

Task-Centric User Interfaces

by

Benjamin J. Lafreniere

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2014

© Benjamin J. Lafreniere 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Software applications for design and creation typically contain hundreds or thousands of commands, which collectively give users enormous expressive power. Unfortunately, rich feature sets also take a toll on usability. Current interfaces to feature-rich software address this dilemma by adopting menus, toolbars, and other hierarchical schemes to organize functionality—approaches that enable efficient navigation to specific commands and features, but do little to reveal how to perform unfamiliar *tasks*.

We present an alternative *task-centric user interface* design that explicitly supports users in performing unfamiliar tasks. A task-centric interface is able to quickly adapt itself to the user’s intended goal, presenting relevant functionality and required procedures in task-specific customized interfaces. To achieve this, task-centric interfaces (1) represent tasks as first-class objects in the interface; (2) allow the user to declare their intended goal (or infer it from the user’s actions); (3) restructure the interface to provide step-by-step scaffolding for the current goal; and (4) provide additional knowledge and guidance within the application’s interface.

Our inspiration for task-centric interfaces comes from a study we conducted, which revealed that a valid use case for feature-rich software is to perform short, targeted tasks that use a small fraction of the application’s full functionality. Task-centric interfaces provide explicit support for this use.

We developed and tested our task-centric interface approach by creating *AdaptableGIMP*, a modified version of the GIMP image editor, and *Workflows*, an iteration on AdaptableGIMP’s design based on insights from a semi-structured interview study and a think-aloud study.

Based on a two-session study of Workflows, we show that task-centric interfaces can successfully support a *guided-and-constrained* problem solving strategy for performing unfamiliar tasks, which enables faster task completion and reduced cognitive load as compared to current practices.

We also provide evidence that task-centric interfaces can enable a higher-level form of application learning, in which the user associates tasks with relevant keywords, as opposed to low-level commands and procedures. This *keyword learning* has potential benefits for memorability, because the keywords themselves are descriptive of the task being learned, and scalability, because a few keywords can map to an arbitrarily complex set of commands and procedures.

Finally, our findings suggest a range of different ways that the idea of task-centric interfaces could be further developed.

Acknowledgements

I feel incredibly lucky to have joined the Human-Computer Interaction lab only three years after its inception, and to have been a part of its growth into the thriving lab it is today. In particular, I want to thank Michael Terry, Edward Lank, Daniel Vogel, and the many other lab members who have together created an ideal place to pursue a PhD.

I want to thank Mike for his guidance and supervision, and for being a tireless champion for myself and my research. I can't think of a better mentor and role model to have had these past years.

I want to thank my committee members: Edward Lank, Daniel Vogel, and Parmit Chilana for providing invaluable advice and guidance, and my external examiner Carl Gutwin for asking tough questions and providing inspiration for future research.

All of my coauthors and collaborators have my sincere thanks. In particular, I want to thank Andrea Bunt, who was a collaborator on all of the projects reported in this dissertation, and has also been a constant source of encouragement and good advice. In addition to Mike and Andrea, I want to thank everyone who contributed the AdaptableGIMP project, including Adam Fourney, Filip Krynicki, Leslie Ng, Matthew Lount, and Jordan Stinson.

In the HCI lab there are more people than I can easily name that have made these past years a memorable and enjoyable experience. I want to thank Adam Fourney, who has always been up for discussing research, and without fail has provided good suggestions; and Matthew Kay, who graduated soon after I started, but has had a big influence on my direction in HCI research.

I want to acknowledge the tireless efforts of the department and university's administrative staff, and Wendy Rush in particular, for helping me with booking rooms, filing reimbursements, and the many other individually small but collectively huge tasks it takes to complete a PhD.

Outside of the university, I owe a debt of gratitude to the great people at Autodesk Research, where I did two summer internships. In particular, I want to thank Tovi Grossman, Justin Matejka, and George Fitzmaurice for giving me an alternative perspective on how to do research.

Finally, I want to thank my family and friends, who have always been supportive of me. I particularly want to thank Valerie Busch, who has been a pillar of support when things got the craziest, and Max (the cat) who directly oversaw the writing of most of this thesis from a chair beside mine (though he was asleep most of the time).

Over the course of my PhD I received financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Graphics, Animation and New Media Network of Centres of Excellence (GRAND NCE), and the University of Waterloo, to all of whom I am grateful.

Parts of Chapter 3 of this dissertation have been published in the proceedings of the Canadian Information Processing Society's Graphics Interface (GI) 2010 conference.

Table of Contents

Author's Declaration.....	ii
Abstract.....	iii
Acknowledgements.....	iv
Table of Contents.....	vi
List of Figures	xi
List of Tables	xii
Chapter 1 Introduction	1
1.1 Task-Centric Interfaces	2
1.2 Workflows – A Prototype Task-Centric Interface	3
1.3 Thesis Statement and Research Questions.....	5
1.4 Research Contributions.....	6
1.5 Research Scope	7
1.5.1 Non-Expert, Occasional Users.....	7
1.5.2 Unfamiliar Tasks.....	8
1.5.3 Tasks Involving Ill-Defined Problem Solving	8
1.5.4 Evaluating an Alternative Problem-Solving Process	9
1.5.5 Simplifying Assumptions.....	10
1.6 Summary and Outline	10
1.7 Terminology.....	13
Chapter 2 Background Literature.....	14
2.1 Learning and Using Feature-Rich Software.....	14
2.1.1 Large-Scale Analyses of Use of Feature-Rich Software.....	14
2.1.2 The Impact of Rich Feature Sets.....	15
2.1.3 Learning to Use Feature-Rich Software.....	16
2.1.4 Problem-Solving Strategies.....	17
2.1.5 The Challenge Posed by Feature-Rich Software	18
2.2 Support Techniques for Feature-Rich Software.....	19
2.2.1 Interface Personalization.....	19
2.2.2 Task Automation.....	21
2.2.3 In-Application Tutoring Systems.....	22
2.2.4 The Need for Techniques to Support Unfamiliar Tasks	23

Chapter 3 Characterizing Large-Scale Use of a Direct-Manipulation Application in the Wild.....	25
3.1 The <i>ingimp</i> Dataset	26
3.1.1 <i>ingimp</i> Deployment and Distribution.....	27
3.2 The <i>ingimp</i> User Base	27
3.2.1 Defining Significant Users	28
3.3 Characterizing Users' Sessions	28
3.4 Characterizing Users' Documents.....	29
3.5 Community Command Usage	29
3.5.1 Command Statistics for Sessions.....	30
3.5.2 Command Coverage and Command Vocabularies.....	31
3.5.3 Overlap in Command Vocabularies	32
3.6 Characterizing Users' High-Level Tasks	32
3.6.1 Clustering Sessions.....	33
3.6.2 Activity Tag Keywords for Clustered Sessions.....	35
3.7 The Typical <i>ingimp</i> User.....	37
Chapter 4 Task-Centric Interfaces.....	38
4.1 Feature-Centric Interfaces	38
4.2 Task-Centric Interfaces	39
4.3 Components of a Task-Centric Interface.....	39
4.4 Comparison with Existing Personalizable Interface Approaches.....	41
4.5 Comparison with the Ribbon and Perspectives	43
4.6 Comparison with Web-Based Tutorials	45
4.7 Our Investigation of Task-Centric Interfaces	46
4.8 Summary	46
Chapter 5 AdaptableGIMP	48
5.1 AdaptableGIMP's Task-Centric Interface.....	48
5.2 Semi-Structured Interview Study	52
5.2.1 Author's Note on Contribution.....	52
5.2.2 Method.....	53
5.2.3 High-level Reactions	54
5.2.4 Task-Centric Interface Utility.....	54
5.2.5 Task-Set Documentation	55

