but it is specialized for the tasks of laying out sets of cards and creating game boards.

To simplify card layout and board design, the editor makes extensive use of *snapping*, which allows objects to be quickly placed into precise alignment using imprecise drag-and-drop gestures (see Figure 8.7). Class relationships determine whether and how two objects snap together. For example, card faces snap to each other and to grid lines on the page, but not to any of the objects that are used to design game boards. (The user can change an object's default snapping behaviour and override snapping altogether if desired.)

To assist the user in assembling the printed output, publisher's marks are automatically generated around compatible objects (also shown in Figure 8.7). The type of mark depends on how the object is related to any neighbouring objects that it contacts. For example, if two different cards are placed next to each other, they will be surrounded by crop marks (indicating where the cards should be cut). However, if the front and back face of the same card are placed next to each other, then the mark along that edge will be converted into a fold mark if and only if folding along that edge would produce a correctly printed card. To avoid confusion, marks that overlap with other objects are suppressed.

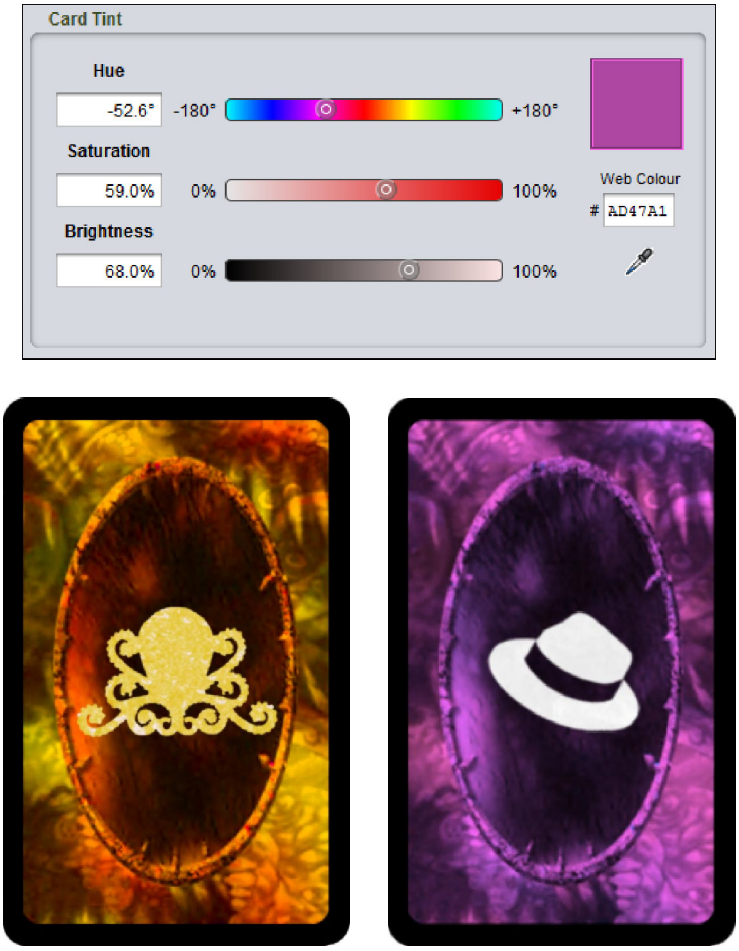



Figure 8.6: Tinting makes it easy to create variants of existing components. In this example, the user creates a new kind of card by choosing a different tint colour (top) and centre graphic for an existing card back (bottom left). The result is a new card back design (bottom right) that is visually distinct from but clearly related to the original design.


Figure 8.7: Laying out a deck of cards for printing. To create a deck, the user first chooses the game component files to include, which are added to a list. Individual component faces are added to the deck by dragging them from this list (not shown) to the deck page. Crop marks (solid lines) and fold marks (dashed lines) are placed around the card automatically. (The marks have been enlarged for clarity in this illustration.) Crop marks become fold marks only when touching faces are compatible: the two faces in the upper left corner have fold marks because they are the front and back face of the same card; the cards below do not because they are both front faces. On the far right, a new face is being dragged into position. The blue highlight indicates that the new face will snap to the right of the highlighted face if dropped at its current location. A target face is chosen by dragging the new face anywhere overtop of it; the location to snap the new card to is determined from the relative positions of the centres of the faces. Snapping allows precise alignments to be determined from coarse, rapid drag gestures: laying out a deck would be much slower without this feature. *Note:* The Exhibit Item card design is from *Arkham Horror*. *Arkham Horror* is Copyright © Fantasy Flight Games. Used with permission for academic purposes.

## 8.4 A Specific Game: Arkham Horror

Since *Strange Eons* creates components for specific games—treating each component as a separate design space—it can provide a second layer of support specifically tailored to each space. The evaluation of *Strange Eons* as a (partial) implementation of the expert support design principles is based thus on the support provided for a particular game, *Arkham Horror* [148], and a specific component for that game, investigator cards.

*Arkham Horror* is a cooperative board game based on the work of H. P. Lovecraft [168], an early 20th century author of speculative fiction. Lovecraft's historical importance is due less to the quality of his writing (which is notoriously prolix) than to the influence of his ideas on other authors. Lovecraft corresponded with a large circle of literary colleagues. These writers often borrowed settings, props, plots, names, and other elements from Lovecraft's stories, incorporating them into their own work. Through the years other writers picked up and continued this practice.[2] The end result of decades of filtering Lovecraft's ideas through the sieves of other minds is a loose accumulation of lore called the *Cthulhu Mythos* [103]. This background is a rich source of material for end-user development.

*Arkham Horror* draws on elements from this larger mythology, but remains closely aligned with several of Lovecraft's key themes, especially dystheism, the insignificance of humanity in relation to the universe (cosmicism), and the dangers of forbidden knowledge. During the game, players assume the role of amateur investigators exploring strange goings-on in the fictional 1920s town of Arkham, Massachusetts. The goal of the game [8] is to prevent a dangerous cult from awakening a long-dormant *ancient one*, one of several god-like entities whose revival would bring destruction.

The investigators explore the town, encountering various people and situations. The encounters may require players to decide between alternative courses of action, and often involve a *skill check*, which uses dice to determine whether an attempted action succeeds. The probability of success is determined by the difficulty of the action and by the investigator's rank in the relevant skill.

The resolution of an encounter may include a gain or loss of resources: the investigator might find an item stuffed in a tree trunk, or win some money in a pool hall bet; or they

[2]Some authors go so far as to write whole new Lovecraft tales, often early in their careers. Stephen King's epistolary short "Jerusalem's Lot" [134, pp. 1–35] is a good example; it reads like Lovecraft and Bram Stoker plotted the story, and King added the characterization and dialogue.

may end up in a fight in a seedy speakeasy and have to discard some of their Stamina tokens (running out of Stamina tokens means a trip to the hospital). For example, an encounter at the Bank of Arkham might read:

> Wealthy landowner Alijah Billington has dropped a scrap of paper ahead of you in line. You surreptitiously pick it up. Make a **Luck (+0) check**. If you succeed, you notice strange writing on the back. Gain 1 Clue token. If you fail, you hand back what appears to be a grocery list and he gives you his thanks.

This encounter gives the investigator the chance to gain a Clue token, one of the most important resources that players can obtain. Clue tokens represent knowledge that can be used to foil the growing threat. As the cultists pursue their plan, gates to strange other worlds begin appearing in town, representing the thinning barrier between this reality and the ancient one's. Investigators may enter these gates, explore the other side, and—if they make it back—use the knowledge gained from Clue tokens to seal the gate. By sealing enough gates, the investigators can thwart the cultists and win the game, but if they act too slowly, the ancient one will pierce the barrier between worlds and the players will lose.

A number of obstacles slow the investigators' progress, but the main obstacle is the use of two timing mechanisms that limit the number of turns available to stop the cult. The first of these mechanisms is the doom track, which counts up as new gates appear and represents the ancient one's waxing power. When this track fills, the ancient one awakens. The second is the terror track, which represents the breakdown of society as the town descends into chaos. The terror track is not tied directly to the cult's progress, but responds to indirect evidence of their activity; for example, if news of a grisly murder is published in the local paper or the investigators fail to control the spread of a rumour. As the terror track rises, resources become scarce: potential allies flee town and store owners board up their shops.

A more active obstacle occurs in the form of roaming monsters. As the town is overrun with gates, denizens from the other side find their way to our world. As the monsters roam the streets, they hamper movement about town: to move through an occupied space, investigators must either sneak past or fight. Either choice is risky. These otherworldly creatures are not only physically dangerous; they are so alien that an investigator's Sanity is threatened just by knowledge of their existence.

Other events, unrelated to the cult's activities, also delay progress towards the goal. At the end of each turn, a new card is drawn that may represent newspaper reports of strange

happenings, a change to bad weather, or other effects. Each of these introduces temporary rules that hamper the player's ability to plan.

## 8.5 The Design Space of Investigators

Each player draws a random investigator at the start of the game. Investigators are printed on large cards that list their starting attributes (see Figure 8.8). At the top of each card is a portrait, the investigator's name, and their occupation. Beneath this are found:

**Sanity and Stamina** These are resources than can be lost as a result of mental or physical harm; if either runs out the player is penalized.

**Home** This is where the investigator begins the game; it is tied to the investigator's back story (see below). The home is usually chosen so that the investigator is initially safe.

**Fixed Possessions** Resources that the investigator always starts the game with, such as items and money.

**Random Possessions** General types of resources from which the player draws randomly at the start of game. These keep play fresh by ensuring different starting resources for each game.

**Special Ability** A rule unique to the investigator, tied to the back story or occupation.

**Skills** A set of six skills (Speed, Sneak, Fight, Will, Lore, Luck) that define the investigator's strengths and weaknesses. In addition to making skill checks as described earlier, most skills also serve other roles. For example, an investigator's speed determines how many spaces it can move on the board each turn. Each skill has one from a range of four possible values at any given moment. At the start of each turn, the player is allowed to adjust skills a step at a time within this range. The skills are paired such that increasing the value for one skill decreases its mate. For example, if the player increases their Speed by one step, then their Sneak is decreased by one step. Predicting how to best distribute skills is an important tactical element of the game.

**Focus** This value limits the total number of steps that the investigator's skill pairs can be adjusted per turn.

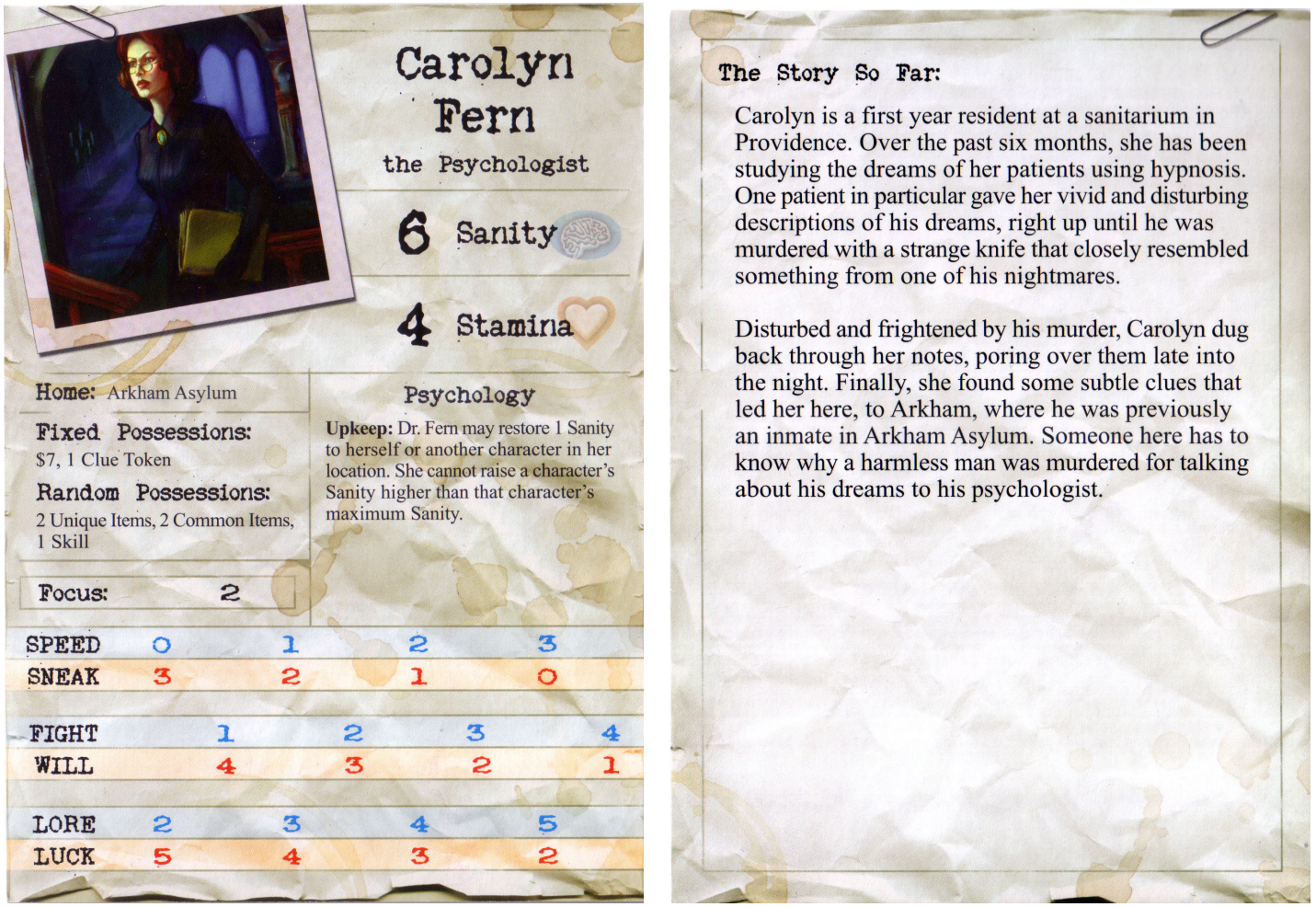



Figure 8.8: Carolyn Fern, one of the investigators included with *Arkham Horror*. Shown are the front and back faces of the card (left and right, respectively). On the front face, in clockwise order from the upper left: the portrait, name and occupation, maximum Sanity and Stamina scores, special ability, skill pairs, Focus, starting equipment, and Home. The back face shows the investigator's back story. *Note: Arkham Horror* is Copyright © Fantasy Flight Games. Used with permission for academic purposes.

In addition to these attributes, the back face of each card provides a back story to assist with playing the role of the investigator. A good back story gives coherence to the entire character by implicitly explaining the various attributes and the investigator's motivations.

## 8.6 The Investigator Editor

The layout of the investigator editor in *Strange Eons* roughly corresponds to that of investigator cards, but some sections are located on separate tabs due to the limits of window real estate (see Figure 8.9). In addition to the attributes described in the previous section, the editor has an additional tab labelled Comments. This is a standard feature of every editor, and serves the same purpose as design rationale annotations in *XDS* (see p. 65).

Several features of the investigator editor help designers who reach an impasse. In addition to entering a name, a random name can be generated based on the selected gender. The names (both given and family) are selected from tables of the most common names for babies born in the years 1880-1900 (i.e., common names for adults in the 1920's; the tables were derived from census data). The title (occupation) field is an editable list (combo box) that allows designers to enter their own value or choose from over 300 built-in options. This list includes not only typical occupations of the time, but also more fantastic options that suit the game's theme, such as "the Rum Runner", "the Antiquarian", "the Exobiologist", or "the Vigilante". The tab that allows the user to edit the investigator's special ability includes presets for all of the special abilities that have been used by official investigators as well as a handful of new examples. This allows designers to compare an idea with what is already available to consider its balance and to verify its novelty. Novice or casual designers can also simply choose the standard ability that best matches the character that they have in mind without spending the additional time needed to develop a novel rule. (For example, a number of people have used *Strange Eons* to create investigator interpretations of their play group or of favourite book or movie characters for one-off use.)