5.2.6 Searching for Task Sets.....	56
5.2.7 Summary of Findings.....	56
5.3 Think-Aloud Study	57
5.3.1 Method	57
5.3.2 Results.....	57
5.4 Summary	60
5.4.1 Contributors to the AdaptableGIMP Project.....	60
Chapter 6 Workflows.....	62
6.1 From Task Sets to Workflows	63
6.2 Workflows' Task-Centric Interface	64
6.3 Creating Customizations.....	66
6.4 Implementation	66
6.4.1 User Interface and Command Invocation	66
6.4.2 Python Server Component	67
6.4.3 Workflows Search Engine	67
6.5 Summary	67
Chapter 7 Assessing the Impact of a Task-Centric Interface.....	68
7.1 Experimental Design.....	69
7.1.1 Basic Study Design.....	69
7.1.2 Method	70
7.1.3 Tasks and Workflow Authoring.....	71
7.1.4 Participants.....	73
7.2 Research Questions.....	73
7.3 Results.....	74
7.3.1 Analysis Notes	74
7.3.2 Task Times and Cognitive Load	75
7.3.3 Problem-Solving Strategies.....	77
7.3.4 Learning in Session 1	85
7.3.5 Participant Reactions.....	86
7.4 Summary	91
7.4.1 Advantages.....	91
7.4.2 Disadvantages	92

7.4.3 Research Questions Revisited	92
Chapter 8 Discussion.....	94
8.1 Implications for Task-Centric Interfaces.....	94
8.2 Learning in Task-Centric Interfaces.....	95
8.2.1 Design Implication – Exploring the Space of Learning Tradeoffs.....	96
8.3 Keyword Learning.....	96
8.3.1 Design Implication – Proactive Assistance with Terminology	98
8.4 Guiding Users through the Space of Solutions.....	98
8.4.1 Design Implication – Supporting Command Settings	99
8.4.2 Design Implication – Guidelines for Weaving Task Knowledge into Interfaces	99
8.5 Use of Feature-Rich Software for Short, Targeted Tasks	100
8.6 Summary	102
Chapter 9 Impact and Opportunities for Future Work	103
9.1 Conclusions	104
9.2 Opportunities for Future Work.....	105
9.2.1 Understanding User Behavior in Feature-Rich Software	105
9.2.2 Usability at the Task Level.....	107
9.2.3 Exploring the Space of Task-Centric Interface Designs.....	108
9.2.4 Toolkit Support.....	110
9.2.5 Building a Comprehensive Library of Task-Centric Customizations	111
9.2.6 Applications for Task-Centric Interfaces	113
9.3 Summary	113
Appendix A Task-Centric Community Customization	115
A.1 Motivation and Related Work	115
A.1.1 The Social Practices of Customization.....	115
A.1.2 Community-Generated Documentation.....	116
A.2 Community Customization in AdaptableGIMP	117
A.2.1 Creating and Sharing Customizations	117
A.2.2 Community Refinement of Customizations	118
A.3 Feedback on Task-Centric Community Customization.....	122
A.4 Deployment and Response	124
A.5 Summary and Discussion	125

Appendix B Study Materials.....	127
B.1 Pre-Task Questionnaire.....	127
B.2 Post-Study Questionnaire	127
B.3 Workflows.....	128
References.....	134

List of Figures

Figure 1. The basic components of a task-centric interface in Workflows	4
Figure 2. Research path showing research problems, activities, and main results.....	11
Figure 3. Histogram showing active usage time for sessions.....	29
Figure 4. Histogram showing the number of command invocations per session	31
Figure 5. Conceptual comparison of task-centric interfaces to existing personalization approaches ..	42
Figure 6. The Ribbon interface from Microsoft PowerPoint 2007.....	43
Figure 7. Conceptual comparison of task-centric interfaces with Ribbon and Perspectives.....	44
Figure 8. Design space of task-support and learning techniques.....	46
Figure 9. The task-centric interface provided by AdaptableGIMP.	49
Figure 10. Task set documentation on the AdaptableGIMP wiki	50
Figure 11. The basic components of a task-centric interface in Workflows	62
Figure 12. Comparison of AdaptableGIMP and Workflows.....	64
Figure 13. Before/after images for the four study tasks performed by participants.....	70
Figure 14. Average per-participant task times.....	75
Figure 15. Average cognitive load for the six axes of the NASA TLX by condition and session.....	76
Figure 16. Self-rated preferences for the two study conditions.....	87
Figure 17. Creating a task set in AdaptableGIMP.....	117
Figure 18. Change tracking for a task set stored on the AdaptableGIMP wiki.....	120

List of Tables

Table 1. The number of observed commands shared by different proportions of users.....	32
Table 2. Task sets for the seven clusters of sessions.....	34
Table 3. Frequently occurring activity tag keywords for each cluster.....	36
Table 4. Participant responses to 7-point Likert-scale statements	54
Table 5. Templates used to create workflows based on procedures found in web tutorials.....	72
Table 6. Research questions and corresponding measures.....	73
Table 7. Summary of qualitative codes for the two study conditions.....	78
Table 8. Research questions and the answers we found for them in our study.....	93

Chapter 1

Introduction

Feature-rich software applications for design and creation are an essential part of modern computing. Adobe Photoshop, with over five-hundred menu items, is the de-facto standard tool of the graphic design industry; Microsoft Word, with over a thousand commands and options, is a near-universal fixture across the business world; and Autodesk’s AutoCAD, with over a thousand commands, is used to design everything from teapots to skyscrapers. In a very literal way, these applications shape the world that we live in.

The feature-richness of these applications is a driver of their enduring success. A feature set consisting of hundreds or thousands of commands supports an exponentially larger set of *tasks*—activities undertaken to achieve specific desired goals. Supporting a wide range of related tasks in one place gives these applications enormous expressive power and broad appeal. Large feature sets also have potential advantages for usability: a given tool can be used for many different tasks, and the software’s interface can establish conventions that are re-used throughout the interface, which allows a user to transfer their knowledge across related tasks.

Unfortunately, rich feature sets also add significant complexity to an application’s interface, and this takes a toll on usability. Previous work has shown that users respond to feature-rich applications by learning a small, and personal, subset of functionality that is relevant to their individual needs (Draper 1984; Greenberg 1993; Linton et al. 2000; Matejka et al. 2009; Sutcliffe and Old 1987), and that once they have learned such a subset, they are hesitant to spend additional time on concerns, such as learning, that are unrelated to their primary tasks (Carroll and Rosson 1987; Carroll 1990; Mackay 1991; Rieman 1996). Irrespective of expertise, many users find feature-rich applications to be overwhelming, frustrating, and difficult to navigate (McGrenere and Moore 2000), and excess interface complexity can have a negative impact on performance, particularly for novice users (Cockburn, Gutwin, and Greenberg 2007).

Our view is that these disadvantages are not innate traits of feature-rich software, but are a result of current *feature-centric* approaches to organizing functionality, in which similar commands are grouped into menus, submenus, toolbars, palettes, and tabs. This organizational scheme allows the interface to fit a large number of commands, and aids in navigating to specific commands, but it does not reveal how to perform unfamiliar tasks, which may require commands found scattered *throughout*

the application. As a result, the user wishing to perform an unfamiliar task is forced to (1) *identify relevant functionality* by inefficiently hunting through the application’s interface, and (2) *determine the required procedures to reach their intended goal* through error-prone trial-and-error and experimentation. In principle, the user could turn to external help resources for assistance, but past work has shown that users are typically hesitant to do so (Carroll and Rosson 1987; Rieman 1996; Andrade, Bean, and Novick 2009; Rettig 1991).

In this dissertation, we introduce the notion of *task-centric user interfaces* for feature-rich software that adapt “on the fly” to support specific tasks. By doing so, a task-centric interface can provide explicit support and guidance within the application for performing unfamiliar tasks.

In this chapter, we first describe task-centric interfaces and the prototype application that we have developed to investigate this idea. We then present a high-level summary of the research questions we set out to answer, and the contributions that we make in this dissertation. Next, we step back and define the scope of our investigation in greater detail, and provide a chapter-by-chapter summary of the research to be presented in this dissertation.

1.1 Task-Centric Interfaces

A task-centric user interface is able to quickly adapt itself to support the user in performing specific intended tasks. By adapting the interface to the user’s intended task, the application reveals the relevant functionality and the required procedures to reach the desired goal in an interface that can be directly interacted with to perform the required actions.

In this dissertation, we introduce the concept of a task-centric interface and define four important properties of this style of user interface. Specifically, task-centric interfaces:

- **Represent tasks as first-class objects** that associate a task goal with relevant commands and functionality, and the procedures required to reach the goal;
- **Can be made aware of the user’s intended task**, either through explicit *task declaration* by the user, or implicit *task inference* by the system;
- **Include a task adaptation mechanism** that restructures the interface to provide explicit, step-by-step scaffolding for task completion; and
- **Provide task knowledge in the interface**, to help users understand how commands and features relate to achieving specific, meaningful goals in the application.

The choice of specific means to satisfying these four properties creates a design space for task-centric user interfaces.

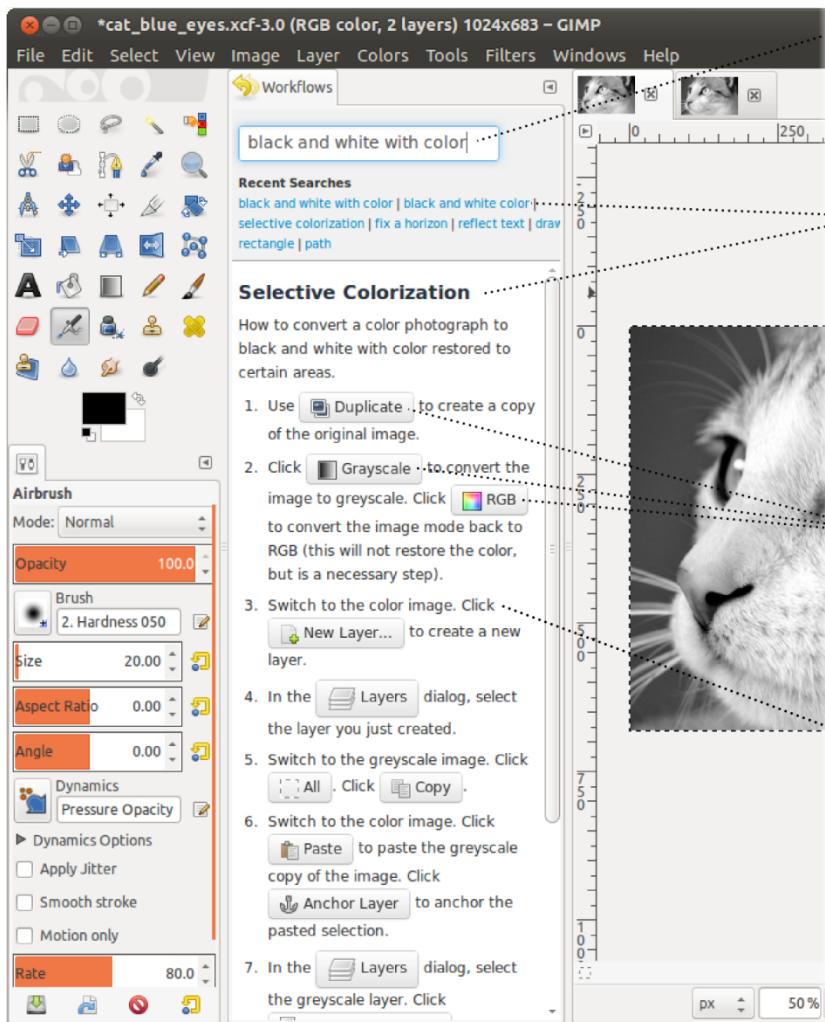
Like adaptable and adaptive user interfaces, task-centric interfaces use customization to provide the user with a simplified interface. Unlike these approaches, the focus is on using customization to assist the user in completing *specific* tasks. One way of thinking of task-centric interfaces is that they take the idea behind the Ribbon interface in Microsoft Office, or Perspectives in the Eclipse IDE, to a logical extreme; instead of using customization to support large classes of related tasks, task-centric interfaces use customization to provide step-by-step scaffolding for achieving individual, specific goals in an application.

To investigate the potential impact of a task-centric interface design, we designed and evaluated *AdaptableGIMP* (Laffreniere et al. 2011) and *Workflows*, two modified versions of the GNU Image Manipulation Program (GIMP).

AdaptableGIMP provided a task-centric interface, and additionally sought to address how a community of users of an application could collectively create a body of task-centric knowledge around an application. *Workflows* was created as an iteration on *AdaptableGIMP*'s design, to further investigate the design of task-centric interfaces.

1.2 Workflows – A Prototype Task-Centric Interface

Workflows supports an interaction in which the user enters keywords describing their intended goal (e.g. “black and white with color”), and the application responds with interface customizations that match the declared task (Figure 1). These interface customizations, which we call *workflows*, include step-by-step instructions for how to complete the task, embedded with *actionable* references to commands—the buttons can be clicked to directly invoke commands from within the workflow itself, without having to locate them in the full interface. This supports a problem-solving strategy for unfamiliar task that starts with declaring the intended task, and then using the commands and guidance provided in the resultant streamlined interface to perform the task.



Task Declaration

The user declares their intended task to the system using keyword search.

Tasks as First-Class Objects

The system responds with workflows, interface customizations tailored for performing specific tasks.

A list of Recent Searches allows the user to quickly return to previously-accessed workflows.

Task Adaptation

workflows act as adaptations of the interface to the user's task, providing an organized set of relevant commands.

Task Knowledge

workflows also provide step-by-step instructions on how to use the included functionality to perform the task.

Figure 1. The basic components of a task-centric interface in Workflows.

Using keyword search as a task-declaration mechanism allows the user to express their intended goal in their own terminology, with the system producing workflows that it believes to be most relevant. Keyword search also naturally handles providing access to a large number of workflows, which is important because the space of potential tasks that a user might wish to perform could be very large.

Task adaptation occurs when a workflow is loaded in response to a search. Because the commands included in the workflow can be directly invoked, the workflow acts as a task-specific customization of the application's interface, organizing its low-level functionality for achieving a particular goal. At

the same time, the workflow provides step-by-step guidance on how to use the included commands to reach that goal.

The design of Workflows was developed over a number of iterations, starting with AdaptableGIMP, an earlier task-centric interface prototype that we developed.

1.3 Thesis Statement and Research Questions

Our investigation of task-centric interfaces is structured around the following thesis statement:

A task-centric interface is able to assist the user with performing unfamiliar tasks by providing explicit support for locating relevant functionality, and determining the procedures required to reach an intended goal. The expected advantages of a task-centric interface are improved task performance, reduced cognitive load, and enabling the user to learn and use the application at the higher conceptual level of tasks, instead of individual commands and features.

To defend this thesis statement, the work presented in this dissertation addresses the following research questions:

- How are users currently using feature-rich software?
- How does a task-centric interface affect the problem-solving strategies used to complete unfamiliar tasks in feature-rich software?
- How does a task-centric interface affect performance measures such as task time and cognitive load when performing unfamiliar tasks?
- What are the relative advantages and disadvantages of a task-centric interface over existing feature-centric interfaces for feature-rich software?
- How do task-centric interfaces change learning in feature-rich software?
- What are users' subjective reactions to a task-centric interface design?
- What are the important components of a task-centric interface design?

Next, we provide a summary of the specific research contributions that we have made through the work presented in this dissertation. Later in this chapter we summarize the specific research activities we carried out to answer these questions.