The main support feature for investigators is its consequence display. The display shows sentences to indicate that the design has crossed various formal and informal boundaries in the design space. The boundaries are a combination of conventions established by the official published investigators and additional heuristics. They have been chosen to help the designer to know when the design may by over- or under-powered compared to the official designs. Since the game is cooperative, the investigators do not need to be completely fair,

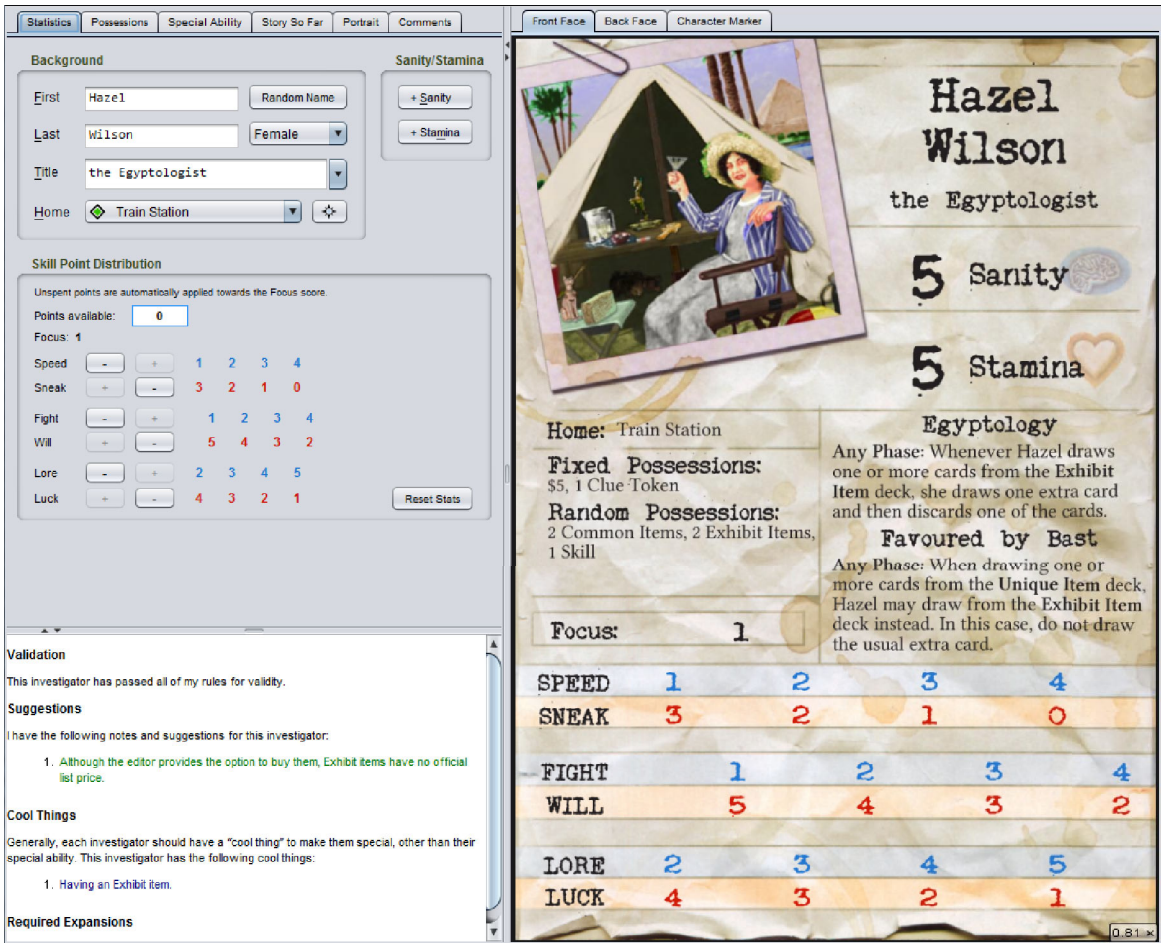



Figure 8.9: The investigator editor. The upper left part of the window contains the widgets used to edit the card. Due to the number of widgets, some are not shown but are found on other tabs (the visible tab allows editing of the name, occupation, and basic numerical attributes). Across from the editor is the zoomable preview window, currently showing the card's front face. The consequence display in the lower left corner shows how design consequences are affected by edits. *Note:* Investigator card design is from *Arkham Horror*. *Arkham Horror* is Copyright © Fantasy Flight Games. Used with permission for academic purposes.

but if there is too much disparity then some players are likely to be relegated to second-tier roles. The consequences also inform the user when the design places much more emphasis on one design consequence and ignore others. For example, if the designer selects many more fixed possessions than random ones, the design probably favours characterization over replayability.

While the consequence displays in *XDS* provided graphical visualizations of important consequences of the design that can be computed or estimated quantitatively, the consequences monitored here are essentially boolean: either a boundary is followed or it is not. And while *XDS* focused on four key consequences of experimental design, in this case there are dozens of guidelines to consider: making the user aware of these consequences requires a distinct approach from that of *XDS*.

The investigator editor displays consequences as a brief report with four possible sections: Validation, Suggestions, Cool Things, and Required Expansions. The Validation section is reserved for a few specific cases where breaking a guideline could have especially egregious effects on the investigator's fairness. Being "invalid" does not prevent the investigator from being used. It does indicate, though, that unusual effort may be required to balance the investigator out.

The Suggestions section, which might more accurately have been called Observations, provides feedback on the user's location in the design space and also points out possible complications and interactions that the designer might not be aware of. For example, the highest value of a skill normally falls between 3 and 6 (inclusive). Having a 6 in a skill is rare—it is listed as a Cool Thing (see below); none of the official investigators have more than one 6. This does not mean that a design with a double 6 is wrong, but it does mean that the designer is venturing into an untested part of the design space. The editor does not discourage the decision—the fact that it has never been done may be exactly why the designer decided to do it—but the editor does point out that there may be unintended consequences: "A skill of 6 or more is rare. This investigator has 2 of them. The investigator may be too weak in other areas."

The entires in the Suggestions section generally fall into one of the following categories:

- The design is outside of the usual upper bounds for investigators (e.g., spent extra money while shopping, chose unusual skill values).
- The design is outside of the usual lower bounds for investigators (e.g., has unspent

money or skill points).

- The design may have unintended consequences: one aspect of the design may interact with another part of the game in a way that makes the game significantly harder or easier.
- A design choice related to one part of the investigator card will have a side effect (not necessarily good or bad) somewhere else on the card.
- An aspect of the design requires support with a special rule (the design creates a situation that is not consistent with the game's usual rules).
- The design is not as cohesive as it could be; for example, the special ability requires a Clue token to work but the investigator does not start with any.
- An aspect of the design may not make thematic sense unless it is explained by the investigator's occupation, back story, or other characterization. (For example, giving an investigator a random Ally, as opposed to a specific one, may seem out of place unless the investigator's story explains why they would have such varied associates.)
- The design uses an "unofficial" design feature that, while supported and considered balanced by *Strange Eons*, is not part of the design space defined by official components. That is, the feature is a transformation of the design space (as established by official published products) that has been introduced by *Strange Eons*. This list has changed over time because the game designers began using *Strange Eons* to help design expansion products—some unofficial features subsequently became official, while at the same time new unofficial features were introduced.
- The investigator has been given an especially powerful item; usually, players are more engaged when such items are reserved for use as rewards [115].

The Cool Things section notes aspects of the investigator that help to differentiate it. While every investigator has a different special ability, all of the official investigators also have a special quality that is not explicitly stated on their card. The game's designers refer to these as "cool things" (Kevin Wilson, personal communication, July 24, 2009). Cool things are generally not special by themselves, but are special in relation to other investigators. For example, only one of the investigators is rich (the only one to start with

$10), and only one has a pet. These are both cool things. Cool things are not always related to the investigator's starting equipment; any rare or unique aspect of the design can qualify. For example, while most investigators have a special ability that only benefits themselves, Carolyn Fern's can be used to help her teammates (see Figure 8.8).

Finally, the Required Expansions section tells the designer if they have chosen an option that requires a particular expansion product. This prevents novices who are unfamiliar with all of the game's components from creating an investigator that would require components that they don't actually own. This is easier to do than one might expect: the base game has over 700 components, while the expansions published to date add nearly 2000 more.

Reading a long list of consequence sentences after each edit would soon become tedious. To help designers focus on how the consequences *change* as the investigator is edited, the list entries are formatted differently when they are new (shown in bold) and when rendered inapplicable by the most recent edit (shown in strikethrough text). Suppose that the designer added a new item to an investigator's list of possessions, and *Strange Eons* noted that this new item is synergistic with another (already selected) starting item in a way that could be unfair. A new consequence would be added to the list to note this, and because it was new it would appear in bold type. If the designer then performed some edit other than removing one of the two conflicting items, then the consequence would be displayed in regular type (because it is still present but no longer new). If one of the items in the synergistic pair was removed, then the consequence would be displayed in strikethrough type (indicating it has been "deleted"); after the next edit it would be removed from the list.

To help designers visually search the list, items in the different sections are listed in different colours. When an edit causes the list to grow or shrink in length, possibly changing the position of a consequence that the designer is trying to influence, these colours help the user to rapidly reacquire the target consequence and check its new status.

The consequence display should help designers create investigators that are explore novel areas of the design space without unbalancing play. The entries help the user to compare their design with established examples, and to recognize if an aspect of the design is so far from established norms that it may require compensation elsewhere. Since the consequences are verbal, it is important to use neutral wording. The consequence display is not meant to be a tutoring system that prefers one alternative over another. The following sentence captures the intended tone: "I realize that you know more about this than I do, but have you considered $X$?"

The consequences provided by the investigator editor are distinctly weaker than those in *XDS*. The consequences in *XDS* are always available, and always have some relevance to the problem. They are closely related to important hypotheses that the experiment designer needs to be able test in order to judge how well a possible solution fits the problem. The consequence display in the investigator editor is backed by a simple expert system. If nothing about the design fits part of the editor's predetermined model, then it will have nothing to say. The information it presents mainly provides a sense of the location of the solution in the solution space; this has only an indirect relationship to how well the solution matches the problem. It therefore requires more interpretation on the part of the designer, whereas *XDS* allows the direct visual comparison of designs.

# Chapter 9

# An Evaluation of *Strange Eons*

## 9.1 *Strange Eons* in Light of the Principles

Describing the principles supported by *Strange Eons* is complicated by the layered organization and the fact that the specific editor used for evaluation does not use all of the support available in the lower layer. The summary in Table 9.1 describes the specific degrees of support provided by the investigator editor, since that is the target of evaluation. Specific games or components other than the investigator editor may have their own custom support; this is not considered here.

### 9.1.1 Make Partial Solutions First-class Entities (Low)

The use of variable and conditional text (symbolic tags) allows the designer to insert placeholders for parts of the design that are incomplete or subject to change. In the investigator editor, the designer can use special tags to insert the investigator's current first, last, or full name (the first name tag parses the first name field to remove middle names and initials), the investigator's home location (with an appropriate article, if desired—so *the Witch House* but *Miskatonic University*), conditional tags tied to gender (see page 117), and the user-defined macros that are available for all mark-up text. (Figure 9.1, which shows the back story of the investigator in Figure 8.9, shows an example of these tags in use.) Using these tags, the designer can develop special ability rules and back stories without committing to specific details in other parts of the design.

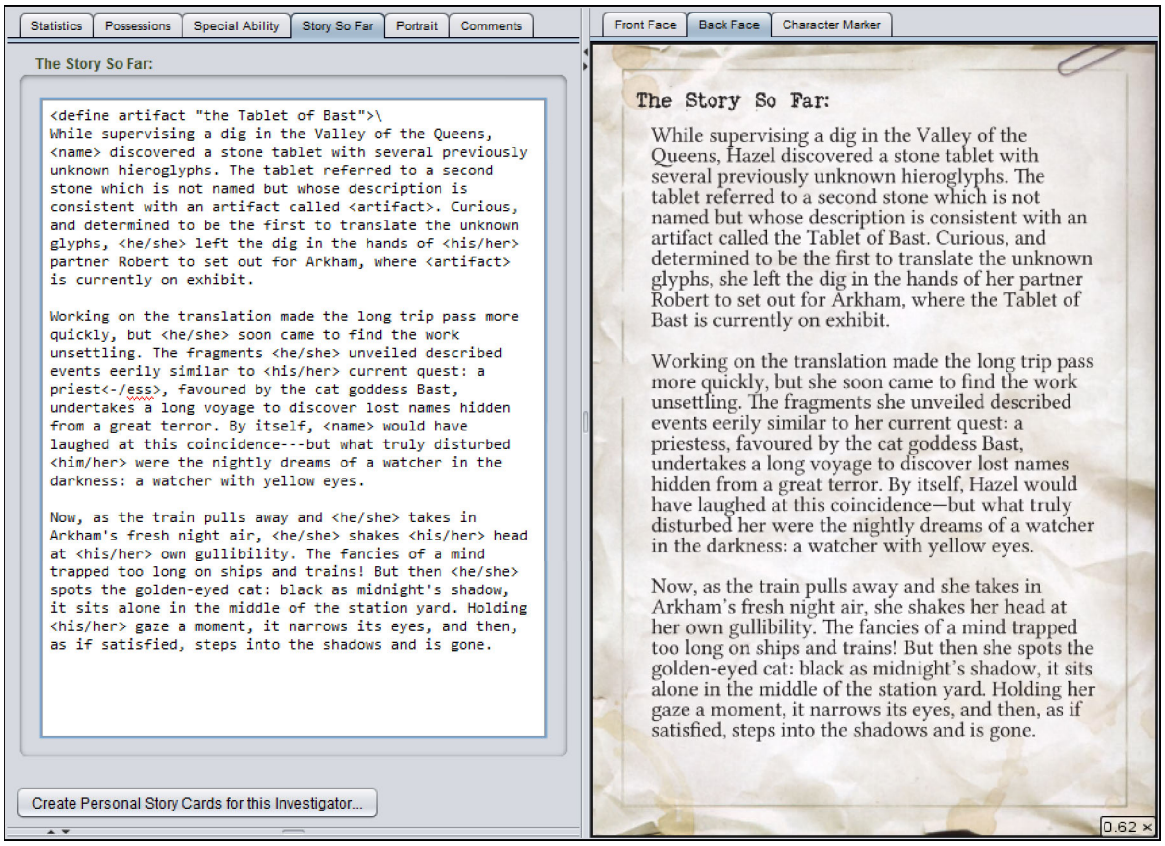



Figure 9.1: A back story (right) that makes extensive use of symbolic tags. The markup text (left) uses variable tags, conditional tags, and a simple macro to keep the description neutral with respect to the investigator's name, gender, and the name of the artifact that motivates travelling to Arkham. Because this travel is central to the story, the designer chose not to refer to the investigator's home (the train station) using the symbolic `<the home>` tag. *Note:* Investigator card design is from *Arkham Horror*. *Arkham Horror* is Copyright © Fantasy Flight Games. Used with permission for academic purposes.

Table 9.1: Degree of Support for the Design Principles in the *Strange Eons* Investigator Editor

| | Design Principle | Support |
|---|---|---|
| 1. | Make Partial Solutions First-class Entities | low |
| 2. | Support Problem and Solution Matching | moderate |
| 3. | Allow Subjectivity and Ambiguity | none |
| 4. | Prefer General, Flexible Actions and Representations | moderate |
| 5. | Engage Multiple Ways of Doing and Thinking | moderate |
| 6. | Support Forming and Testing Hypotheses | moderate |
| 7. | Encourage Parallel Exploration of Breadth and Depth | none |
| 8. | Provide Rich History Mechanisms | none |
| 9. | Assist the Construction and Use of Repertoires | moderate |
| 10. | Create an Effective Environment | none |

Details like the investigator's name are simple examples of cross-cutting concerns [132]: elements of the design that must be weaved into other concerns (such as the wording of rules). The provision of special markup tags allows the designer to work in terms of generic partial solutions whose reification (the weaving) is partially automated. Unfinalized details do not need to be updated manually at each point where they occur, a task both time consuming and prone to error.

### 9.1.2 Support Problem and Solution Matching (Moderate)

The investigator designer has a characterization in mind and is trying to match this to a combination of attributes that expresses that character while remaining fair and fun to play. The consequence display helps the designer understand their location in the design space relative to the limits of exemplar designs. By giving the designer a better sense of the play balance of an alternative, he or she can focus on discovering different ways to express the character. This is particularly helpful when the designer unknowingly strays into unexplored territory, as adding a novel feature to one part of the design may create unintended effects elsewhere. At the same time, the consequence display uses neutral wording to avoid scaring the user away from new territory altogether. Game design is a flexible domain that allows a lot of room for creative solutions. Although the display prompts the user with information about potential issues, it does not insist upon resolving them, or how.

### 9.1.3 Prefer General, Flexible Actions and Representations (Moderate)

End-user game content development is a large space, but it consists of many pocket design spaces for particular games. A system that tried to support the entire space with a single set of tools would amount to an illustration and page design package. Instead, *Strange Eons* uses a layered approach: providing a core of features that can be used for many games and the infrastructure needed to easily build more specific support on top. A specific aspect of this support that also fits under this principle is tinting, which allows a single card representation to be reinterpreted to fit a variety of design situations.

### 9.1.4 Engage Multiple Ways of Doing and Thinking (Moderate)

Support for end-user development, automation, and data import/export is extensive. Intra-application support is minor: the editors are designed to avoid staged disclosure (in which a task's options are broken down into multiple pages, each of which is completed in sequence) so that the different parts of a component can be edited in any order. While an investigator's possessions are normally edited using a shopping metaphor (where the desired items are selected from lists and the cost of the items is tracked automatically), there is also an option to enter this as free-form text.

### 9.1.5 Support Forming and Testing Hypotheses (Moderate)

Although consequence displays support the forming and testing of hypotheses, they do not do so equally well. The consequences in *XDS* are always present and always provide the same kinds of information. The consequences in *Strange Eons* are less predictable, as the designer does not know in advance what information will be available. They are also less reliable, because they are triggered by specific conditions and there may be gaps in this database. In *XDS* the user can make apples-to-apples comparisons when comparing designs, while in *Strange Eons*, comparisons are more subjective because entirely different kinds of consequences may be shown for two designs.