1.4 Research Contributions

In this dissertation, we make the following research contributions:

1. Based on a study of two-years' worth of usage data gathered from a publicly-deployed, instrumented image editor, we provide evidence that a valid use case for feature-rich software is to perform short, targeted tasks using a small fraction of the available functionality.
2. We also replicate previous findings that individual users' command vocabularies tend to be small and idiosyncratic, and that command frequencies follow a long-tailed distribution, when considered across a community of users.
3. We argue that current *feature-centric* paradigms for organizing functionality in feature-rich software are poorly suited to the type of use that we have identified above, because they do not support users in performing unfamiliar tasks.
4. To explicitly support this scenario of use, we present the concept of a *task-centric interface*—an alternative interface paradigm for feature-rich software that is able to quickly adapt itself as needed to support the user in performing specific tasks.
5. We present AdaptableGIMP, an initial task-centric interface design that we developed. Based on two user studies of AdaptableGIMP, we present evidence that users appreciate the features of a task-centric interface, and derive insights into how the design can be improved.
6. We present Workflows, an iteration on the task-centric interface design in AdaptableGIMP, and the main task-centric interface prototype that we have developed to evaluate the potential of task-centric interfaces.
7. Based on the results of a laboratory study held over two sessions, we find evidence that a task-centric interface design can provide advantages over current practices for using feature-rich software to perform unfamiliar tasks. In particular, we present evidence that:
 - a. A task-centric interface design can enable users to complete tasks significantly faster and with reduced cognitive load, both when performing an unfamiliar task for the first time, and re-performing that task at least two weeks later.
 - b. Users express a subjective preference for the task-centric interface design over current practices.

8. We present evidence for three main mechanisms by which task-centric interfaces provide the benefits listed above:
 - a. The task adaptation provided by the task-centric interface helps users to locate relevant functionality for a task.
 - b. The task knowledge provided by the task-centric interface helps users to form a plan for how to achieve their intended goal.
 - c. By explicitly supporting the process of performing an unfamiliar task, the task-centric interface reduces self-directed exploration of the interface, which is a common source of frustration and errors.
9. We present evidence that a task declaration mechanism based on keyword search allows users to engage in *keyword learning* to efficiently return to task-centric help resources. This enables the user to learn the interface at a higher conceptual level of tasks, rather than learning lower-level functionality and procedures. Keyword learning also has potential advantages in terms of scalability, because a few keywords can map to an arbitrarily complex set of commands and procedure, and memorability, because the keywords themselves are descriptive of the task being learned.
10. Based on the results of all of the studies described above, we outline specific areas for future work to continue to develop the idea of task-centric interfaces.

Having presented a high-level summary of the contributions of the work, we now step back and describe the particular problem setting and scope of our investigation, and then give an outline of the specific research activities presented in this dissertation.

1.5 Research Scope

We start by defining the scope of our work in this dissertation, including the particular problem, user group, scenario of use, and type of task that we are focusing on in this work. We also discuss the particular independent variable that we are manipulating in our investigation.

1.5.1 Non-Expert, Occasional Users

Our goal is to assist non-expert, occasional users of feature-rich software. This user group is characterized by the following attributes:

- They do not use the application frequently (e.g. daily or weekly), though they may still use it regularly (e.g. on a monthly basis).
- They may have a weak or incomplete understanding of the commands and features available in the application.
- They may not have a strong understanding of the vocabulary and terminology surrounding the application and the application's domain.
- They are primarily interested in completing their primary task, with learning and expanding their skill set as secondary considerations.

In Chapter 3, we provide evidence to suggest that occasional use of feature-rich software by non-expert users is a common practice.

1.5.2 Unfamiliar Tasks

Our scenario of use is the user performing an unfamiliar task. This could be a *new* task that the user is performing for the first time or an *occasional* task that the user does not perform regularly enough to remember the full details of. In Chapter 3 we present evidence to suggest that this is a common scenario for certain feature-rich software applications.

Within this scenario, the specific problem that we are seeking to address is what Norman referred to as the “gulf of execution”, or the gap between a user’s goal for action and the means to execute that goal (D. A. Norman and Draper 1986).

1.5.3 Tasks Involving Ill-Defined Problem Solving

We focus on supporting users in performing tasks that involve an element of ill-defined problem solving. Ill-defined problems require a human to judge the results of individual operations, and then adjust their actions accordingly (Schön 1983). For example, selective colorization (retaining color in some parts of an image, while converting the rest to greyscale) can be considered to be an ill-defined task because the user must be “in the loop” to select the portion of the image to remain colored.

This type of task is common in image editing applications, where users often engage in creative activities that have a subjective or aesthetic element to the desired outcome. However, ill-defined tasks are not limited to this area. For example, exploratory data analysis also includes a strong

element of ill-defined problem solving, because though there are established processes for drawing insights out of data, human judgment is required throughout the process.

1.5.3.1 Relationship to Automation

A trend in feature-rich software applications has been to create automatic tools to perform tasks that once required human judgment. For example, Photoshop CS5 introduced several “content-aware” versions of fill and brushing tools that analyze the contents of an image to intelligently remove unwanted objects or regions within the image, tasks that once required human judgment throughout the process. Numerous techniques have also been investigated to completely automate tasks using macros (Bergman et al. 2005; Berthouzoz et al. 2011), machine learning (Berthouzoz et al. 2011; Grabler et al. 2009b; Yeh, Chang, and Miller 2009), mechanisms to automatically personalize scripts for the current user (Leshed et al. 2008), or the crowd (human workers) (Bernstein et al. 2010).

There are two points to make on the relationship between ill-defined tasks and task automation. First, there are certain tasks that, by their nature, cannot be automated because the desired result is too dependent on the aesthetic judgment of the user. Second, as more tasks become automated through macros, scripts, and higher-level commands, these become component operations for even higher level ill-defined tasks. As a result, it’s possible that task automation will lead to users spend a larger proportion of time on ill-defined tasks, as automatable tasks become “one click” operations.

Considering these points, task automation is not at odds with techniques to support ill-defined tasks, and developments in these two areas have the potential to benefit one another.

1.5.4 Evaluating an Alternative Problem-Solving Process

Our intervention to support the scenario outlined above (non-expert users performing unfamiliar, ill-defined tasks) is to introduce a new kind of interface designed to support an alternative problem-solving process to that currently supported in feature-rich software.

As a result, the goal of our evaluation is both to show that task-centric interfaces successfully support an alternative problem-solving process, and to demonstrate that this provides benefits over current practices. To achieve this, we employ a mix of quantitative and qualitative approaches to evaluate the impact of our task-centric interface design.

1.5.5 Simplifying Assumptions

To focus on investigating the potential of task-centric interfaces, we make a few additional simplifying assumptions in this dissertation.

First, we focus on investigating task-centric interfaces from the perspective of the end-user, leaving the question of how to create a suitable corpus of task-specific knowledge to future work. In Chapter 9, we discuss some potential ideas for how this could be accomplished.

Second, in AdaptableGIMP and Workflows we use keyword search as a “black box” component, assuming that off-the-shelf techniques can be used to match a user’s query with appropriate resources.

Finally, in our study of Workflows, we confine our investigation to a scenario in which a user wishes to perform an unfamiliar task for which a single appropriate workflow is available in the system. This is a logical place to start, because demonstrating benefits in this scenario is a precondition for the task-centric interfaces approach working more generally. In Chapter 9, we discuss how our investigation could be extended to other scenarios.

1.6 Summary and Outline

In this section we give an overview of the research activities presented in this dissertation, and how they fit together to accomplish our research objectives. A diagram of the path of questions and research activities taken in this work is shown in Figure 2.

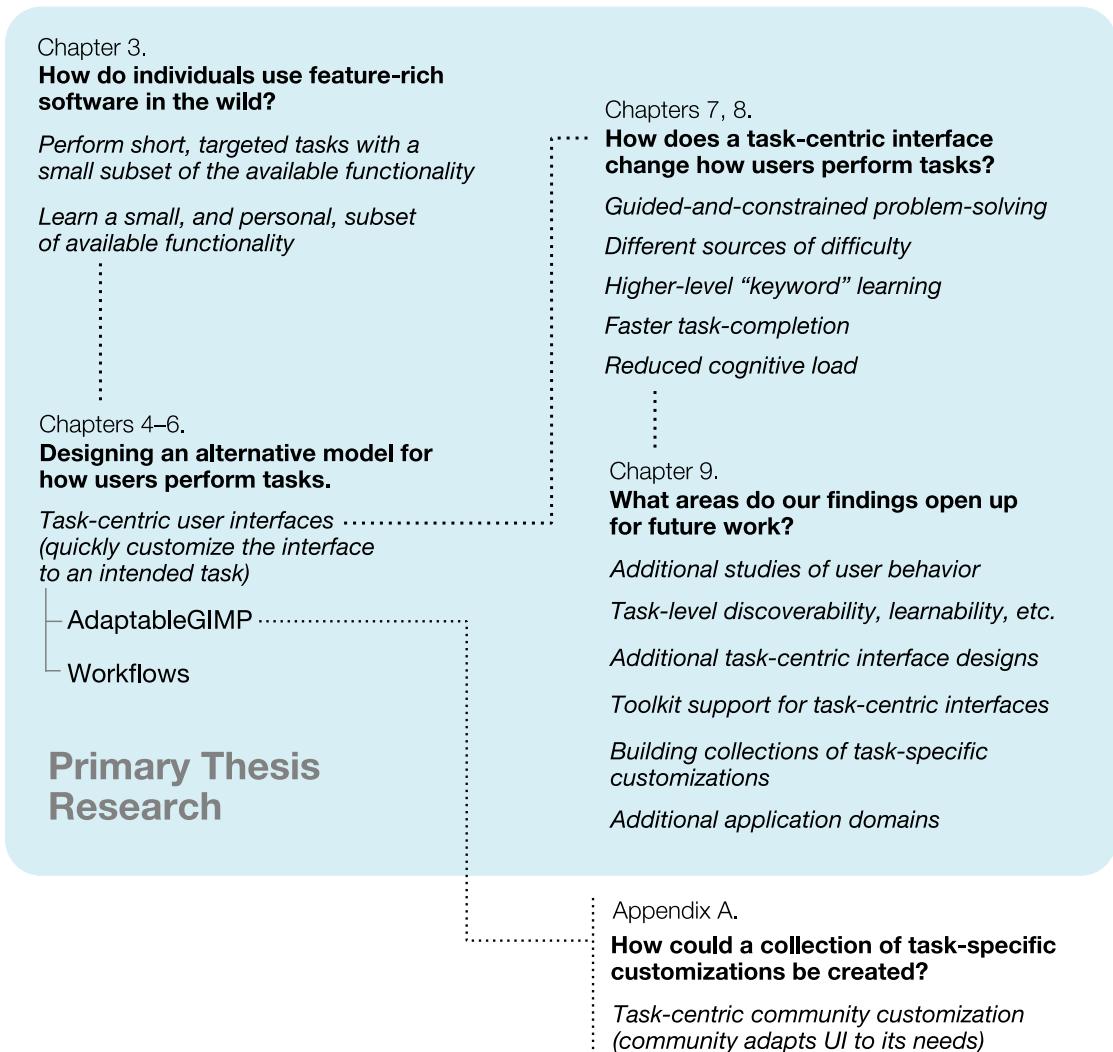


Figure 2. Research path showing research problems, activities, and main results. Bold text indicates research questions or activities. Italics indicate main results.

To start, the work presented in this dissertation builds upon work in a number of related research areas, which we outline in Chapter 2.

In Chapter 3, we present an analysis of usage data gathered over a 2-year period using *ingimp*, an instrumented version of the GIMP image editor. This provided an ecologically valid window into how feature-rich image editing software is used “in the wild” by actual users.

In Chapter 4, we present a detailed introduction to the idea of a task-centric interface. We describe the properties of a task-centric interface, and contrast it with related techniques that have been explored in research and in existing feature-rich software.

In Chapters 5 and 6 we present the design phase of the project. In Chapter 5 we present AdaptableGIMP, which included an initial iteration of our task-centric interface design, and also explored the idea of *task-centric community customization*, or the idea of providing a set of tools to allow a community of users to build up a body of task-centric knowledge to meet their needs. In this chapter, we focus on the task-centric interface component of AdaptableGIMP, and present a semi-structured interview study and a think-aloud study. The results of these studies prompted us to iterate on the task-centric interface design provided by AdaptableGIMP to create Workflows, which we describe in detail in Chapter 6.

In Chapter 7 we present a laboratory study that we conducted to understand how the task-centric interface provided by Workflows impacts the strategies employed by users to perform unfamiliar tasks. In particular, we compared use of Workflows with an unmodified version of GIMP and a web browser, representing current practices for completing unfamiliar tasks in feature-rich software. Participants performed the same tasks in two sessions separated by at least two weeks, allowing us to test the impact of our design for tasks that the user had never performed before, and for occasionally performed tasks for which the user has some familiarity. This also allowed us to gain insights into what users learned in the first study session.

In Chapter 8, we discuss important findings from our study of Workflows and our large-scale study of ingimp in greater detail. In particular, we tie the results of our study of Workflows back to the conceptual design of task-centric interfaces, discuss the impact of task-centric interfaces on learning, and highlight the potential of keyword learning to enable learning of feature-rich software at a higher conceptual level than individual commands. We also discuss how task-centric interfaces could lead to radically different interfaces for assisting users with performing short, targeted tasks.

Finally, in Chapter 9 we provide a set of high level conclusions for the research described in this dissertation, and outline specific directions for future work, concluding our discussion of task-centric interfaces and the main content of this dissertation.

In Appendix A, we present some additional information the AdaptableGIMP project. In particular, we discuss the idea of task-centric community customization, and present additional results from the semi-structured interview study from Chapter 5, and findings from a ten-month public deployment of the system.

In Appendix B, we present study materials that we developed for use in the study of Workflows presented in Chapter 7.

1.7 Terminology

Before we continue, we briefly define a number of terms that are frequently used throughout this dissertation.

task – We use the term “task” to refer to an activity to be undertaken to achieve a desired goal. This is similar to the definition of task used in the task modeling literature (e.g. see the introduction to (Mori, Paternò, and Santoro 2002)), however in this work we frequently discuss tasks with the understanding that the user does not currently know how to reach the desired goal.

goal – The task modeling literature refers to goals as “either a desired modification of state or an inquiry to obtain information on the current state.” (Mori, Paternò, and Santoro 2002). Our use of “task” tends toward the former of these definitions, with the desired modification of state typically being in a document or the state of an application.

commands and tools – We use the term “command” to refer to discrete capabilities of an application that are available to the user, typically presented through toolbars, toolboxes, and menus. In places, we substitute the term “tools” for commands when referring to capabilities that have a modal component (e.g. the Paintbrush tool in an image editing application, which can be selected once and then invoked multiple times).

features (of software) and functionality – We also use the terms “features” and “functionality” to refer to the capabilities that the application makes available to the user. In comparison to the terms “commands” and “tools”, these terms are more expansive, additionally encompassing settings, options, dialogs, and other capabilities of the system.

Chapter 2

Background Literature

The ultimate goal of this work is to better support individuals as they perform unfamiliar tasks in feature-rich software. In this chapter we review what is currently known about how users respond to feature-rich interfaces, and present existing techniques that have been proposed to assist users of these applications.

2.1 Learning and Using Feature-Rich Software

A number of projects have been undertaken to understand how individuals use and respond to rich feature sets in software. The work we review in this section serves to motivate our approach, and also establishes a backdrop of what is currently known about how users currently learn and use feature-rich software.