### 9.1.6 Assist the Construction and Use of Repertoires (Moderate)

*Strange Eons* does not support the explicit construction of repertoires, but its project and deck editing features do help to organize and collect game components created with the program. The name and occupation databases also serve some of the functions of a repertoire

for a few crucial aspects of an investigator design. The occupation in particular plays an important role in helping the designer to shape an investigator's attributes.

Overall, the design of *Strange Eons* reflects what one might develop by extending an existing application to incorporate some of the principles without the cost of a radical redesign. The application is still structured around editing a single, concrete alternative at a time—for example, there is no explicit history mechanism, not even an undo command. The small degree of support for partial solutions mitigates this slightly by making parts of the design more generic. And while the application provides little support for managing alternatives, the consequence display does help users to understand and compare them. The rest of this chapter will evaluate the effectiveness of this support, particularly that provided by the consequence display.

## 9.2 Two Studies

I conducted a pair of studies in parallel to asses the effects of using *Strange Eons* when designing investigator cards for *Arkham Horror*. These consisted of an observational study and a controlled experiment. There were two overall goals: to verify that the principles (specifically, those tied to consequence displays) are effective in more than one problem domain, and to gather more general information about the kinds of effects that consequence displays might have on design. To this end, the following hypotheses were tested:

1. Designs will be ranked higher in overall quality when consequences are available.

2. Designs will be completed more quickly when the consequence display is available.

3. Participants will execute more design moves when the consequence display is available.

Both studies compared investigators created using *Strange Eons* to investigators created without this support. The observational study used investigators uploaded to public Web sites, while the controlled experiment recruited participants to create investigators specifically for the experiment. Due to the overlap in the kinds of data that were gathered, the results from both studies will be presented together in a single section.

### 9.2.1 Methods for Controlled Experiment

A total of 17 participants took part in the controlled experiment. Participants were recruited by posting a brief description of the study to online forums that discussed *Arkham Horror*. The post included a link to further information that obtained informed consent and an email address used for further communication.

Before starting, participants were given a brief questionnaire intended to stratify participants according to their experience with *Arkham Horror*, game design, and related applications (such as drawing software). However, stratification was not practical due to the small number of participants and the range of responses, so participants were randomly assigned to either the control group (no consequences) or the treatment group (consequences). Nine participants were initially assigned to the no consequences group while eight were assigned to the consequences group. One participant later withdrew, leaving eight participants in each group.

Once a participant was assigned to a group, he or she was sent an email message containing a unique identifier and a link to download one of two versions of the software (depending on the assigned group). After this information was sent, only the unique identifier was used to refer to participants.

Participants downloaded the software at their leisure, but the time of each download was recorded to determine time to completion. Participants were given up to two weeks to complete an investigator design and return it. The modified copies of the application were designed to expire after the end of the experiment to protect participant privacy, since their save files were modified to store edit histories and other information.

Participants were asked to enter their unique identifier on first running the application. (The identifier consisted of three randomly assigned characters plus a checksum character for validation.) Once entered, this code was stored for future runs.

Investigators were returned by choosing a special menu command that uploaded the finished design to a secure server, tagged with the participant's unique identifier. Participants in the treatment group ("Consequences") then completed a brief questionnaire of reflective questions about their experience with the consequence display. Questionnaire results were likewise uploaded to the server. Participation ended at this point.

To avoid bias, the portait images for all designs were deleted: portrait images vary widely in quality since some designers create their own while others appropriate images

produced by professional artists. The designs from both the controlled experiment and the observational study (see below) were then combined for blind evaluation by four expert reviewers. The reviewers were recruited from the online community of *Arkham Horror* custom content developers by contacting prolific investigator designers with a reputation for good designs. Due to the work involved, each investigator was judged by only two of the four reviewers. Investigators were assigned to reviewers at random, but the pairings of reviewers were restricted to distribute them as evenly as possible. (So, if reviewers 1 and 2 were paired together to evaluate the first design, they were not paired again until every other possible pairing had also appeared exactly once.)

Each investigator was scored using a double-sided sheet that included 22 categories; each category was scored using a 10-point rating scale (with 10 always indicating the best score).[1] Investigators were scored according to the general criteria of balance (power relative to official investigators), characterization (how well the various attributes represent the character suggested by the name, occupation, and back story), creativity (meaning novel but playable ideas), replayability (as investigators are generally played more than once), theme (how well the investigator fits the game's setting), as well as overall impression. The individual categories were chosen to ensure that the reviewer would consider each element of the investigator sheet in turn before finally grading the complete design for the above criteria. However, the purpose of this was to ensure that each investigator got similar consideration: only the overall score for the complete design was used to compare groups.

Because of the possibility of biasing judgements to favour those factors covered by the consequence displays, the reviewers were not trained for inter-rater reliability. However, they were given practice investigators in order to become familiar with the scoring process and to establish individual baselines.

While all of the investigators from both studies were evaluated by the expert reviewers, some additional data was collected from the controlled experiment. The turnaround time for each participant was measured. This was the duration from the time that they downloaded the modified application to the time that the finished design was submitted. (If a participant downloaded the application more than once, this was assumed to be due to a network error

[1] A 10-point scale was chosen in order to allow the reviewers to make fine distinctions if they felt the need, and also because many popular kinds of judging contests use a similar scale and it was felt that this familiarity might make rating easier. Ratings that use a 5-point or 7-point scale have been found to result in slightly higher mean scores (relative to the maximum score) than a 10-point scale, but exhibit no other statistically significant differences in their descriptive statistics [56].

and the most recent time was used.) To allow for the vagaries of sleep, work, school, and other interruptions, turnaround times were rounded to the nearest number of days. (The range of return times was 5 days, with a median of 3 days.)

The modified application recorded the edit commands issued by each participant. I subsequently analyzed these transcripts to infer the number of design moves that were performed. The following procedure, refined by analyzing several test investigators created by the author, was used to count moves:

> A *design move* is inferred when the design changes from one state to another, and the new state is *constructive*. To be constructive, the new state must result in a design at least as complete as the previous state. For example, if the user had a blank design and then entered a name, then this is constructive because it moves the design closer to being a complete solution even though it is not a complete solution yet. If the name was then deleted again, this would not be constructive because it would make the design less complete than it was previously.
>
> When an attribute is defined verbally, moves are judged by comparing the semantics of the content from the time that the text field gains focus (becomes active) until it loses focus (another control is activated). If the semantics change, then a new move may be counted. For special abilities, each separate rule is counted as if it appeared in its own text field. Design rationale comments, along with spelling and grammatical errors or corrections to same, are ignored.
>
> Many numeric attributes are adjusted using step-wise controls. For example, to increase an investigator's Speed from 3 to 5, the skill must be increased twice. When such an attribute is adjusted monotonically and no other control is used between steps, this is counted as at most one move. It is assumed that the intent was to set the attribute to the final value and that the intermediate values were incidental.

Counting moves rather than edits yields a metric that better reflects the number of explored alternatives. Like any attempt to infer intent from command sequences, it is not exact. For example, it will count the initial moves that are required to set up the first alternative as a sequence of alternatives (some of which may not be intended as such). Since solutions are often partial during divergence and transformation, the designer may switch

their goal to a second alternative before completing the first, and this may not be detectable from the edit history. (Indeed, some alternatives may be considered and rejected without the designer ever touching an input device.) There are also cases where this metric may miss alternatives: for example, the designer may sometimes step through a numeric attribute in order to reflect on each value in turn. These flaws notwithstanding, experience with the test analyses indicates that it is a reasonable approximation. It is certainly more accurate than a simple edit count, which grossly overestimates alternatives during text entry.

### 9.2.2 Methods for Observational Study

For the observational study, a total of 34 investigator designs were sampled to obtain two groups of 12 (one group of investigators created with *Strange Eons*, and one group created without it). The designs were collected from public Web sites and chosen from examples posted during the same six month period shortly after *Strange Eons* was first available. Thus, as nearly as reasonably determinable, all of the designs were developed at about the same point in time, and the designs completed without *Strange Eons* were unlikely to have been influenced by it. This rules out the effect of community experience as a source of error. (It is expected that the designs produced by a community would gradually improve in quality over time as the community members learn from each other.)

The designs were separated into two subsets depending on whether they had been created with *Strange Eons*. It was possible to determine this with virtual certainty by comparing a combination of factors: the presence of certain colour artifacts and the pixel dimensions of the template image used for the investigator cards, the selection of fonts and colours, and the precise positioning of the various textual elements on the card.

From each subset, 12 investigators were drawn at random, producing a control group and a treatment group. The control group consisted of 12 investigators (from 16) that were designed using traditional tools such as *Adobe Photoshop* and *Microsoft Paint* ("Traditional Methods"). The treatment group consisted of 12 investigators (from 18) that were created in the standard production version of *Strange Eons* available at the time, which included a consequence display ("Strange Eons"). Each of the designs in these groups was assigned a unique identifier using the same format as that used in the controlled experiment. The designs were then recreated in a standard version of *Strange Eons* without a portrait image so that they would be indistinguishable from other judged designs. Once the investigators from the controlled experiment were returned, all of the designs were combined and randomly

assigned to expert reviewers for judgement, as described above.

The remaining investigators were shuffled together, and six were selected at random. These were supplied to the expert reviewers as practice designs.

## 9.3 Results and Discussion

The results of the experiments reconfirm that the use of consequence displays benefits users of expert support systems.

Figure 9.2 is a graph of the overall scores that the reviewers assigned to the investigators from both experiments. (Since each investigator was evaluated by two different reviewers, there are two points for each investigator within a group.) It was intended to analyze this data using a statistical model that would have accounted for both inter-rater reliability and the pairing of reviewers, but it was later determined that this model was inappropriate. Consequently, these results are presented without a detailed statistical analysis.[2]

However, the graph shows a clear upward trend in overall scores between both control and treatment groups, supporting the hypothesis that the presence of the consequence display helped the investigator designers to produce higher quality investigators. In the observational study, the traditional methods group scores range from 3 to 8 ($Mdn = 5.5$) while the *Strange Eons* group scores range from 5 to 10 ($Mdn = 7$). In the controlled experiment, the no consequences group scores range from 4 to 9 ($Mdn = 6$) while the consequences group scores range from 5 to 10 ($Mdn = 8.5$). Notably, *both* control groups (traditional methods and *Strange Eons* without consequences display) ranked lower than the worst-performing treatment group (*Strange Eons*).

The overall increase in scores between the two studies may be due to a number of factors. First, the investigators in the observational study were all created at an earlier point in time than those in the controlled experiment; the overall skill level of designers may have risen during this time due to community experience. Second, participants in the controlled experiment knew that their designs would be evaluated and so might have put more effort into them. Third, the two experiments involved different versions of *Strange*

[2] A simple nonparametric test suggests the strength of this evidence, although statistical conclusions cannot be drawn since the scores are not independent. The observational study scores have Mann-Whitney $U = 143.5$ ($p = .002$, two-tailed), while the controlled experiment scores have $U = 48$ ($p = .002$, two-tailed). These low $p$-values suggest that a redesigned experiment would likely find a significant difference in both studies.

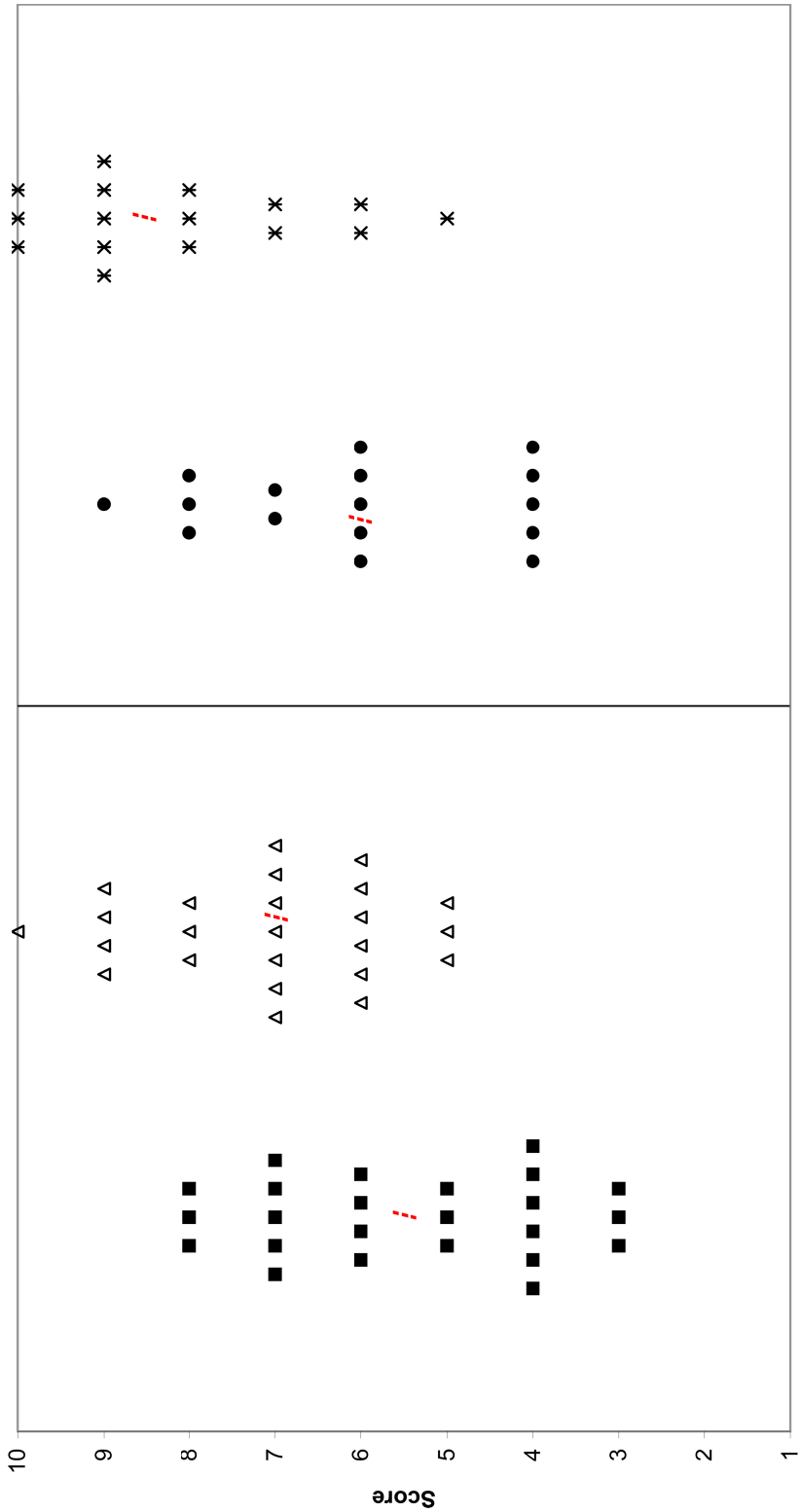



Figure 9.2: Overall scores assigned by the expert reviewers to investigator designs from all groups. A total of 40 designs were rated by 2 reviewers each. Ratings consisted of a score from 1 to 10, with higher scores indicating higher quality designs. A diagonal dashed line indicates the median of each group.

Table 9.2: Time Taken to Submit a Design After Downloading the Application

| Group | Turnaround time (days) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| No Consequences | 2 | 3 | 4 | 4 | 4 | 5 | 6 | 6 |
| Consequences | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 |

Table 9.3: Number of Inferred Design Moves Executed to Complete a Design

| Group | Inferred design moves | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| No Consequences | 19 | 22 | 25 | 31 | 34 | 37 | 37 | 52 |
| Consequences | 25 | 34 | 34 | 37 | 40 | 43 | 46 | 49 |

*Eons*. Each used the version that was current at the time that the investigators from the experiment were designed, so the version used in the controlled experiment was much newer. The investigator editor has seen steady improvement to both the number of consequences it provides and the size of the design space that it covers. Finally, the difference between the two control groups may be explained by the fact that even though the version of *Strange Eons* used in the controlled experiment lacked a consequence display, it still provided some other support features (such as variable and conditional tags) not available to the traditional methods group.

The turnaround times of participants in the consequences group ($Mdn = 2$ days; $M = 2.13$ days) were significantly shorter than those of participants in the no consequences group ($Mdn = 4$ days; $M = 4.25$ days; Mann-Whitney $U = 6$, $p = .005$). Table 9.2 lists the turnaround times for all participants. Traditionally, the time to complete a task has been one of the most important variables in HCI evaluations. It is far less important in expert work, where the quality of a solution is generally much more important than how long it takes to produce. However, when an expert support system helps users craft solutions twice as fast, that is a significant development—especially when the resulting solutions are as good or better than those produced without that support.