2.1.1 Large-Scale Analyses of Use of Feature-Rich Software

We start by reviewing work that has investigated use of feature-rich software at a macro-level, using large-scale analyses of log data. Macro-level analyses of feature-rich software are able to characterize population-wide trends that may not be apparent in small-scale studies, and also have high ecological validity as compared to laboratory studies.

Large scale analyses of feature-rich software has examined use of the UNIX command line (Greenberg 1993; Draper 1984; Hanson, Kraut, and Farber 1984), text editing applications (Whiteside et al. 1982; Cook et al. 1995; Linton, Joy, and Schaefer 1999; Kay and Thomas 1995), and software development environments (Murphy, Kersten, and Findlater 2006).

A robust finding from these studies is that users' *command vocabularies*—the set of unique commands used by a user—are relatively small as compared to the set of available commands, and also tend to be idiosyncratic, with low overlap between users (Greenberg 1993; Cook et al. 1995; Linton et al. 2000).

Another robust finding is that the frequency with which individual commands are used, across the entire community of users, follows a long-tailed distribution in which a very small number of commands accounts for the vast majority of observed command use, with the remaining commands

used by a small percentage of the user population each (Cook et al. 1995; Greenberg 1993; Kay and Thomas 1995; Cook et al. 1995; Linton, Joy, and Schaefer 1999; Whiteside et al. 1982; Draper 1984).

We note two holes in existing large-scale studies of feature-rich software. First, existing work has focused on domains where text is the main unit of concern (the UNIX command line, text editors, and IDEs). In particular, we are unaware of any work that has investigated large-scale use of applications that follow a primarily direct-manipulation paradigm. Second, existing large-scale analyses of usage data have focused on analyzing use of commands and features (e.g. command frequencies, command vocabularies). What is missing is work that characterizes use of feature-rich software at the level of sessions and tasks.

This motivated us to conduct an analysis of data from *ingimp*, an instrumented version of the GIMP image editor, which we present in Chapter 3. The results of our study show that the major findings discussed above also apply to direct manipulation applications in the image editing domain. We also found that *ingimp* users tend to use the application to perform short, targeted tasks using a small subset of the full functionality available. The findings from this study directly motivated a number of aspects of our task-centric interface design.

2.1.2 The Impact of Rich Feature Sets

Given that large-scale studies have shown that users only make use of a small amount of the available functionality in feature-rich software, a logical next question is what effect this unused functionality is having on users.

Quantitatively, past work has found that the number of interface elements negatively impacts task time, particularly for novice users, whose visual search time is a linear function of the number of “relevant” items present in an interface (e.g. within a given menu or toolbar) (Cockburn, Gutwin, and Greenberg 2007). As expertise increases, the impact of excess complexity on performance is lessened. However, for all but perhaps the most expert users, command selection time is always some function of the number of relevant commands available (Cockburn, Gutwin, and Greenberg 2007; K. L. Norman 1991).

Qualitatively, McGrenere and Moore found that many users find rich feature sets to be overwhelming, frustrating, and difficult to navigate (McGrenere and Moore 2000); and Lazar et al. found that “Missing / hard-to-find or unusable features” was the second most commonly cited cause of frustrating experiences with computers in the workplace (Lazar, Jones, and Shneiderman 2006).

Though the McGrenere and Moore study did find evidence that unused functionality had a negative effect, there was no broad agreement among participants on how they wanted to see excess functionality handled. Only 25% of participants wanted unused functionality to be removed entirely, and 45% wanted unused functions to be tucked away. Moreover, 51% of participants agreed that it is important to continually discover new functions of the application (McGrenere and Moore 2000).

In concert with the findings from the previous section, these findings highlight the challenge of designing interfaces for rich feature sets. Unused functionality has a negative impact on usage, but you cannot simply trim down the interface to include the “most important” commands, because this set of commands varies from user to user. In our task-centric interface design we address this problem through task adaptation, which customizes the interface for specific tasks.

2.1.3 Learning to Use Feature-Rich Software

The work reviewed in the previous two sections focuses on the high-level impact of rich feature sets on users. In this section, we discuss what is currently known about how users go about learning and performing tasks in feature-rich software.

This is not a new question. Some of the earliest research into the usability of software, conducted in the 1980s, examined how individuals learn to use early word processing systems (Carroll et al. 1985; Mack, Lewis, and Carroll 1983). In the introductory chapter to *The Nurnberg Funnel*, a collection of a series of studies on this topic carried out by researchers at the IBM Watson Research Center, John Carroll summarizes the zeitgeist of this period in the history of computing:

In the late 1970s, access to computing equipment was expanding rapidly to people who were not programmers or other computer professionals. These people often experienced serious and frustrating difficulties when they attempted to use computers. Tasks they had been able to do had to be relearned. Competent secretaries, accountants, and lawyers were—at least temporarily—returned to varying levels of incompetence until they could master the system. (Carroll 1990)

As a response, Carroll and others set out to understand the difficulties that individuals had with learning software and how to better help users to learn this increasingly prevalent technology. This early work provided some of the first insights into how users react when faced with new software, and has had an enduring impact on research in this area.

A robust finding from this early work is that users of software systems are *task-focused*; continuous progress toward goals is paramount, and time spent on other concerns, including learning how to

perform a task, are minimized to the greatest extent possible (Carroll and Rosson 1987; Rieman 1996; Carroll et al. 1987; Carroll 1990; Mackay 1991).

In particular, Carroll and Rosson coined the term the “paradox of the active user” to refer to the phenomenon that users could ultimately save time by spending an initial period learning a system, but that this is counter to how they behave in practice (Carroll and Rosson 1987); and Rieman concluded that “Task-free exploration is unlikely to occur, not only because systems are uninteresting, but because the users’ fundamental goal is to complete their work-related tasks. To that end, they will use their knowledge of the available resources to choose the fastest, most focused learning strategies available.” (Rieman 1996). Finally, in a study of software customization behavior, Mackay found that users were hesitant to take time away from their primary tasks to spend time customizing because it meant trading off short-term productivity for an unknown long-term benefit (Mackay 1991).

Our idea for task-centric interfaces was motivated by these findings in a number of ways. First, task-centric interfaces provide assistance with performing specific tasks, as opposed to more general assistance with using the system. Second, the task adaptation component of task-centric interfaces organizes commands and features for the task at hand, directly supporting the user in completing their current task. Finally, the task knowledge provided by the task-centric interface is introduced in the context of a task, so users can engage in just-in-time learning as needed.

2.1.4 Problem-Solving Strategies

Past work has also been undertaken to understand the specific problem-solving strategies that users employ when faced with unfamiliar tasks in feature-rich software. This work is relevant to our work, because we are positing a new way for users to solve problems and achieve goals in feature-rich software.

Studies have shown that users adopt a range of different problem-solving strategies to perform tasks in feature-rich software, but that they favor an exploratory approach consisting of self-guided exploration of the interface, and trial-and-error experimentation, to attempt to make progress toward their goals (Rieman 1996; Novick, Andrade, and Bean 2009; Andrade, Bean, and Novick 2009; Trudel and Payne 1995; Carroll 1990).

Novick, Andrade, and Bean showed that this exploratory strategy can be effective for completing tasks, but can lead to suboptimal solutions, and its effectiveness is heavily dependent on the affordances provided by the interface (Novick, Andrade, and Bean 2009). The authors also identify

two common difficulties faced by users when following a trial-and-error approach: *hidden affordances*, where the interface does not provide sufficient affordances to allow the user to proceed with the task; and *false affordances*, where the system includes affordances that mislead the user because they appear to be related to the task, but are actually not.

Trudel and Payne found that users engaging in exploratory problem-solving exhibited maladaptive behavior characterized by a large number of mode changes and goal-shifts, and argue that exploratory problem-solving can lead to poorer learning outcomes if the user does not reflect on the results of their explorations (Trudel and Payne 1995).