The test for the number of inferred design moves was not statistically significant: $t(13) = -1.38$, $p = .19$ (two-tailed). Table 9.3 lists the number of moves for all participants. The no consequences group performed a mean of 32.1 moves ($SD = 10.5$), while the consequences

group performed a mean of 38.5 moves ($SD = 7.69$). (An alpha level of .05 was used for all statistical tests.)

These results suggest that, while consequence displays help the user to understand the effects of different design decisions, they do not necessarily lead users to directly try more alternatives. The user may decide that the design is off, but may not be able to think of alternatives. Better support for Principle 7 (Encourage Parallel Exploration of Breadth and Depth), such as halo menus, could help by suggesting possible courses of action. Alternatively, the user may be limited by the need to manage alternatives. In this case, better history support (Principle 8—Provide Rich History Mechanisms) could free up cognitive resources by externalizing and organizing previously considered alternatives.

At least 19 moves are needed to create a complete investigator. The low minimum and wide range in the no consequences group (range of 33 in the no consequences group, versus 24 in the consequences group) suggests that while some participants in this group did explore alternatives, some implemented the first alternative that they considered (typical of novices); this does not appear to have happened in the consequences group.

The lack of significant difference in number of moves is notable given the large discrepancy in return times. The participants in the no consequences group may have been less confident in their designs due to the lack of feedback, and thus more reluctant to submit them. These participants may also have spent more time reflecting (the long durations particularly suggest reflection-on-action): while they performed similar numbers of moves as participants in the consequences group, it took longer to navigate these alternatives because they had no help evaluating their consequences.

While performing some degree of exploration leads to better designs overall, not all exploration is productive. Individuals with the soft cognitive style may meander about the design space before settling in to solve the main problem [221]. And while experienced workers may tend to perform more exploration, they do not necessarily choose more efficient sequences of commands compared to less experienced users [14]. When design moves must be inferred from command sequences, this may exaggerate the number of apparent moves that they perform.

Participants in the consequences group were asked to reflect on their experience with the consequence displays via a brief questionnaire after submitting their designs (see Table 9.3). One goal of this questionnaire was to verify that the consequence display was not used like a tutoring system and unquestioningly obeyed. Responses to the first two questions

Table 9.4: Responses to the Consequences Group Follow-up Questions

| Question[a] | Yes | No | Not sure |
|---|---|---|---|
| Did you *change your investigator* as a result of [the consequence displays]? | 8 | 0 | 0 |
| Did you *consider a change* to your investigator as a result of [the consequence displays] that you ultimately decided not to make? | 7 | 1 | 0 |
| Did [the consequence displays] help you to design a *better investigator overall*? | 5 | 0 | 3 |
| Did [the consequence displays] help you to design a *more balanced* investigator (one that is fair compared with other investigators, neither too powerful nor too weak)? | 5 | 0 | 3 |
| Did [the consequence displays] help you to design an investigator that is *more creative* (one with new ideas that set it apart from other investigators)? | 4 | 2 | 2 |
| Did [the consequence displays] help you design an investigator with *more replayability* (one that can be played in several games without becoming boring)? | 5 | 0 | 3 |

[a]Emphasis added.

clearly indicate that this did not happen, with all eight reporting that they changed their investigator at least once due to the consequences, and all but one reporting that on at least one occasion they chose not to act on a consequence.

The fact that all eight participants in the group reported using the consequence display was heartening. In *XDS*, the consequence display is difficult to avoid because it forms the main representation used to present designs. In *Strange Eons*, it is relegated to a small window in the corner of the editor. It is easily ignored if desired, and it can be completely hidden by dragging a splitter bar.

The remaining questions ask if the participant feels that the consequence display improved their design in various ways (overall quality, balance, creativity, and replayability). Of all 32 responses, participants answered negatively only 2 times, although about a third of responses were "Not sure". These participants may not have felt qualified to judge their own designs; alternatively, they may have been unable to later recall and compare the alternatives that they had considered.

The questionnaire features a high percentage of positive responses. This may indicate response bias and/or acquiescence bias in the results.

Overall, it is clear from the results of the studies that using *Strange Eons* with its consequence display feature benefits investigator designers. They can create designs as good as or better than designers without this aid, in significantly less time, and without loss of autonomy. The number of inferred moves they perform, while it does not show a statistically significant increase, at least suggests that coverage of the design space does not decrease.

## 9.4 Other Evidence

Note: I used "Other" rather than "Anecdotal" in the section heading due to the pejorative connotation of the latter term. Borrowing again from legal practice, the evidence which follows is essentially eye witness and expert testimony. Excluding the possibility of false testimony, which seems unlikely in the case of unsolicited and mostly anonymous feedback, such testimony is not fundamentally different from an existence proof.

*Strange Eons* has been available since November of 2006. Since then I have received many hundreds of positive messages describing how the application has helped the correspondent with various projects, ranging from fan-created expansions to commercial products. Many of

these messages have specifically praised the application's expert support features, including the consequence displays.

One of these messages particularly stands out, as it was sent by Kevin Wilson, one of the co-designers of *Arkham Horror*. He had started using *Strange Eons* to design content for *Arkham Horror* expansion products, and asked about the possibility of creating a similar support tool for a different game (his then-current assignment). I asked him to comment on his use of the tool, and he replied:

> Strange Eons has been great for me. Shortly after it came out, I started prototyping the new investigators and ancient ones [for subsequently published expansions to the base game] in it. ... As far as the [consequences display], I've found it to be pretty accurate. Obviously it can't judge how cool a special ability is going to be, but it's really on the ball when it comes to more definable things such as skill values and starting equipment. After awhile, I even used some of your experimental costs. ... Overall, I found Strange Eons' assessments to be right on the money.
>
> (Kevin Wilson, personal communication, July 24, 2009)

I already suspected that the designers were at least familiar with the application. Shortly after *Strange Eons* was first released, the first expansion to include new investigators was published. To verify that the application's model of the design space remained accurate, I used it to recreate the new investigators. This revealed some false assumptions in the model, as well as some relatively small transformations of the design space that the new designs introduced. The model in *Strange Eons* was updated accordingly, and afterwards I also began adding other small transformations to add new options for designers. (These are the "experimental costs" mentioned above.) For example, in the base game, an investigator's Sanity and Stamina scores always sum to 10. I added an option to disrupt this balance, but in order to compensate the designer pays for the extra points out of the $25 budget normally spent on starting possessions.

When the next expansion was published and I checked these investigators against the model, I was surprised to find that it accounted for the new investigators perfectly, including some that used the same kinds of transformations that I had added. The one exception was an investigator that incorporated a radical transformation that required extensive modification to the card's graphic design. If the designers included this design in response to effective

competition from the amateur design community, then this was certainly an unanticipated benefit of *Strange Eons* and its implementation of the design principles!

## 9.5 Considering Individual Effects

As a starting point, it is fine to say: "Here are some design principles for building expert support systems. If you use them, good things will happen." However, support system designers will soon come back with questions like "which principle is the most important?", or "I only care about getting the user to consider more alternatives; which principles do I need to support?" While it is possible to make reasonable guesses as to what effects a principle should have, it is important to verify such assumptions before applying them to real projects. Besides, many principles could have unexpected side effects—good or bad. Support system designers need to be aware of these in order to find good solutions. Finding answers to such questions would require years of work, but because *Strange Eons* focuses on a smaller subset of the design principles than *XDS*, it does provide a place to start.

Based on the inconclusive results regarding design moves in the controlled experiment, consequence displays (and hence Principle 6, and to a lesser extent Principles 2 and 3), may be insufficient to get some users to explore a variety of alternatives. However, it appears that these principles can help users find good solutions more quickly.

Since the results of the *XDS* study *did* find that users explored more alternatives, principles supported more strongly in *XDS* than *Strange Eons* may be responsible for this difference. Likely candidates are Principle 7 (Encourage Parallel Exploration of Breadth and Depth) and Principle 8 (Provide Rich History Mechanisms): both are strongly supported by *XDS*, unsupported by *Strange Eons*, and relate directly to alternatives.

This assertion can be probed using Mohseni's *Treesta* [125,186], an expert support system that supports the *analysis* of designed experiments. (Where *XDS* supports experimental design up to the point of gathering data, *Treesta* can be used to support the process of analyzing data once it has been gathered.) While *XDS* and *Strange Eons* both implement support for many principles, *Treesta* implements only moderate support for Principles 7 and 8. (This pattern is roughly the complement of the pattern of support provided by *Strange Eons*; see Table 9.1 on page 132.)

*Treesta* acts as a front-end to the *MATLAB* statistical computing language. It records *MATLAB* commands and their results in a history after forwarding them on to *MATLAB*

for evaluation. The history is automatically organized into a tree of *workspaces*. A new workspace is created each time the user performs an analysis, and includes all of the commands and data to that point.

Unlike the simple linear command history provided by *MATLAB*, the representation of each analysis as a separate workspace makes searching the command space visible, supporting Principle 7 (Encourage Parallel Exploration of Breadth and Depth). It also has the ability to display a summary view of all workspaces, allowing the user to reflect on the history of analyses performed so far, supporting Principle 8 (Provide Rich History Mechanisms).

A qualitative study [186] with a single user confirmed the usefulness of the support provided by *Treesta*. The study did not specifically consider whether the tool increased the number of alternatives explored, but the participant's comments strongly suggest that it had this effect: "You think about your next step in a more structured way; you set some goal about what you want to achieve in this workspace, and you work around until the goal is achieved." [186, p. 60]. This supports the hypothesis that the principles have complementary, and probably synergistic, effects: that, for example, helping the user think about the effects of their actions and helping the user get an overview of what those actions have been does not produce the same benefits. While support systems that supplement traditional application designs by supporting a few of the principles will benefit, maximum benefit will accrue to applications that, like *XDS*, make broad support of the principles a design priority.

This analysis suggests some of the effects that individual principles may contribute to expert support, but it is neither complete nor conclusive. Much more work is needed to achieve the depth of understanding that would allow support system designers to make informed trade-offs.

# Chapter 10

# Convergence: Conclusion and Future Work

## 10.1 Closing Arguments

Chapters 5–9 presented the available evidence in support of the design principles. Based on the strength of this evidence, an initial verdict of the principles' feasibility and effectiveness can be made. The principles will be judged by their ability to satisfy the following hypotheses (first presented in Chapter 4):

1. There is at least one expert support system designed in accordance with the principles that has a positive impact on expert work.

2. In general, expert support systems designed in accordance with the principles will have positive impacts on expert work.

3. The design principles meet at least some of the additional criteria for "good" design principles. (These are repeated below.)

*XDS* is an obvious choice to satisfy hypothesis (1), since *Strange Eons* and *Treesta* are primarily discussed as reactions to it. However, a benefit of presenting three broadly different prototypes is that any one of these systems can potentially satisfy hypothesis (1), even if flaws are identified in the evidence presented for the other two. Given that all three prototypes provide evidence of positive outcomes, hypothesis (1) can be accepted as the

best explanation of the evidence (the competing explanation being that the hypothesis is false). Consequently, the principles meet the minimum criterion necessary to be worthy of further investigation [92], regardless of any other outcomes.

Hypothesis (2) makes a much stronger claim. To ascribe a causal relationship between the principles and the evaluation of a single prototype invokes an especially weak abductive argument; further inferring that these effects generalize to the broader class of expert support systems would be dubious indeed. A better way to bolster support for hypothesis (2) is to perform repeated testing with multiple prototypes, but for this to be convincing the prototypes must be chosen for consilience—especially if the number of prototypes is small. The three systems discussed here yield highly consilient evidence: they tackle different kinds of expert work; their problem spaces have different degrees of formalizability; they implement different subsets of the principles; they represent different degrees of intervention compared to contemporary application designs (from a subtle extension to typical application design to a radical paradigm shift); and they take different approaches to modelling the problem space, from complete (*XDS*), to incomplete (*Treesta*), to a multilayered approach that can be extended by the end-user to support specific subspaces and transformational creativity (*Strange Eons*). Even the scale of the support systems ranges from an add-on for an existing system (*Treesta*, about 4 000 lines [186, p. 57]), through a small standalone application (*XDS*, about 24 000 lines), to a large-scale suite of tools that itself provides extensive support for add-ons (*Strange Eons*, about 250 000 lines when support for *Arkham Horror* is included). Given this, and since all three systems are supported by evidence of positive outcomes, hypothesis (2) can also be accepted as the best explanation of the evidence. An alternative explanation competing with hypothesis (2) is that the limitations on the studies (including those listed in Chapters 6 and 9) together account for the results. But each of these limitations is a separate hypothesis about a single result; the competing explanation is thus less *simple* (see p. 48) than the explanation given by hypothesis (2).

Satisfying hypothesis (3) is not strictly necessary. Satisfying the first two hypotheses recommends them as guides to design; satisfying hypothesis (3) is an indicator of their quality *as principles*. The additional criteria referred to by hypothesis (3) are:

1. Good design principles should inspire further research and innovation.

2. Good design design principles should help to identify open problems and gaps in current understanding.

3. Good design principles should prove to be a solid framework for understanding the domain in which they apply.

If the principles are hard to apply and do not help to expand the space of design possibilities that the support system designer can imagine, they probably need to be reworked. It suggests that the principles are too restrictive, placing so many constraints on designs that the resulting design space is too sparse to support innovation: the principles would amount to design constraints that limit creativity rather than enhancing it. Principles that do not satisfy this hypothesis are unlikely to have broad, long-term impact, because the narrowing effect of their constraints means that their potential for guidance will soon be exhausted.

Evidence for all three of the qualities that make up hypothesis (3) have been presented in this dissertation. In terms of innovation, the following major interaction techniques have been introduced:

**Explicit Traversal of Problem Spaces** Explicit problem spaces allow immediate access to history and support its replay and non-destructive editing. They support rapid comparison of alternatives, and they provide an overview of the coverage of the problem space. Although the explicit space presentation in *XDS* makes heavy use of graphical summaries, this is not the only viable approach. An explicit document space might summarize the differences between states. And zooming user interfaces can help move between structural views and full details quickly and fluently.

Notably, the explicit problem space representation in *XDS* is built atop a cooperative design space explorer. Previous cooperative design space explorers have been criticized for modelling only toy problems [4]. But *XDS* implements a complete model for a domain of tremendous practical importance. It therefore establishes design space exploration as a valid approach for at least some domains. Also notable is that *XDS* models an abstract mathematical space; previous cooperative design space explorers have focused on geometric design spaces (owing to their roots in shape grammars).

**Consequence Displays** Consequence displays can help the user understand their location in a problem space, pose and test hypotheses about courses of action (supporting reflective practice), and make explicit important causal effects that would otherwise be hidden or need to be estimated independently. Although consequence displays can provide the most detail when the problem space can be completely represented by the

application and the consequences are quantifiable, most problems have at least some significant consequences that can be represented effectively.

**Halo Menus** Compared to other techniques, halo menus provide a balance of discoverability and accessibility. They increase the user's awareness of available options in an effort to coax the user into trying more—and different kinds—of alternatives. While this is particularly useful when novices are performing expert work, any task that is enhanced by thorough experimentation could benefit from this technique. Accordingly, claims analysis has been used to elevate this technique into a more general form suitable for application in other domains, both within and without the support of expert work. That the principles should inspire a novel command selection technique is particularly notable, because this area of HCI has already been heavily mined (e.g., [2, 19, 35, 72, 96, 114, 140, 142, 260]).

In addition to inspiring innovation in HCI, the design principles also fulfil the second criterion by identifying open problems. Many of these have already been discussed, particularly in Chapters 2 and 3, but the list of problems is further discussed and expanded in Section 10.3.

The third criterion for good design principles has the weakest support (although both the first and second criterion could be considered special cases of the third.) There are sporadic examples in this dissertation that use the principles to interpret, classify, and apply various results; perhaps the clearest example is their use when evaluating *Treesta* at the end of Chapter 9.

A difficulty when trying to interpret information in the framework of the principles is that their definitions are complex and often subtle. Even I found the need to parse some definitions carefully when using the principles this way. Future work that further clarifies the relative roles of the principles may allow the definitions to be further limned and simplified.

While identifying this limitation early on is valuable in directing future work, in the end it is too early to give a complete assessment of criterion (3). It can only be accurately gauged as new results appear in the domain that could not have been anticipated when the principles were written [255].

Although support for the third criterion for quality may not be strong, there is ample evidence for the other two. Hypothesis (3) is well represented—though with room for improvement. By meeting the test of all three hypotheses, the design principles have been

shown to not only be worthy of further study, but also ready for careful application to real production systems. (Indeed, *Strange Eons* is such a system.)

The form of this argument has also been a useful test case for the feasibility of accepting a broader range of evidence in the HCI community, to be judged by its explanatory coherence. Explanatory coherence theory encompasses the standards for accepting explanations in science, law, and other domains—while HCI has traditionally favoured quantitative methods adopted from psychology. By embracing this larger theory, it is possible to adapt concepts from these other domains as well as methods from HCI that have been proposed but are not yet widely accepted.

I have found two concepts adopted from legal proceedings to be particularly helpful in preparing the evidence in this dissertation: the use of appropriate standards of proof and the structuring of arguments as factual matrices of disparate supporting evidence.