Finally, both work by Rieman, and by Novick, Andrade, and Bean, suggest that users' choice of problem-solving strategies are dependent on their level of experience with the system (Rieman 1996; Novick, Andrade, and Bean 2009). In particular, Rieman found that inexperienced users were more likely to select whatever learning method is most readily available, while more experienced users were more selective, choosing the strategy that would help them acquire the skills they needed in the least possible time.

In summary, past work has shown that users tend to favor and adopt exploratory problem-solving strategies, but that these strategies are associated with a number of problems: exploratory learning can lead to suboptimal results; it can lead to problems caused by false or hidden affordances; and it can lead to less learning if the user does not reflect on their actions. Moreover, these difficulties appear to be more pronounced for novice users, who are less selective about their choice of problem-solving strategies.

2.1.5 The Challenge Posed by Feature-Rich Software

The main points from this section can be summarized as follows:

- Command frequencies across large numbers of users tend to follow a long-tailed distribution, and individual users' command vocabularies are small and idiosyncratic. This indicates that users will be faced with excess interface complexity from unused features.
- Rich feature sets and excess interface complexity have a negative effect on user experience.
- Users are task focused, which makes them hesitant to spend time on activities that do not directly advance their primary task, including learning.

- Users adopt a range of problem-solving strategies, but appear to favor exploratory and trial-and-error approaches.
- Evidence for the effectiveness of this exploratory problem-solving strategy is mixed. It can enable users to complete tasks, but may lead to suboptimal solutions; its effectiveness appears to be heavily dependent on the affordances provided by the interface; and it may lead to poorer learning outcomes if the user does not reflect on the results of their explorations. Moreover, these difficulties may be more pronounced for novice users.

Overall, these findings argue that feature-rich software is particularly challenging for novice users, and for users performing unfamiliar tasks. Moreover, current help resources appear to be incompatible with users' task focus and preference for exploratory problem-solving strategies. This suggests that it is worthwhile to investigate alternative ways to support users in performing tasks in feature-rich software. In the next section, we review existing work that has been done in this area.

2.2 Support Techniques for Feature-Rich Software

Research into how to improve use of feature-rich software has focused on three main areas: interface personalization, task automation, and tutoring. We review each area in turn.

2.2.1 Interface Personalization

Interface personalization approaches provide a mechanism for dealing with interface complexity by customizing the interface to better suit the individual user. A wide range of different approaches to personalization have been explored:

Level-structured interfaces offer the user access to a small number of predefined feature subsets (Shneiderman 2002). Examples include the Training Wheels interface (Carroll and Carrithers 1984), which blocked invocation of commands for advanced tasks, and software that offer “Basic” and “Advanced” modes.

Adaptable interfaces give the user a mechanism to customize the interface to suit their needs (McGrenere, Baecker, and Booth 2007; MacLean et al. 1990), typically by adding or removing commands from toolbars or menus. The advantage of this approach is that it gives the user control over how the interface is customized. However, empirical studies of customization behavior have found that users are often hesitant to deviate from their primary task to spend time to learn and use customization facilities (Mackay 1990; Mackay 1991) (with gamers being a notable exception (Dyck

et al. 2003)), or to re-customize when their work habits change (McGrenere, Baecker, and Booth 2002). Mackay also found that users desired to customize based on “patterns of behavior” rather than lists of commands and features (Mackay 1991), which provides some motivation for our task-centric approach.

Adaptive interfaces model the user’s interests, preferences, and usage and automatically initiate changes to the interface (Findlater et al. 2009; Gajos et al. 2008; Gajos et al. 2006). This approach has the advantage of not requiring the user to take time to customize. However, studies have shown that system-initiated changes to the interface can lead users to feel a lack of control, transparency, and predictability (Fischer 1993; Höök 2000).

Finally, *mixed-initiative* approaches have been investigated, which combine the elements of adaptable and adaptive strategies (Bunt, Conati, and McGrenere 2007; Oppermann 1994; Fischer 1993; Bunt, Conati, and McGrenere 2004).

Arcing back to the goal of this dissertation, there are a number of reasons why existing approaches to interface personalization are not well suited to supporting users in performing unfamiliar tasks.

First, existing personalization approaches have focused on providing fast navigation to individual commands and features. They do not address how an interface might be optimized for performing specific tasks.

A partial exception to this is the Perspectives feature in the Eclipse IDE. Perspectives can be considered to be a level-structured interface based on tasks; loading a perspective (e.g. “Debug”) rearranges the layout of views and editors in the interface to suit it to the various activities involved in debugging. While this has clear conceptual similarities with our task-centric approach, the customizations provided by Perspectives are suited to broad classes of tasks instead of specific tasks (e.g. debugging encompasses many specific activities, including setting a breakpoint, advancing the instruction pointer, etc.), and customizations do not provide specific guidance on how to perform tasks (that is, they do not provide task knowledge). Thus, Perspectives does not address the goal of supporting users in performing unfamiliar tasks, which is our focus.

Second, to provide benefits, existing personalization approaches typically require the user to be familiar with the application’s functionality (so the user can effectively customize the interface), or the system to have a model of the user’s past behavior (so the system can perform reasonable

customizations). In the situation of a user performing an unfamiliar task, neither of these is likely to be the case.

There is some work on interface personalization that circumvents these requirements by taking advantage of the efforts or usage of other users of an application. In the area of adaptive interfaces, OWL (Linton et al. 2000) and CommunityCommands (Matejka et al. 2009) provide adaptive recommendations of commands based on the usage data of similar users in an organization, and for adaptable interfaces, Kahler proposed the notion of *collaborative tailoring*, or the idea of providing explicit support for sharing customizations with other users (Kahler 2001a). However, we are unaware of any work that has applied such an approach to assisting the user with performing unfamiliar tasks.

In summary, existing work on interface personalization has focused on customizing interfaces on the granularity of individual users, to support efficient access to individual commands. We are unaware of work that has explored interface customization as a method to assist with performing specific unfamiliar tasks.

2.2.2 Task Automation

A second major approach to assisting users of feature-rich software has been task automation. The classic examples of task automation in feature-rich software are wizards, macros, and scripts. Wizards “[guide] a person along an efficient path to success, autonomously completing those steps of the task that do not require the person’s attention.” (Dryer 1997) In their most common form, wizards take the form of a linear dialog composed of multiple steps that prompt the user for information necessary to further the task. Scripts and macros execute a pre-defined set of operations, typically without input from the user.

Wizards and scripting capabilities are common in feature-rich applications. However, these techniques have a number of limitations. First, they are most effective for tasks that have algorithmically-derived solutions, making them poorly suited to ill-defined tasks. Second, they tend to adapt poorly to unforeseen conditions. Finally, they are laborious to create and maintain, often requiring advanced skills.

To address these drawbacks, a number of alternative approaches to task automation have been investigated. Numerous approaches have been investigated to provide automation of traditionally difficult to automate tasks. These include using machine learning and image recognition (Berthouzoz

et al. 2011; Grabler et al. 2009b; Yeh, Chang, and Miller 2009), automatically personalizing scripts for the current user (Leshed et al. 2008), and using the crowd (human workers) to complete a task (Bernstein et al. 2010).

To address the problem that scripts are laborious to create, a number of systems have adopted a *by demonstration* approach where the task is performed in the system to record a script (Grabler et al. 2009b; Leshed et al. 2008; Bergman et al. 2005; Berthouzoz et al. 2011; “Adobe Labs Tutorial Builder” 2012). The use of data from multiple recordings to make scripts more general has also been explored (Bergman et al. 2005; Berthouzoz et al. 2011).