The concept of different standards of proof for different kinds of research is implicitly understood in HCI, at least in some circles, but putting a name to the concept promotes a more thoughtful consideration: For example, while the evidence for hypothesis (2) is stronger than that of an existence proof, should this difference be considered substantial? The standards of proof used in law are a useful starting point for such an evaluation. These standards of proof include, in order of increasing strength [153, 183]: the precautionary principle; preponderance of evidence; clear and convincing evidence; beyond a reasonable doubt; irrefutable.[1]

The precautionary principle, which allows legislation to be enacted even in the face of disputed scientific evidence, is broadly comparable in strength to the existence proof. It is also comparable to the standard for probable cause [183], which is the basis upon which police are granted warrants to search for additional evidence—roughly analogous to the application of existence proofs in HCI.

The next highest standard, preponderance of evidence, requires sufficient evidence to show that something is more likely true than not. Regarding the principles, considering the basis in prior research, the variety of the prototypes,[2] and the lack of unexplained negative

[1] In legal parlance, irrefutable evidence refers to expert testimony that can only be refuted by another expert witness [183].

[2] In systems that place the burden of proof on the accuser, similar fact evidence (evidence of similar misconduct in the past) is considered so convincing that it is usually inadmissible unless its probative value can be shown to exceed its potential prejudicial effects. When similar fact evidence is admitted, it alone can establish causality at the "beyond a reasonable doubt" standard applied in criminal proceedings [183].

results [224], it is plausible to the author that the evidence presented here does meet this standard. As in law, however, this question can only be decided fairly by unbiased observers.

In addition to the use of standards of proof, viewing arguments as factual matrices has been particularly helpful as an organizing structure. The factual matrix developed over the past several chapters presents evidence from design rationale, claims analysis, expert opinion, and reflection in conjunction with traditional empirical research. Of particular note, halo menus were developed, evaluated, and generalized from *XDS* almost entirely through argumentation. Even when evaluating their selection efficiency I chose an argumentative method based on a simple GOMS analysis when a straightforward quantitative evaluation technique was available. My reasoning was that the reliability of GOMS has already been established (see p. 49, though that refers to a more accurate variant), and since selection efficiency is not the primary goal of halo menus, the cost of obtaining a precise quantitative result is not justified by the needs of this argument. This decision embraces the assertion that researchers can and should choose methods based upon which questions are important and not upon which questions are easily answered or readily published.

Overall, the success of this approach to making HCI arguments should serve as an encouragement to others. Researchers should be open to both generating and accepting untraditional arguments—especially when this allows the researcher to ask more valuable research questions—as long as those arguments meet the standards of explanatory coherence.

## 10.2 Limitations

Expert work is an intentionally broad term that covers many domains. It is possible that some of these domains, although they employ similar processes and face similar problems to other expert domains, may not allow effective computer support. An important factor in this determination is whether the value added by the support tool outweighs the cost of using it when compared to alternative methods.

One aspect of this cost is captured by a method's *directness* and *commitment* [193]. The directness of a method is a function of the immediacy and degree of translation required to use it. Sketching is a very direct activity: the sketcher simply picks up a pencil and draws whatever comes to mind. Working through a computer is less direct than working with a passive medium, since the user's gestures must be digitized and interpreted before a result can be generated. Computer tools often require the user to interrupt their work to order

to translate their intent into indirect commands. If one wishes to place one sketch next to another, one simply moves the hand and continues drawing. To do this in a computer drawing application might require inserting a new sketch area, selecting it and the original sketch, choosing a command to make the sketch areas the same size, choosing alignment commands to place them next to each other, choosing an edit command on the new sketch area, and then finally starting the new sketch.[3]

Commitment refers to the fixedness of meaning in a representation. The commitment required by a given method is a function of its ability to accept and represent imprecision and ambiguity: one of the major benefits of sketches is their low commitment, which allows them to be reinterpreted as a means of generating new ideas. Typical computer applications, however, have an inherent degree of commitment that stems from their need to represent objects in a form that can be manipulated algorithmically. To draw a line algorithmically, the line must have certain attributes—coordinates, thickness, colour, dash pattern, and so on—each of which has a precise meaning and interpretation. Forcing the user to choose these attributes ahead of time is a reversal of the dialectic process in sketching, in which the expert first creates a representation and then perceives and interprets its attributes [193].

Most applications have low directness and high commitment compared to sketching, but they also offer benefits in trade. For example, if the expert later wants to rearrange sketches, or to delete some of them in order to focus on others, this can be done quickly with software, while the pencil-and-paper sketcher might need to redraw the sketches on a fresh page. This flexibility is a result of the directness and commitment balance in the application: treating each sketch as a separate object requires more commitment than treating them as marks on a page, while the availability of abstract commands (edit, delete, move, and so on) are possible due to the computer's indirectness. For a support tool to be attractive, the benefits of this added flexibility must be greater than the costs of its decreased directness and increased commitment.

While certain domains may present barriers to effective computer support, acceptance by individual users may also be an issue. This may indicate a failure to apply Principle 5 (Engage Multiple Ways of Doing and Thinking), but there are limits to such application. Interaction designers are admonished to study how people currently perform a task as a basis

[3]Such patterns of interaction are another example of how current computer tools emphasize convergence. The interface designer implicitly expects that the user will not just want to make two nearby sketches, but that they will need precise control over size, position, and the atomicity of the entities.

for designing new systems, but this may be misleading when the expert uses a combination of ad hoc methods and support tools that overemphasize convergence.

The individuals most likely to resist change are those who are the most experienced and have the most invested in current methods. These individuals already have personal techniques for interpreting and navigating the problem space. Because ill-structured problem solving is a bringing together of a problem and a solution, their methodology determines not just how they approach solving a problem, but also how they define and understand it. For these experienced individuals, learning to use a support system may be more difficult than it would be for a novice because they will hang on to the safety of familiar methods and attempt to translate them into the features offered by the support system. I observed this phenomenon firsthand while discussing *XDS* with a statistical consultant. When the aims of *XDS* were described in general terms, he was highly enthusiastic about the system, but when a prototype system was demonstrated for him, he had difficulty following the presentation. It became apparent that this was in part because he was trying to guess the purpose and meaning of various parts of the display based on his own experience.

A similar phenomenon was observed during the study conducted with *XDS*. I observed that the most experienced participant had the hardest time using the application, while the second-most experienced participant had the easiest. This suggests that users with an intermediate degree of expertise will be most receptive to expert support: they will have enough experience to understand the system's features, but their methods are not so rigidly ingrained that it inhibits learning a different approach. (Both of these participants were professors in university computer science departments, so their general familiarity with computer interfaces is unlikely to account for the difference in their learning performance.)

Whether it is a domain that resists support or a particular user, the main factor in determining its eventual acceptance is the same. If an expert uses the support tool long enough to get past their preconceptions of it, they will accept it if they find a significant advantage over their usual methods. The user base of *Strange Eons* illustrates this effect. The main alternative to *Strange Eons* is to use a graphics application. While most *Strange Eons* users develop custom content with it either exclusively or in combination with other tools, there are a small number of users who only use it to view content created by others and who use a separate graphics application to develop custom content of their own. This choice is easily explained in terms of a cost-benefit analysis. Like *Strange Eons*, once a template has been set up for a particular type of component, it is reasonably efficient to edit that

template to create a specific design. It is not as easy or efficient as using a dedicated tool, but it is comparable if one has already learned the graphics application. The net result is that, for these experienced users, *Strange Eons* has only a small advantage in editing time for most components. (It has a large advantage in learnability, which is why this effect is only seen with experienced users of graphics applications.) The critical difference is in how these applications support transformational creativity. In the case of the graphics application, this requires a skill that the user already has (using the graphics application), while for *Strange Eons* it requires a skill that the user may not have (light programming). Thus, these users are willing to endure a small penalty in the common case for the reassurance that they can use familiar methods when a problem calls for transformational creativity.

It may appear that part of this cost-benefit analysis is missing, because no mention has been made of the expert support features in *Strange Eons*, and particularly of consequence displays. In fact, this distinction is critical, because the users that choose to keep using a graphics application work almost exclusively with games for which the plug-in author has declined to provide additional forms of expert support. Should these games add such support in the future, these users might migrate to *Strange Eons*.

Another limitation is that the design principles are probably incomplete: research in this area is ongoing and there are many gaps to fill. In this respect, it is advantageous that the principles were derived from many sources. The success of a set of principles derived from a single theory or model (such as Norman's interaction design principles [201]) is tied to the success of that theory. A set of principles derived from many sources, while possibly harder to understand and apply, can adapt to change more readily—and as missing information is incorporated, it can still converge to a stable form over time.

Two aspects of the principles that will become clearer in time is the degree of independence and the relative importance of the principles in comparison to each other. At this point there is insufficient evidence to make a compelling argument about either aspect, but there is enough for some initial speculation.

The relative independence of principles is important because the more interdependent the principles are, the more rapid the rate of diminishing returns as more and more are applied. Understanding this relationship helps designers of support systems to balance the benefit of applying the principles in conjunction with competing concerns. Although interdependence can indicate unnecessary duplication or overlap, it can also simply indicate that a phenomenon is complex or can only be manipulated indirectly. In such cases it is

useful to be able to manipulate it in multiple ways while designing, since different approaches will have different consequences for the design.

With only a handful of examples as a basis, there are only a few common points for comparison, though a few early results are presented at the end of Chapter 9. The evidence there suggests that good feedback and support for hypothesis testing are much more effective when used in conjunction with a mechanism to encourage exploration of a depth and breadth of solutions, as halo menus do.

Relative importance is difficult to assess because the different principles support different needs of expert work, and therefore produce different outcomes. Choosing one of these needs as more important than others is subjective, and at any rate probably depends on the individual strengths of particular users. Instead of deciding on a single metric for comparing the contribution of the principles, it will be more useful to support system designers to have lists of their individual strengths and weaknesses, which will allow them to tailor tools to the needs of particular domains and groups.

The design principles are also incomplete in another sense: they focus on extending support to the processes of expert work, but there are other factors to consider. (This is unavoidable: interactive system design is ill-structured problem solving; the context of its problems can be extended arbitrarily.) One such consideration is the social dimension of expert work, which has two relevant aspects.

The first social aspect of expert work is that many experts may contribute to a solution. *Computer-supported cooperative work* [94] is a large and well-established area within HCI that focuses on this aspect of interaction design [13]. Interested support system designers should familiarize themselves with this research, especially when supporting a domain where collaboration is common. While collaboration implies active contribution, an expert's social interactions contribute to expert performance as well. Coughlan and Johnson [48] point out that Schön's repertoires represent just one of the resources that influence idea generation. All of our experiences, including social ones, form the stock of memories that are associated to produce new ideas. Accordingly, they emphasize that the long-term development of creative potential depends not only on collecting ideas and artifacts, but also on building social relationships.

The second social aspect of expert work is that, while it may be performed by individuals or small groups, its value is ultimately judged by a larger society [52, 275]—for example, via peer review. The design principles in this dissertation may help expert workers to discover

innovative solutions to important problems, but if they can not be presented in a convincing manner to societal gatekeepers, they will have little impact. Developing a solution and communicating it to others are generally distinct aspects of expert work that might best be dealt with using two or more separate support systems. Nonetheless, designers of support systems should consider the expert's need to communicate findings to a larger audience. (Although not part of the implementation of the principles, both *XDS* and *Strange Eons* do support informal design rationale annotations in recognition of the importance of critical review to improving the quality of expert work.)

## 10.3 Open Problems

Each of the principles represents a broadly defined open problem: namely, finding the most effective techniques for the principle's implementation. In addition, the process of developing the principles and their subsequent evaluation posed several more specific problems:

The principles focus primarily on promoting designs that are expected to enhance expert work. An alternative is to discourage designs that might inhibit it, or to build systems that can correct such conditions if they are unavoidable. For example, a tool that supports writing might either avoid triggering, or else treat, the phenomenon known as writer's block. (Flaherty proposes some technological interventions based on a hypothesized neurological origin for the condition [73].)

As discussed in Section 2.5.3, diagrams and sketches have been shown to have a number of effects on problem solving. Less well understood is how these effects interact, and how vital the *act* of sketching is compared to the results. The evaluation of *XDS* points out the similarity between the explicit design space representation populated with design summaries and a sketch pad (see p. 92). The discussion there speculates about the value of creating or arranging sketch-like entities automatically, and in particular whether automatic arrangements have the same juxtapositional benefits as traditional sketching. Although automated arrangement did not appear to hinder problem solving, that is far from showing that it helps. Future work should follow up on these suppositions, and also compare the similarity-based organization used in *XDS* to alternatives like the history trees used in *Treesta*.

Section 2.6 introduced de Bono's techniques for creativity enhancement. While popular, these techniques have not been validated in scientific studies [243, p. 3]. Investigating de Bono's claims would place both creativity and expert support on a firmer foundation.

During the discussion of Principle 3 (see p. 37), it was pointed out that the inherent precision of computers could be at odds with the need for ambiguity in partial solutions. It was further suggested that clever presentation forms, imprecise input methods, or fuzzy systems might be examples of ways to combat this precision. Interested researchers could pick up this thread by comparing the feasibility of these and other techniques.

The problem of supporting transformational creativity was discussed in multiple places. When it was introduced in Section 2.6, it was suggested that it might best be avoided in favour of more specific support. That statement notwithstanding, solutions that entail acts of transformational creativity also have the greatest potential to revolutionize a problem domain. Moreover, advances in modelling transformational creativity would be important not just for expert support, but for artificial intelligence and psychology as well. Transformational creativity remains a vital topic for theoretical research, regardless of how well it can be integrated into expert support systems.

The discussion of Principle 4 (see p. 38) suggested that one approach to supporting transformational creativity would be to build a general, flexible core for others to build upon. The latter is in effect what *Strange Eons* does; in this case, traditional end-user development using a scripting language provides the needed flexibility. However, it is far from clear that this is the best method, or that a well-chosen subset of more accessible features might not provide a comparable degree of flexibility while including more users. For example, many different component types in *Strange Eons* could be built using a simple system of drag-and-drop control layout to create an editor (paired with matching operations to draw the card face), and simple pattern-matching rules to detect relevant consequences.

Principle 5 (see p. 39) also touches on the value of end-user development, although in the interest of making the features of a support system available to experts who need to work with multiple tools or who want to implement customized support for their preferred work processes. The dream of opening up vast possibilities by making programming more accessible to the masses has been around since the development of *Smalltalk* for the *Star Information System* and earlier [130]. It remains an important area, not just for its potential to make transformational creativity more feasible, but for also for its ability to give expert workers the freedom to adapt and replace the contents of their problem-solving tool kit. As alluded to on page 39, there are two directions that this work could take: pursuing alternative ways to create and represent programs, and defining programming structures and idioms that allow users to express relevant programs simply and succinctly.

The discussion of Principle 7 (see p. 41) noted that little is known about the ideal amount of exploration when comparing alternative solutions. Too little exploration limits discovery to weak solutions, and too much exploration bogs the expert down manging the set of solutions—but how much is too much? Having data on the ideal amount of exploration for different domains and problem types (and even whether this varies) would assist expert support system researchers in answering such questions as: Can the presence of computer support extend the number of solutions that can be explored without negative consequences? If so, does this lead to better solutions overall? Can computer support reduce the amount of exploration needed to reach a given quality of solution? Given enough data of this kind, the effect on the required and possible amounts of exploration could become a standard of comparison used to measure new support system designs.

Better support for Principle 9 (see p. 42) could be achieved with a deeper understanding of the role of repertoires in expert work. In particular, how important is it that the expert have seen the artifact previously? Does exposure to the artifact cue the recall of internalized knowledge that it is associated with? Does the artifact directly prompt the identification of consequences and relationships in the current situation as in the dialectic process? If both, which is more important? The answers to these questions will help determine how to best model repertoires in software.

Of all of the design principles, the prototypes evaluated so far have provided the least support for Principle 10 (Create an Effective Environment, p. 42). This dissertation is primarily interested in software interventions, and software has a limited ability to control the external environment in which it runs. However, software can exert control over the *software* environment in which it runs, and since the user's attention is presumably focused in large part on the display, this limited control might nonetheless be leveraged to great effect. One source of ideas is the large body of experimental data from cognitive psychology that investigates how various contextual factors affect cognition. A thorough review of these results could provide a wealth of inspiration for support system designers. One caveat of such a review is that psychological studies are concerned primarily with statistical significance, as this indicates results that may be important to understanding how the mind works. Support system designers must also be concerned with effect magnitudes, as this indicates whether a given manipulation will have an appreciable effect in practice.

In recent years, HCI practitioners have shown an increased interest in the emotional component of interaction design [203]. While this discussion is sometimes driven by a desire

to increase sales, in expert work and other creative tasks there is a deeper reason to consider emotional affect. While work is commonly seen as the opposite of fun, work can be very enjoyable when it is interesting and challenging. Since people report a strong sense of enjoyment when engaged in flow [51], the level of enjoyment experienced by a support tool user is an important indicator of its capacity to support this highly productive state.

I did not ask participants if using *XDS* or *Strange Eons* was enjoyable. However, the participants in the *XDS* study did appear to enjoy exploring the space and trying new moves, and many *Strange Eons* users have indicated enjoyment or fun as one reason that they use the software. When testing expert support tools, impact on emotional affect should be a standard part of evaluation.

As mentioned earlier, a vital area of future work is to develop additional implementations of the principles. *XDS* and *Strange Eons*, while chosen for their differences, still represent only a handful of many possible alternatives. Applying the principles to still more domains provides an opportunity to implement and evaluate still more approaches. If these attempts are documented and generalized using techniques such as claims analysis, the result will be a growing library of techniques for designers to draw on, and a growing body of evidence for researchers to build on. For example, while explicit problem spaces are a powerful way to capture and present histories, they may only be an option for domains where the problem space can be formalized. In other domains, a different approach may be necessary. An alternative for domains that feature structured information is to offer localized histories tied to those structures. Like an explicit problem space, this adds a spatial dimension to edit histories that is usually absent. In writing, for example, it might be possible to adapt recent work on collaborative undo mechanisms [248] by tagging and tracking each sentence as it moves between various document structures (paragraphs, sections, chapters) during editing. This would allow history manipulation to be localized to a selected structure on demand: for example, one could scan back through the history of a single paragraph while ignoring intervening changes to other parts of the document. While not as powerful as working directly with moves, this would still filter out many extraneous actions, allowing the writer to focus on the structural element of interest, which should also approximate the semantic element of interest (excepting cross-cutting concerns [132]).

Examples of adapting consequence displays to different domains have been discussed already. The introductory example of a consequence display for document length demonstrates that one may still find useful, computable consequence even in spaces with consequences

**Similar Monsters**

The following standard monsters are the most similar to this one:

| G | Mv | Aw | HM | HD | CM | CD | Tg | Am | ∞ | Un | M/S | EI | WI | PI | MI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ygroth** | | | | | | | | | | | | | | | |
| | | -1 | +1 | 1 | -3 | 1 | 2 | × | - | - | | - | - | - | - |
| **Ghast** (fairly) | | | | | | | | | | | | | | | |
| | | -2 | +0 | 1 | -3 | 1 | 2 | × | - | - | | - | - | - | - |
| **Nightgaunt** (somewhat) | | | | | | | | | | | | | | | |
| | | -2 | -1 | 1 | -2 | 0 | 2 | - | - | - | | - | - | - | - |
| **Combat check**: if failed, drawn through nearest gate | | | | | | | | | | | | | | | |
| **Goat Spawn** (somewhat) | | | | | | | | | | | | | | | |
| | | -1 | -1 | 1 | -2 | 1 | 2 | - | - | - | | - | - | -½ | - |
| **Shan** (somewhat) | | | | | | | | | | | | | | | |
| | | +0 | -2 | | -2 | 1 | 2 | - | - | - | | - | - | - | - |
| **Horror check**: if failed, you are **devoured** | | | | | | | | | | | | | | | |
| **Ghoul** (dissimilar) | | | | | | | | | | | | | | | |
| | | -3 | +0 | 1 | -1 | 1 | 1 | × | - | - | | - | - | - | - |

Figure 10.1: The consequence display for *Arkham Horror* monsters. The top entry ("Ygroth") is the monster being edited; subsequent entries are the closest official monsters, in order of increasing distance. The blue label next to the monster name describes its distance from the design using one of four categories (from "very" to "dissimilar"). The columns summarize each monster's attributes. By comparing a design to the most similar monsters, the user can quickly gauge the design's originality and in-game difficulty.

that are mostly unamenable to computer representation. This is an important lesson for designers of expert support systems: it is better to provide weak support than none at all. Not all spaces can be expressed as neatly as the space of ANOVA designs, and that is fine. The goal of the support system is not to replace the expert, but to amplify their abilities [68]; rich models of the domain are not always necessary to accomplish this. Indeed, spaces that are the hardest to support might also be those that benefit most from support.

Other spaces, even closely related ones, will call for still other approaches. For example, the *Arkham Horror* monster editor in *Strange Eons* uses a tabular consequence display (see Figure 10.1). The table lists the official monsters (including a summary of their attributes) that are closest in distance to the monster being edited. The distance metric is a simple adaptation of Levenshtein distance [158] to multiple dimensions; it is approximately equal to the number of edits that would be required to transform the attributes of one monster into

those of another. (The monster's name and portrait are not considered; only the attributes that affect how it behaves in the game participate.)

Where the investigator editor tells the user when they have crossed certain boundary lines in the space, the monster editor tells the user where they are in relation to a set of prototype designs. In both cases, this is tailored to the needs of the expert working in that space: while an important consideration for investigators is fairness, the two most important factors when judging a monster are its uniqueness and its difficulty. A monster that is similar to existing monsters in everything but name does not add an interesting challenge to the game; a monster's difficulty should reflect its rank in the established pantheon of the Cthulhu Mythos, and also helps determine how many copies should be included (harder monsters are also rare). The monster consequence display allows users to quickly gauge both of these consequences: originality can be gauged from how far away the most similar monsters are, while difficulty can be gauged relative to the known difficulties of similar official monsters. As more systems that incorporate consequence displays or similar support become available, the use of claims analysis will make it easier to compare different display techniques and identify the most effective techniques for different domains.

A final long-term goal is the development of one or more measurement instruments to gauge the effectiveness of expert support systems based on the design principles, or on the previous work from which the principles were derived. (The Creativity Support Index [38], which serves a similar role for creativity support tools, should be a useful model for this task.) This would allow support system designers to evaluate new prototype systems in a standardized way that simplifies the comparison of different systems and techniques.

## 10.4 Concluding Remarks

The extraordinary potential of computers to support expert work has been recognized since the early days of computing. As hardware has become more powerful, specialists have responded by creating systems ever closer to realizing those early visions. Where once computers were just another tool on the expert's belt, now they are nearly indispensable to the task of expressing, refining, and sharing the results of expert work. The next advance is to move beyond a product-centric approach and encompass support for the earlier stages of expert work. The result could be a shift as radical as the one from tools that support computing artillery trajectories to tools that support editing novels.

The design principles offered by this dissertation represent one step towards this goal, and they have found firm ground. They have passed their inaugural evaluation, demonstrating the potential to help experts find better, more thoroughly considered solutions, and to do so more quickly than before. The principles have also proven hardy. Even when applied sparingly, they can produce substantial benefits. The success of *Strange Eons* is a notable example. As a hybrid that straddles the gap between contemporary support and the more extreme support exemplified by *XDS*, its success means that designers can start improving support tools today—without convincing managers of the need to redesign flagship products from scratch. This is heartening. After all, expert workers make decisions that impact every area of our lives, from the placement of the switches on our coffee makers to the most sweeping government policies. Navigating the alternatives in these problem spaces is challenging, and it is in everyone's interest that the experts trusted with these decisions be empowered to find the best possible solutions. Let's take another step in the development of this support and see where it leads.

# Bibliography


[1] J. Accot and S. Zhai. Beyond Fitts' law: Models for trajectory-based HCI tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 295–302, New York, 1997. ACM.

[2] J. Accot and S. Zhai. More than dotting the i's—foundations for crossing-based interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 73–80, New York, 2002. ACM Press.

[3] Ö. Akin. Variants in design cognition. In C. M. Eastman, W. M. McCracken, and W. C. Newstetter, editors, *Design Knowing and Learning: Cognition in Design Education*, pages 105–124. Elsevier, Amsterdam, 2001.

[4] Ö. Akin. The whittled design space. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20(2):83–88, 2006.

[5] C. Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1977.

[6] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.

[7] R. Andorno. The precautionary principle: A new legal standard for a technological age. *Journal of International Biotechnology Law*, 1(1):11–19, 2004.

[8] S. Appelcline, K. Wilson, C. T. Petersen, and G. Benage. *Arkham Horror Rulebook*. Fantasy Flight Games, Roseville, MN, USA, second (revised) edition, 2006. Retrieved April 10, 2010, from `http://www.fantasyflightgames.com/ffg_content/Arkham_Horror/AH_Rules_internat.pdf`.

[9] K. Arnold and A. Lam S. L. Napkin Look & Feel. Retrieved May 12, 2009, from `http://napkinlaf.sourceforge.net/`, 2009.

[10] I. Asimov. *Words from the Myths*. New American Library (Signet), New York, NY, USA, 1969.

[11] C. J. Atman, J. R. Chimka, K. M. Bursic, and H. L. Nachtmann. A comparison of freshman and senior engineering design processes. *Design Studies*, 20(2):131–152, 1999.

[12] B. J. Baars. *A Cognitive Theory of Consciousness*. Cambridge University Press, Cambridge, UK, 1988.

[13] R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg. Groupware and computer-supported cooperative work. In R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg, editors, *Readings in Human-computer Interaction: Toward the Year 2000*, pages 741–753. Morgan Kaufmann Publishers, San Francisco, CA, USA, second edition, 1995.

[14] J. L. Baher and B. Westerman. The usability of creativity: experts v. novices. In *Proceedings of the 7th Conference on Creativity & Cognition*, pages 351–352, 2009.

[15] B. P. Bailey and S. T. Iqbal. Understanding changes in mental workload during execution of goal-directed tasks and its application for interruption management. *ACM Transactions on Computer-Human Interaction*, 14(4):1–28, 2008.

[16] B. P. Bailey and J. A. Konstan. On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate and affective state. *Journal of Computers in Human Behavior*, 22(4):685–708, 2006.

[17] M. Bal. *Reading Rembrandt: Beyond the Word-Image Opposition*. Amsterdam University Press, 1991.

[18] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.

[19] E. A. Bier, M. C. Stone, K. Fishkin, W. Buxton, and T. Baudel. A taxonomy of see-through tools. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 358–364, New York, 1994. ACM Press.

[20] W. Blackstone. *Commentaries on the Laws of England*, volume 4. Clarendon Press, Oxford, UK, fourth edition, 1770.

[21] M. A. Boden. Computer models of creativity. In R. J. Sternberg, editor, *Handbook of Creativity*, pages 351–372. Cambridge University Press, New York, NY, USA, 1999.

[22] S. Boukhechem and E.-B. Bourennane. TLM platform based on SystemC for STAR-SoC design space exploration. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems*, pages 354–361. IEEE Computer Society, 2008.

[23] C. P. Bourne and D. F. Ford. A study of methods for systematically abbreviating English words and names. *Journal of the Association for Computing Machinery*, 8(4):538–552, 1961.

[24] G. E. P. Box and N. Draper. *Empirical Model Building and Response Surfaces*. John Wiley & Sons, New York, NY, USA, 1987.

[25] G. E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions (with discussion). *Journal of the Royal Statistical Society, Series B*, 13(1):1–45, 1951.

[26] N. Boyd and contributors. Rhino: JavaScript for Java. Retrieved May 30, 2010, from `http://www.mozilla.org/rhino/`.

[27] N. O. Brown. The birth of Athena. *Transactions and Proceedings of the American Philological Association*, 83:130–143, 1952.

[28] P. S. Brown and J. D. Gould. An experimental study of people creating spreadsheets. *ACM Transactions on Information Systems*, 5(3):258–272, 1987.

[29] J. Bruner. *Studies in Cognitive Growth*. John Wiley & Sons, New York, NY, USA, 1966.

[30] J. Bruner. *Toward a Theory of Instruction*. Harvard University Press, Cambridge, MA, USA, 1966.

[31] J. Bruner. *The Culture of Education*. Harvard University Press, 1997.

[32] C. A. Burnham and K. G. Davis. The nine-dot problem: Beyond perceptual organisation. *Psychonomic Science*, 17:321–323, 1969.

[33] V. Bush. As we may think. *Life*, September 10 1945. Substantially the same article, but without diagrams, was published in the July 1945 *Atlantic Monthly*.

[34] M. D. Byrne and S. Bovair. A working memory model of a common procedural error. *Cognitive Science*, 21(1):31–61, 1997.

[35] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. An empirical comparison of pie vs. linear menus. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 95–100, New York, 1988. ACM Press.

[36] L. Candy and E. A. Edmonds. Supporting the creative user: a criteria-based approach to interaction design. *Design Studies*, 18(2):185–194, 1997.

[37] S. K. Card, A. Newell, and T. P. Moran. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1983.

[38] E. A. Carroll, C. Latulipe, R. Fung, and M. Terry. Creativity factor evaluation: towards a standardized survey metric for creativity support. In *Proceedings of the ACM Conference on Creativity & Cognition*, pages 127–136, 2009.

[39] J. M. Carroll and W. A. Kellogg. Artifact as theory-nexus: Hermeneutics meet theory-based design. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 7–14, 1989.

[40] J. M. Carroll, W. A. Kellogg, and M. B. Rosson. The task-artifact cycle. In J. M. Carroll, editor, *Designing Interaction: Psychology at the Human-Computer Interface*, pages 74–102. Cambridge University Press, New York, NY, USA, 1991.

[41] John M. Carroll and C. Carrithers. Training wheels in a user interface. *Communications of the ACM*, 27(8):800–806, 1984.

[42] A. G. Cass, C. S. T. Fernandes, and A. Polidore. An empirical evaluation of undo mechanisms. In *Proceedings of the Nordic Conference on Human-computer Interaction*, pages 19–27, New York, NY, USA, 2006. ACM Press.

[43] C. C. Chamberlin. Arkham Investigations. Retrieved September 5, 2010, from `http://arkhaminvestigations.barkingdoginteractive.com/`.

[44] P. C.-H. Cheng. Why diagrams are (sometimes) six times easier than words: Benefits beyond locational indexing. In *Diagrammatic Representation and Inference*, Lecture Notes in Computer Science, pages 167–174, 2004.

[45] H. Christiaans and K. Dorst. Cognitive models in industrial design engineering: a protocol study. In *Design Theory and Methodology*, volume 42, pages 131–137, New York, 1992. American Society of Mechanical Engineers.

[46] W. J. Clancey. Situated action: A neuropsychological interpretation (Response to Vera and Simon). *Cognitive Science*, 17(1):87–107, 1993.

[47] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum, Hillsdale, NJ, USA, second edition, 1988.

[48] T. Coughlan and P. Johnson. Understanding productive, structural and longitudinal interactions in the design of tools for creative activities. In *Proceedings of the ACM Conference on Creativity & Cognition*, pages 155–164, New York, NY, USA, 2009. ACM Press.

[49] D. Crockford. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

[50] N. Cross and K. Dorst. Co-evolution of problem and solution spaces in creative design: observations from an empirical study. In *Computational Models of Creative Design*. University of Sydney, 1998.

[51] M. Csíkszentmihályi. *Flow: The Psychology of Optimal Experience*. Harper and Row, New York, 1990.

[52] M. Csíkszentmihályi. *Creativity: Flow and the Psychology of Discovery and Invention*. Harper Perennial, New York, 1996.

[53] M. Czerwinski, E. Cutrell, and E. Horvitz. Instant messaging and interruption: Influence of task type on performance. In *Proceedings of the Conference of the Human Factors and Ergonomics Society of Australia*, pages 356–361, 2000.

[54] F. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[55] J. E. Davidson. The suddenness of insight. In R. J. Sternberg and J. E. Davidson, editors, *The Nature of Insight*, pages 125–155. MIT Press, Cambridge, MA, USA, 1995.

[56] J. Dawes. Do data characteristics change according to the number of scale points used? An experiment using 5-point, 7-point and 10-point scales. *International Journal of Market Research*, 50(1):61—77, 2008.

[57] E. de Bono. *Lateral Thinking: Creativity Step by Step*. Harper & Row, 1970.

[58] C. Dubach, T. M. Jones, E. V. Bonilla, G. Fursin, and M. F. P. O'Boyle. Portable compiler optimisation across embedded programs and microarchitectures using machine learning. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 78–88, 2009.

[59] K. Duncker. On problem solving. *Psychological Monographs*, 58(270), 1945.

[60] D. Dutton. *The Art Instinct: Beauty, Pleasure, and Human Evolution*. Oxford University Press, Oxford, UK, 2009.

[61] D. W. Ecker. The artistic process as qualitative problem solving. *The Journal of Aesthetics and Art Criticism*, 21(3):283–290, 1963.

[62] ECMA International. ECMAScript language specification. Retrieved September 1, 2010, from `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`, 1999.

[63] A. Einstein. Induction and deduction in physics. In *The Collected Papers of Albert Einstein*, volume 7, pages 108–109. Princeton University Press, Princeton, NJ, USA, 2002. Translation by A. Engel and E. Schucking.

[64] M. Eisenberg and G. Fischer. Programmable design environments: Integrating end-user programming with domain-oriented assistance. In *Proceedings of ACM CHI 1994 Conference on Human Factors in Computing Systems*, pages 431–437, 1994.

[65] N. M. Else-Quest, J. S. Hyde, and M. C. Linn. Cross-national patterns of gender differences in mathematics: a meta-analysis. *Psychological Bulletin*, 136(1):103–27, 2010.

[66] G. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem Solving*. Academic Press, New York, 1969.

[67] J. V. A. Fine. *The Ancient Greeks: A Critical History*. Harvard University Press, Cambridge, MA, USA, 1983.

[68] G. Fischer and K. Nakakoji. Beyond the macho approach of artificial intelligence: Empower human designers—do not replace them. *Knowledge-Based Systems*, 5(1):15–30, 1992. Artificial Intelligence in Design Conference 1991 Special Issue.

[69] J. Fish and S. Scrivener. Amplifying the mind's eye: Sketching and visual cognition. *Leonardo*, 23(1):117–126, 1990.

[70] R. A. Fisher. The correlation between relatives on the supposition of Mendelian inheritance. *Transactions of the Royal Society of Edinburgh*, 52:399–433, 1918.

[71] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.

[72] G. Fitzmaurice, J. Matejka, A. Khan, M. Glueck, and G. Kurtenbach. PieCursor: Merging pointing and command selection for rapid in-place tool switching. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1361–1370, New York, NY, USA, 2008. ACM.

[73] A. W. Flaherty. *The Midnight Disease: The Drive to Write, Writer's Block, and the Creative Brain*. Houghton Mifflin, New York, NY, USA, 2004.

[74] J. A. Fodor and Z. W. Pylyshyn. How direct is visual perception?: Some reflections on Gibson's "Ecological Approach". *Cognition*, 9:139–196, 1981.

[75] J. A. Fodor and Z. W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. In S. Pinker and M. Jacques, editors, *Connections and Symbols*, pages 3–71. MIT Press, Cambridge, MA, USA, 1988.

[76] M. Fonseca, B. Barroso, P. Ribeiro, and J. Jorge. Sketch-based retrieval of ClipArt drawings. In *Proceedings of Advanced Visual Interfaces*, pages 429–432, New York, NY, USA, 2004. ACM.

[77] T. Freeth, Y. Bitsakis, X. Moussas, J. H. Seiradakis, A. Tselikas, H. Mangou, M. Zafeiropoulou, R. Hadland, D. Bate, A. Ramsey, M. Allen, A. Crawley, P. Hockley, T. Malzbender, D. Gelb, W. Ambrisco, and M. G. Edmunds. Decoding the ancient Greek astronomical calculator known as the Antikythera Mechanism. *Nature*, 444:587–591, November 30 2006.

[78] G. Fricke. Successful individual approaches in engineering design. *Research in Engineering Design*, 8(3):151–165, 1996.

[79] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3D models. *ACM Transactions on Graphics*, 22:83–105, 2003.

[80] E. Gamma, R. Helm, R. E. Johnson, and J. M. Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *Proceedings of the European Conference on Object-Oriented Programming*, pages 406–431, London, UK, 1993. Springer-Verlag.

[81] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[82] J. S. Gero. Computational models of creative designing based on situated cognition. In *Proceedings of the 4th Conference on Creativity & Cognition*, pages 3–10, 2002.

[83] J. J. Gibson. The theory of affordances. In R. Shaw and J. Bransford, editors, *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, pages 67–82. Lawrence Erlbaum, Hillsdale, NJ, USA, 1977.

[84] J. J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston, MA, USA, 1979.

[85] S. Glucksberg and J. Danks. Effects of discriminative labels and of nonsense labels upon availability of novel function. *Journal of Verbal Learning and Verbal Behaviour*, 7:72–76, 1968.

[86] S. Glucksberg and R. W. Weisberg. Verbal behaviour and problem solving: Some effects of labeling in a functional fixedness problem. *Journal of Experimental Psychology*, 71:659–664, 1966.

[87] A. Goldberg and D. Robson. *Smalltalk-80: the Language and Its Implementation*. Addison-Wesley Longman, Boston, MA, USA, 1983.

[88] G. Goldschmidt. The dialectics of sketching. *Creativity Research Journal*, 4(2):123–143, 1991.

[89] J. Gosling, B. Joy, G. Steele, and G. Bracha. *Java Language Specification*. Addison-Wesley Professional, Reading, MA, USA, third edition, 2005.

[90] W. S. Gosset (as Student). On the probable error of the mean. *Biometrika*, 6(1):1–25, 1908.

[91] W. D. Gray, B. E. John, R. Stuart, D. Lawrence, and M. E. Atwood. GOMS meets the phone company: Analytic modeling applied to real-world problems. In R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg, editors, *Readings in Human-computer Interaction: Toward the Year 2000*, pages 634–639. Morgan Kaufmann Publishers, San Francisco, CA, USA, second edition, 1995.

[92] S. Greenberg and B. Buxton. Usability evaluation considered harmful (some of the time). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 111–120, New York, NY, USA, 2008. ACM Press.

[93] J. G. Greeno and J. L. Moore. Situativity and symbols: Response to Vera and Simon. *Cognitive Science*, 17(1):49–59, 1993.

[94] I. Greif, editor. *Computer-Supported Cooperative Work: A Book of Readings*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1988.

[95] R. Grudin. *Design and Truth*. Yale University Press, New Haven, CT, USA, 2010.

[96] F. Guimbretiére and T. Winograd. FlowMenu: Combining command, text, and parameter entry. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 213–216, New York, 2000. ACM Press.

[97] J. Hadamard. *The Psychology of Invention in the Mathematical Field*. Dover Publications, 1954.

[98] E. P. Hall, S. P. Gott, and R. A. Pokorny. A procedural guide to cognitive task analysis: The PARI methodology. Technical Report AL/HR-TR-1995-0108, Brooks Air Force Base, TX, USA: Human Resources Directorate, 1995.

[99] E. Hamilton. *Mythology: Timeless Tales of Gods and Heroes*. New American Library (Mentor), Scarborough, Ontario, Canada, 1969.

[100] T. Hammond and K. Gajos. An agent-based system for capturing and indexing software design meetings. Presented at the International Workshop On Agents in Design, August 2002.

[101] M. Harada, A. Witkin, and D. Baraff. Interactive physically-based manipulation of discrete/continuous models. In *SIGGRAPH 1995*, pages 199–208, 1995.

[102] G. Harman. The inference to the best explanation. *The Philosophical Review*, 74(1):88–95, 1965.

[103] D. Harms. *The Cthulhu Mythos Encyclopedia*. Elder Signs Press, Chicago, IL, USA, third (revised) edition, 2008.

[104] J. Hartmanis and L. Berman. On polynomial time isomorphisms of some new complete sets. *Journal of Computer and System Sciences*, 16(3):418–422, 1978.

[105] Y. W. Hau and M. Khalil-Hani. SystemC-based HW/SW co-simulation platform for system-on-chip (SoC) design space exploration. *International Journal of Information and Communication Technology*, 2(1/2):108–119, 2009.

[106] J. Heisserman. Generative geometric design. *IEEE Computer Graphics and Applications*, 14(2):37–45, 1994.

[107] W. Heron. Perception as a function of retinal locus and attention. *The American Journal of Psychology*, 70(1):38–48, 1957.

[108] T. Hewett. Informing the design of computer-based environments to support creativity. *International Journal of Human-Computer Studies*, 63(4–5):383–409, 2005.

[109] T. Hewett, M. Czerwinski, M. Terry, J. Nunamaker, L. Candy, B. Kules, and E. Sylvan. Creativity support tool evaluation methods and metrics. In *Creativity Support Tools*, pages 10–24, 2005. Retrieved January 3, 2008, from `http://www.cs.umd.edu/hci/CST`.

[110] T. Hines. Left brain/right brain mythology and implications for management and training. *The Academy of Management Review*, 12(4):600–606, 1987.

[111] D. R. Hofstader. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, twentieth anniversary edition, 1999.

[112] Homer. *The Odyssey*. Johns Hopkins University Press, Baltimore, MD, USA, 2004. Translation by E. McCrorie.

[113] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing, Don Mills, ON, Canada, 1979.

[114] D. Hopkins. The design and implementation of pie menus. *Dr. Dobb's Journal*, 16(12):16–26, 1991.

[115] J. Hopson. Behavioral game design. *Gamasutra*, April 27 2001. Retrieved May 9, 2010, from `http://www.gamasutra.com/view/feature/3085/behavioral_game_design.php`.

[116] J. W. Humphrey, J. P. Oleson, and A. N. Sherwood. *Greek and Roman Technology: A Sourcebook*. Routledge, London, UK, 1997.

[117] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. In D. A. Norman and S. W. Draper, editors, *User Centered System Design: New Perspectives on Human-Computer Interaction*, pages 87–124. Erlbaum, Hillsdale, NJ, 1986.

[118] J. S. Hyde. The gender similarities hypothesis. *American Psychologist*, 60(6):581–92, 2005.

[119] H. H. S. Ip, A. K. Y. Cheng, W. Y. F. Wong, and J. Feng. Affine-invariant sketch-based retrieval of images. In *Proceedings of the International Conference on Computer Graphics*, pages 55–61, Washington, DC, USA, 2001. IEEE Computer Society.

[120] R. J. K. Jacob. New human-computer interaction techniques. In M. D. Brouwer-Janse and T. L. Harrington, editors, *Human-Machine Communication for Educational Systems Design*. Springer-Verlag, 1994.

[121] C. G. Jennings. Strange Eons: A custom component design tool for board and card games. Retrieved May 15, 2010, from `http://cgjennings.ca/eons/`.

[122] C. G. Jennings. Turing's reaction-diffusion model of morphogenesis. Retrieved May 15, 2010, from `http://cgjennings.ca/toybox/turingmorph/index.html`.

[123] C. G. Jennings and A. E. Kirkpatrick. XDS demonstration videos. Retrieved February 5, 2009, from `http://gruvi.cs.sfu.ca/videos/xds/`.

[124] C. G. Jennings and A. E. Kirkpatrick. Design as traversal and consequences: an exploration tool for experimental designs. In *Proceedings of the Graphics Interface 2007 Conference*, pages 79–86. A. K. Peters, 2007.

[125] C. G. Jennings, A. E. Kirkpatrick, and P. Mohseni. Supporting expert work processes. In P. Barker and P. van Schaik, editors, *Electronic Performance Support: Using Digital Technology to Enhance Human Ability*, pages 249–262. Gower, 2010.

[126] C. G. Jennings (editor). Strange Eons 3 user manual. Retrieved September 5, 2010, from `http://basement.cgjennings.ca/User+Manual`.

[127] B. E. John. Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 107–115, 1990.

[128] J. Johnson, T. L. Roberts, W. Verplank, D. C. Smith, C. H. Irby, M. Beard, and K. Mackey. The Xerox Star: A retrospective. *IEEE Computer*, 22(9):11–29, 1989.

[129] J. C. Jones. *Design Methods: Seeds of Human Futures*. John Wiley & Sons, New York, NY, USA, second edition, 1981.

[130] A. Kay. Doing with images makes symbols: Communicating with computers. University Video Communications. Retrieved May 12, 2010, from `http://www.archive.org/details/AlanKeyD1987/` (part 1) and `http://www.archive.org/details/AlanKeyD1987_2/` (part 2), 1987.

[131] A. Kay. User interface: A personal view. In B. Laurel, editor, *The Art of Human-Computer Interface Design*, pages 191–207. Addison-Wesley, Reading, MA, USA, 1990.

[132] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, Jyväskylä, Finland, 1997. Springer-Verlag.

[133] D. E. Kieras. Towards a practical GOMS model methodology for user interface design. In Helander, Martin, editor, *Handbook of Human-Computer Interaction*, number 7 in I. Models and Theories of Human-Computer Interaction, pages 135–157. North-Holland, New York, NY, 1988.

[134] S. King. *Night Shift*. Doubleday & Company, Garden City, NY, USA, 1978.

[135] S. King. *Skeleton Crew*. Putnam Press, New York, NY, USA, 1985.

[136] M. Kitajima and P. G. Polson. A comprehension-based model of exploration. *Human Computer Interaction*, 12(9):345–390, 1997.

[137] S. Klemmer, M. Thomsen, E. Phelps-Goodman, R. Lee, and J. A. Landay. Where do Web sites come from?: Capturing and interacting with design history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1–8, New York, 2002. ACM Press.

[138] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1995.

[139] K. Kotovsky, J. R. Hayes, and H. A. Simon. Why are some problems hard? Evidence from Tower of Hanoi. *Cognitive Psychology*, 17(2):248–294, 1985.

[140] P. O. Kristensson and S. Zhai. Command strokes with and without preview: Using pen gestures on keyboard for command selection. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1137–1146, New York, NY, USA, 2007. ACM.

[141] B. Kules. Supporting creativity with search tools. In *Creativity Support Tools*, pages 53–64, 2005. Retrieved January 3, 2008, from `http://www.cs.umd.edu/hci/CST`.

[142] G. Kurtenbach and W. Buxton. User learning and performance with marking menus. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 258–264, New York, 1994. ACM Press.

[143] J. Landay and B. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of ACM SIGCHI Conference on Human factors in Computing Systems*, pages 43–50, 1995.

[144] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, January 1964.

[145] J. Larkin, J. McDermott, D. P. Simon, and H. A. Simon. Expert and novice performance in solving physics problems. *Science*, 208(4450):1335–1342, 1980.

[146] J. Larkin and H. A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(3):65–99, 1987.

[147] K. A. Latorella. Investigating interruptions: An example from the flight deck. In *Proceedings of the Human Factors and Ergonomics Society*, pages 249–253, 1996.

[148] R. Launius and K. Wilson. *Arkham Horror*. Fantasy Flight Games, Roseville, MN, USA, second (revised) edition, 2006.

[149] B. Lawson. Cognitive strategies in architectural design. *Ergonomics*, 22(1):59–68, 1979.

[150] B. Lawson. *Design in Mind*. Architectural Press, Woburn, MA, USA, 1994.

[151] D. N. Lee, J. R. Lishman, and J. A. Thomson. Regulation of gait in long jumping. *Journal of Experimental Psychology: Human Perception and Performance*, 8:448–459, 1982.

[152] D. N. Lee, D. S. Young, P. E. Reddish, S. Lough, and T. M. H. Clayton. Visual timing in hitting an accelerating ball. *Quarterly Journal of Experimental Psychology*, 35(A):333–346, 1983.

[153] Legal burden of proof. In *Wikipedia*. Retrieved September 12, 2010, from `http://en.wikipedia.org/wiki/Legal_burden_of_proof`.

[154] G. Lemons, A. Carberry, C. Rogers, and C. Swan. Using a model-building task to compare the design process of service learning and non-service learning engineering students. In *Proceedings of the Research in Engineering Education Symposium*, 2009.

[155] R. V. Lenth. Some practical guidelines for effective sample size determination. *The American Statistician*, 55:187–193, 2001.

[156] Leonardo (da Vinci). A Study for an Equestrian Monument. Metalpoint on prepared paper. Royal Library, Windsor Castle, RL 12358r, c. 1485–90.

[157] Leonardo (da Vinci). Sketches for the Trivulzio Monument. Pen and ink. Royal Library, Windsor Castle, RL 12355, c. 1508–10.

[158] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[159] C. H. Lewis. Using the "Thinking Aloud" method in cognitive interface design. Technical Report RC-9265, IBM, 1982.

[160] Y. Liang and T. Mitra. Static analysis for fast and accurate design space exploration of caches. In *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 103–108. ACM, 2008.

[161] J. Licklider. Man-computer symbiosis. *IRE Transactions of Human Factors in Electronics*, 1(1):4–11, 1960.

[162] H. Lieberman, editor. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann, San Francisco, CA, USA, 2001.

[163] J. Lin, M. Thomsen, and J. A. Landay. A visual language for sketching large and complex interactive designs. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 307–314, 2002.

[164] P. Lloyd and P. Scott. Discovering the design problem. *Design Studies*, 15(2):125–140, 1994.

[165] P. Lloyd and P. Scott. Difference in similarity: interpreting the architectural design process. *Planning and Design*, 22(4):383–406, 1995.

[166] E. F. Loftus and J. C. Palmer. Reconstruction of automobile destruction: An example of the interaction between language and memory. *Journal of Verbal Learning and Verbal Behaviour*, 13:585–589, 1974.

[167] T. J. Lorenzen and V. L. Anderson. *Design of Experiments: A No-Name Approach*, volume 139 of *Statistics: Textbooks and Monographs*. Marcel Dekker, Inc., New York, 1993.

[168] H. P. Lovecraft. *The Fiction: Complete and Unabridged*. Library of Essential Writers. Barnes & Noble, 2008.

[169] J. Löwgren. Applying design methodology to software development. In *Proceedings of the Conference on Designing Interactive Systems*, pages 87–95, New York, NY, USA, 1995. ACM Press.

[170] S. Loyd. *Cyclopedia of Puzzles*. The Lamb Publishing Company, New York, NY, USA, 1914.

[171] É. Lucas. *Récréations Mathématiques*, volume 3. Gauther-Villars, Paris, France, 1894.

[172] É. Lucas (as N. Claus). La tour d'Hanoï: Veritable casse-tête Annamite. Instruction sheet provided with Lucas's wooden puzzle set, Paris, France, 1883.

[173] A. S. Luchins. Mechanization in problem solving: the effect of Einstellung. *Psychological Monographs*, 54(248), 1942.

[174] I. S. MacKenzie and W. Buxton. Extending Fitts' law to two-dimensional tasks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 219–226, New York, 1992. ACM Press.

[175] G. F. Marcus. Rethinking eliminative connectionism. *Cognitive Psychology*, 37(3):243–282, 1998.

[176] M. Mayer, K. Hom, and J. Wiley. Now you see it, now you don't. *The Official Google Blog*, December 2 2009. Retrieved May 21, 2010 from `http://googleblog.blogspot.com/2009/12/now-you-see-it-now-you-dont.html`.

[177] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, Part I. *Communications of the ACM*, 3(4):184–195, 1960.

[178] M. McCurdy, C. Connors, G. Pyrzak, B. Kanefsky, and A. Vera. Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1233–1242, 2006.

[179] J. McGrenere and W. Ho. Affordances: Clarifying and evolving a concept. In *Graphics Interface*, pages 179–186, Hillsdale, NJ, USA, 2000. Lawrence Erlbaum.

[180] R. Mehta and R. J. Zhu. Blue or red? Exploring the effect of color on cognitive task performances. *Science*, 323(5918):1226–9, 2009.

[181] J. J. G. van Merriënboer. *Training Complex Cognitive Skills: A Four-Component Instructional Design Model for Technical Training.* Educational Technology Publications, 1997.

[182] J. Michell. *An Introduction to the Logic of Psychological Measurement.* Lawrence Erlbaum Associates, 1990.

[183] D. W. Miller, Jr. and C. G. Miller. On evidence, medical and legal. *Journal of American Physicians and Surgeons*, 10(3):70–75, 2005.

[184] Y. Miyata and D. A. Norman. Psychological issues in support of multiple activities. In D. A. Norman and S. W. Draper, editors, *User Centered System Design: New Perspectives on Human-Computer Interaction*, pages 265–284. Erlbaum, Hillsdale, NJ, 1986.

[185] B. A. Moffitt, T. H. Bradley, D. Mavris, and D. E. Parekh. Design space exploration of small-scale PEM fuel cell long endurance aircraft. In *Proceedings of the AIAA Aviation Technology, Integration and Operations Conference*, 2006.

[186] P. Mohseni. *Treesta: A System for Supporting Statistical Analysis Using ANOVA.* Master's thesis, Simon Fraser University, Burnaby, British Columbia, Canada, 2008.

[187] D. C. Montgomery. *Design and Analysis of Experiments.* John Wiley & Sons, Inc., sixth edition, 2005.

[188] T. P. Moran and J. M. Carroll. *Design Rationale: Concepts, Techniques, and Use.* Lawrence Erlbaum Associates, 1996.

[189] Mozilla Foundation. New in JavaScript 1.6. Retrieved September 1, 2010, from `https://developer.mozilla.org/en/New_in_JavaScript_1.6`.

[190] Mozilla Foundation. New in JavaScript 1.7. Retrieved September 1, 2010, from `https://developer.mozilla.org/en/New_in_JavaScript_1.7`.

[191] B. A. Myers, S. E. Hudson, and R. Pausch. Past, present and future of user interface software tools. *ACM Transactions on Computer Human Interaction*, 7(1):3–28, 2000.

[192] K. Nakakoji, Y. Yamamoto, T. Suzuki, S. Takada, and M. Gross. Beyond critiquing: Using representational talkback to elicit design intention. *Knowledge-Based Systems Journal*, 11(7–8):457–468, 1998.

[193] K. Nakakoji, Y. Yamamoto, S. Takada, and B. Reeves. Two-dimensional spatial positioning as a means for reflection in design. In *Symposium on Designing Interactive Systems*, pages 145–154. ACM Press, 2000.

[194] A. Newell. Physical symbol systems. *Cognitive Science*, 4:135–183, 1980.

[195] A. Newell, J. C. Shaw, and H. A. Simon. Elements of a theory of human problem solving. *Psychological Review*, 65(3):151–166, 1958.

[196] A. Newell and H. A. Simon. *Human problem solving*. Prentice-Hall, 1972.

[197] A. Newell and H. A. Simon. Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19:113–126, 1976.

[198] L. F. Niklasson and T. van Gelder. On being systematically connectionist. *Mind & Language*, 9(3), 1994.

[199] H. Noguchi. How do material constraints affect design creativity? In *Proceedings of the 3rd Conference on Creativity & Cognition*, pages 82–87, New York, NY, USA, 1999. ACM.

[200] D. A. Norman. Design rules based on analyses of human error. *Communications of the ACM*, 26(4):254–258, 1983.

[201] D. A. Norman. *The Design of Everyday Things*. Basic Books, second edition, 2002.

[202] D. A. Norman and S. W. Draper, editors. *User Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, 1986.

[203] Donald A. Norman. *Emotional Design: Why We Love (or Hate) Everyday Things*. Basic Books, 2005.

[204] T. C. Ormerod. Planning and ill-defined problems. In R. Morris and G. Ward, editors, *The Cognitive Psychology of Planning*, pages 53–70. Psychology Press, London, 2005.

[205] A. Oulasvirta and P. Saariluoma. Long-term working memory and interrupting messages in human-computer interaction. *Behaviour & Information Technology*, 23(1):53–64, 2004.

[206] S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, 1980.

[207] D. N. Perkins. *The Mind's Best Work*. Harvard University Press, Cambridge, USA, 1981.

[208] K. Perlin and D. Fox. Pad—An alternative approach to the computer interface. In *SIGGRAPH*, pages 57–64, 1993.

[209] C. Petzold. *The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine*. Wiley Publishing, New York, NY, USA, 2008.

[210] L. Philips. The Double Metaphone search algorithm. *C/C++ Users Journal*, 18(6):38–43, 2000.

[211] J. Piaget. Piaget's theory. In P. H. Mussen, editor, *Carmichael's Manual of Child Psychology*, volume 1, pages 703–723. John Wiley & Sons, New York, NY, USA, third edition, 1970.

[212] H. Poinecaré. *The Foundations of Science*. Science House, New York, USA, 1929.

[213] K. Popper. *The Logic of Scientific Discovery*. Basic Books, New York, NY, USA, 1959.

[214] Z. W. Pylyshyn. *Computation and Cognition. Toward a Foundation for Cognitive Science*. MIT Press, Cambridge, MA, USA, 1984.

[215] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41:1–28, 2005.

[216] S. K. Reed. A structure-mapping model for word problems. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 13(1):125–139, 1987.

[217] S. K. Reed, A. Dempster, and M. Ettinger. Usefulness of analogous solutions for solving algebra word problems. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 11(1):106–125, 1985.

[218] D. Reisberg. External representations and the advantages of externalizing one's thoughts. In *The Ninth Annual Conference of the Cognitive Science Society*, pages 281–293, Hillsdale, NJ, USA, 1987. Lawrence Erlbaum Associates.

[219] W. R. Reitman. *Cognition and Thought*. John Wiley & Sons, New York, NY, USA, 1965.

[220] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, and M. Eisenberg. Design principles for tools to support creative thinking. In *Creativity Support Tools*, pages 25–36, 2005. Retrieved January 3, 2008, from `http://www.cs.umd.edu/hci/CST`.

[221] M. Resnick and B. Silverman. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 Conference on Interaction Design and Children*, pages 117–122. ACM Press, 2005.

[222] H. Rittel and M. Webber. Dilemmas in a general theory of planning. *Policy Sciences*, 4:155–169, 1973.

[223] S. D. Sala, editor. *Mind Myths: Exploring Popular Assumptions about the Mind and Brain*. Wiley Publishing, New York, NY, USA, 1999.

[224] W. A. Sandoval. Developing learning theory by refining conjectures embodied in educational designs. *Educational Psychologist*, 39(4):213–223, 2004.

[225] D. L. Schacter. Memory distortion: History and current status. In D. L. Schacter, editor, *Memory Distortion: How Minds, Brains, and Societies Reconstruct the Past*, pages 1–43, Cambridge, MA, USA, 1995. Harvard University Press.

[226] M. Scheerer. Problem solving. *Scientific American*, 208(4):118–128, 1963.

[227] U. Schmid, J. Wirth, and K. Polkehn. Analogical transfer of non-isomorphic source problems. In *Proceedings of the Conference of the Cognitive Science Society*, pages 631–636. Morgan Kaufmann, 1999.

[228] D. A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, 1983.

[229] D. A. Schön and G. Wiggins. Kinds of seeing and their functions in designing. *Design Studies*, 13(2):135–156, 1992.

[230] A. A. Schwartz. Board games offer refuge for families in recession. *National Post*, February 13 2009.

[231] T. Selker. Fostering motivation and creativity for computer users. *International Journal of Human-Computer Studies*, 63(4–5):410–421, 2005.

[232] H. Shin and T. Igarashi. Magic Canvas: Interactive design of a 3-D scene prototype from freehand sketches. In *Proceedings of Graphics Interface 2007*, pages 63–70. A. K. Peters, 2007.

[233] F. M. Shipman and H. Hseih. Navigable history: A reader's view of writer's time. *The New Review of Hypermedia and Multimedia*, 6(1):147–167, 2000.

[234] B. Shneiderman. Creating creativity: User interfaces for supporting innovation. *ACM Transactions on Computer Human Interaction*, 7(1):114–138, 2000.

[235] B. Shneiderman. Creativity support tools. *Communications of the ACM*, 12(50):20–32, 2007.

[236] S. Sick, S. Reinker, J. Timmer, and T. Schlake. WNT and DKK determine hair follicle spacing through a reaction-diffusion mechanism. *Science*, 314(5804):1447–1450, 2006.

[237] M. Smith. *The Ancient Greeks*. Cornell University Press, Ithaca, NY, USA, 1960.

[238] S. M. Smith. Getting into and out of mental ruts: A theory of fixation, incubation, and insight. In R. J. Sternberg and J. E. Davidson, editors, *The Nature of Insight*, pages 229–251. MIT Press, Cambridge, USA, 1995.

[239] A. Sokal and J. Bricmont. *Fashionable Nonsense: Postmodern Intellectuals' Abuse of Science*. Picador, New York, NY, USA, 1998.

[240] M. W. van Someren, Y. F. Barnard, and J. A. C. Sandberg. *The Think Aloud Method: A Practical Guide to Modelling Cognitive Processes*. Academic Press, London, 1994.

[241] G. L. Steele, Jr. *Common LISP: the Language*. Digital Press, Newton, MA, USA, second edition, 1990.

[242] R. J. Sternberg and T. I. Lubart. An investment theory of creativity and its development. *Human Development*, 34:1–32, 1991.

[243] R. J. Sternberg and T. I. Lubart. The concept of creativity: Prospects and paradigms. In R. J. Sternberg, editor, *Handbook of Creativity*, pages 3–15. Cambridge University Press, New York, NY, USA, 1999.

[244] George Stiny. Introduction to shape and shape grammars. *Environment and Planning B: Planning and Design*, 7(3):343–352, 1980.

[245] G. M. Stump, M. Yukish, T. W. Simpson, and E. N. Harris. Design space visualization and its application to a design by shopping paradigm. In *Proceedings of ASME 2002 Design Engineering Technical Conferences*, volume 2, pages 795–804, 2003.

[246] L. A. Suchman. *Plans and Situated Action: The Problem of Human-Machine Interaction*. Cambridge University Press, Cambridge, England, 1987.

[247] L. A. Suchman. Response to Vera and Simon's situated action: A symbolic interpretation. *Cognitive Science*, 17(1):71–75, 1993.

[248] C. Sun. Undo as concurrent inverse in group editors. *ACM Transactions on Computer-Human Interaction*, 9(4):309–361, 2002.

[249] A. G. Sutcliffe. On the effective use and reuse of HCI knowledge. *ACM Transactions on Computer-Human Interaction*, 7(2):197–221, 2000.

[250] A. G. Sutcliffe and J. M. Carroll. Designing claims for reuse in interactive systems design. *International Journal of Human-Computer Studies*, 50:213–241, 1999.

[251] I. E. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System*. Garland Publishers, New York, 1980.

[252] M. Suwa, J. S. Gero, and T. Purcell. The roles of sketches in early conceptual design processes. In *Twentieth Annual Meeting of the Cognitive Science Society*, pages 1043–1048. Lawrence Erlbaum Associates, 1998.

[253] D. S. Tan, D. Gergle, P. Scupelli, and R. Pausch. Physically large displays improve performance on spatial tasks. *ACM Transactions on Computer-Human Interaction*, 13(1):71–99, 2006.

[254] M. Terry, E. D. Mynatt, K. Nakakoji, and Y. Yamamoto. Variation in element and action: Simultaneous development of alternative solutions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 711–718, New York, 2004. ACM Press.

[255] P. Thagard. The best explanation: Criteria for theory choice. *Journal of Philosophy*, 75(2):76–92, 1978.

[256] P. Thagard. Explanatory coherence. *Behavioral and Brain Sciences*, 12:435–502, 1989.

[257] P. Thagard. *How Scientists Explain Disease*. Princeton University Press, Princeton, NJ, USA, 2000.

[258] P. Thagard. Coherence, truth, and the development of scientific knowledge. *Philosophy of Science*, 74(1):28–47, 2007.

[259] J. C. Thomas and J. M. Carroll. The psychological study of design. *Design Studies*, 1(1):5–11, 1979.

[260] F. Tian, L. Xu, H. Wang, X. Zhang, Y. Liu, V. Setlur, and G. Dai. Tilt Menu: Using the 3D orientation information of pen devices to extend the selection capability of pen-based user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1371–1380, New York, NY, USA, 2008. ACM.

[261] J. Travers. *The Puzzle-Mine: Puzzles Collected from the Works of the Late Henry Ernest Dudeney*. Thomas Nelson & Sons, London, UK, 1951.

[262] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society. Series 2*, 42:230–265, 1936.

[263] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37—72, 1952.

[264] S. Turkle. *The Second Self: Computers and the Human Spirit*. Simon and Schuster, New York, 1984.

[265] S. Turkle and S. Papert. Epistemological pluralism and the revaluation of the concrete. *Journal of Mathematical Behavior*, 11(1):3–33, 1992.

[266] D. Ungar and R. B. Smith. Self: The power of simplicity. *SIGPLAN Notices*, 22(12):227–242, 1987.

[267] P. Van Tornout, J. Pelgroms, and J. van der Meeren. Sweetness evaluation of mixtures of fructose with saccharin, aspartame or Acesulfame K. *Journal of Food Science*, 50(2):469–472, March 1985.

[268] A. H. Vera and H. A. Simon. Situated action: A symbolic interpretation. *Cognitive Science*, 17(1):7–48, 1993.

[269] A. H. Vera and H. A. Simon. Situated action: Reply to reviewers. *Cognitive Science*, 17(1):77–86, 1993.

[270] A. H. Vera and H. A. Simon. Situated action: Reply to William Clancy. *Cognitive Science*, 17(1):117–133, 1993.

[271] R. W. Weisberg. *Creativity: Beyond the Myth of Genius*. Freeman, New York, NY, USA, 1993.

[272] R. W. Weisberg and J. W. Alba. An examination of the alleged role of "fixation" in the solution of several "insight" problems. *Journal of Experimental Psychology: General*, 110:169–192, 1981.

[273] A. Wells. Situated action, symbol systems and universal computation. *Minds and Machines*, 6(1):33–46, 1996.

[274] W. A. Wickelgren. *How to Solve Problems*. Freeman, San Francisco, USA, 1974.

[275] T. V. Wolf, J. A. Rode, J. B. Sussman, and W. A. Kellogg. Dispelling "design" as the black art of CHI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, volume 1, pages 521–530, New York, NY, USA, 2006. ACM Press.

[276] R. Woodbury, S. Datta, and A. Burrow. Erasure in design space exploration. In *Artificial Intelligence in Design 2000*, pages 521–544, Dordrecht, Netherlands, 2000. Kluwer Academic Publishers.

[277] R. F. Woodbury and A. L. Burrow. Whither design space? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20(2):63–82, 2006.

[278] Y. Yamamoto and K. Nakakoji. Interaction design of tools for fostering creativity in the early stages of information design. *International Journal of Human-Computer Studies*, 63(4–5):513–535, 2005.

[279] Y. Yamamoto, S. Takada, M. D. Gross, and K. Nakakoji. Representational talkback: An approach to support writing as design. In *Proceedings of the Asia Pacific Computer Human Interaction Conference*, pages 125–131. IEEE Computer Society, 1998.

[280] A. Yazici and R. George. *Fuzzy Database Modeling*. Physica-Verlag, Heidelberg, Germany, 1999.

[281] R. Yeo. Before Memex: Robert Hooke, John Locke, and Vannevar Bush on external memory. *Science in Context*, 20(1):21–47, 2007.

[282] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[283] J. Zhang. The nature of external representations in problem solving. *Cognitive Science*, 21(2):179–217, 1997.