Typically, research on automation techniques has focused on minimizing the input required from the user. However, some of these projects have explored how to automate parts of a task, while keeping the user “in the loop”. DocWizards (Bergman et al. 2005) and CoScripter (Leshed et al. 2008) provide scripts that can prompt the user to take actions. Tutorial Builder (“Adobe Labs Tutorial Builder” 2012) creates text/image tutorials that include a “Show me in Photoshop” button that executes the corresponding action in-application. Tutorial-Based Applications (Laput et al. 2012) convert web-based tutorials into interactive macros that allow the user to adjust how an individual step is applied, and have the results automatically apply through the rest of the tutorial.

By expanding the set of tasks that can be completed with minimal input from the user, task automation techniques do have the potential to help users with performing unfamiliar tasks. However, as discussed in Chapter 1, there will always be ill-defined tasks that cannot be completely automated, and task automation may in fact increase the proportion of these tasks faced by users. Thus, these approaches do not represent a general-purpose technique for task support.

Moreover, with task-centric interfaces, we are positing a new way of working with feature-rich software that includes both a mechanism for helping the user to complete a given task, as well as a mechanism for accessing assistance for a task. That is, the scope of our investigation is different from work on task automation techniques, which do not address how users would locate relevant wizards, scripts, or macros.

2.2.3 In-Application Tutoring Systems

A third major approach that has been explored to assist users with using feature-rich applications is in-application tutoring systems, which provides instruction on how to complete tasks within the context of the application’s interface.

Early work in this area by Tuck and Olsen referred to this as a *guided task paradigm* for help delivery. Tuck and Olsen's system translated scripts describing tasks into in-application demonstrations, with arrows and explanatory text to indicate the interactions with the interface that would be required at each step (Tuck and Olsen 1990).

A number of in-application tutoring systems in this style have been developed. Stencils (Kelleher and Pausch 2005) presents instructions on a semi-transparent coloured layer over the application's interface, limiting the user's interaction with the interface to components that are revealed through holes in the translucent layer. Google SketchUp's "self-paced tutorials" ("Google SketchUp Training" 2012) present instruction in-context in documents. Sketch-Sketch Revolution (SSR) (Fernquist, Grossman, and Fitzmaurice 2011) uses many of the same techniques as previous systems, and additionally applies a number of techniques designed to allow the user to experience success, and keep the user in a flow state (Csikszentmihalyi 2008) while they learn.

Research on these systems has explored a number of novel interaction techniques for tutoring the user in how to perform tasks in-application. However, these techniques do not support the user while they are in the process of performing their own tasks. Instead, the focus is on teaching the user how to use the system while performing an artificial task, with the idea that they could then apply what they have learned to their own goals. In principle, many of these techniques could be adapted to provide support while the user was advancing their own goals, and this has been suggested in the form of Guides, a user interface agent that monitor the user's interaction with the system and presents information to assist their progress (Dryer 1997). However, implementing this support is difficult in practice, because it requires that the system have an accurate model of the user's specific intentions as they interact with the system.

In contrast, our work on task-centric interfaces investigates a mechanism for assisting the user with completing their own unique tasks. This is more compatible with the task-focus of users, discussed earlier in this chapter.

2.2.4 The Need for Techniques to Support Unfamiliar Tasks

The main points from this section can be summarized as follows:

- Existing interface personalization techniques have not been designed to assist users with performing unfamiliar tasks.

- Existing approaches to in-application task support have focused on either (1) providing more sophisticated techniques for automating a given task, or (2) teaching the user how to perform a task in an artificial situation. Neither of these techniques offers a general-purpose approach to help the user with performing unfamiliar tasks.
- Existing work on in-application task support has not addressed mechanisms for locating task support for a user's current goal.

We conclude that the space of general-purpose techniques to support users in performing unfamiliar tasks in feature-rich software is largely unexplored. Moreover, the idea of designing such a technique around interface personalization on the granularity of tasks is novel, and well-motivated by our current understandings about how users learn and use feature-rich software. Our work on task-centric interfaces in this dissertation is intended to contribute to filling these gaps, and exploring this space for design.

In the next chapter, we begin our investigation of task-centric interfaces by returning to another gap in current research that we identified earlier in this chapter. While there have been a number of large-scale analyses of usage of feature-rich software, none have investigated large-scale use of applications that follow a primarily direct-manipulation paradigm. Moreover, existing large-scale analyses have focused on use of commands or features (e.g. overall command frequencies, users' command vocabularies), and have not characterized use of feature-rich software at the level of sessions and tasks. In the next chapter, we present the results of a study conducted to fill this gap.

Chapter 3

Characterizing Large-Scale Use of a Direct-Manipulation Application in the Wild

Our investigation of task-centric interfaces starts with a study that we conducted to understand how individuals make use of feature-rich image editing software “in the wild”, outside of a laboratory setting. In particular, in this chapter we present an analysis of usage logs collected from a public deployment of *ingimp* (Terry et al. 2008), an instrumented version of the open source GNU Image Manipulation Program. Our analysis includes data from more than 200 users and 4000 sessions, collected over a two-year period from May 2007 to May 2009.

As discussed in Chapter 2, existing large-scale analyses of feature-rich software have focused on applications in primarily text-based domains, and have not characterized use of feature-rich software at the level of sessions and tasks.

The analysis presented in this chapter extends previous work by studying a primarily direct-manipulation application, and providing characterizations of sessions and tasks. Specifically, the key findings from our analysis are:

- The majority of users of *ingimp* use the software to perform relatively short, targeted tasks (averaging ~6 minutes of active usage per session.)
- Users mostly work on documents of modest complexity, both in terms of resolution and number of layers per document.
- Users perform tasks using only a small proportion of the full functionality of the application (an average of only ~6 unique commands are used in any particular session, of the nearly 500 available.)
- Users’ command vocabularies tend to be small (averaging only ~27 commands), and have little overlap with one another (the vast majority of observed commands are used by no more than 10% of users each.)
- Users of *ingimp* perform a range of different high-level tasks, including some relatively sophisticated image editing tasks.

These findings directly motivated us to develop the idea of task-centric interfaces. Our finding that the majority of users use the application for short, targeted tasks establishes this as a valid use case for feature-rich software that should be designed for; and our finding that users only require a small percentage of the available functionality to perform tasks suggests task-centric customization as a potentially effective solution.

The rest of this chapter is structured as follows. We start by describing the ingimp project and the ingimp dataset that we analysed. Next, we provide a basic summary of ingimp users, including the characteristics of their sessions (e.g. length, frequency of use) and the characteristics of the documents that they worked on. We then present a summary of command usage across the ingimp user community. Finally, we characterize the specific kinds of tasks performed by ingimp users, through use of an unsupervised clustering technique. We conclude with a discussion of the implications of our findings for task-centric interfaces.

Note that the content presented in this chapter is drawn from the published paper *Characterizing Large-Scale Use of a Direct Manipulation Application in the Wild* (Lafreniere et al. 2010) which reports on work carried out by myself in collaboration with Andrea Bunt, John Whissell, Charles Clarke, and Michael Terry.

3.1 The ingimp Dataset

ingimp is an instrumented version of the open source GNU Image Manipulation Program (Terry et al. 2008) and an experiment in *open instrumentation*, or the idea of openly collecting and publicly disseminating rich application usage data. All collected data was made publicly available on the project’s website (www.ingimp.org) for the duration of the project.

ingimp was designed to collect the following information:

- *Activity tags* – optional user-supplied keywords describing how the user intends to use the application (prompted for at the start of each session)
- *System characteristics*, including operating system, CPU, number of monitors, screen resolution, and time zone
- *Document summaries*, including the resolution and number of layers in images
- *Command invocations* that appear on the undo stack

Users' locales were not explicitly recorded, but could be deduced by examining the localized command names in log files to determine the likely locale.

Users' logs were automatically transmitted to the server when the application closes. We consider each such instance of the application being opened and closed a *session*. To permit tracking of users' activities across sessions, each user was assigned a randomly-generated identification number when they first ran ingimp, which was subsequently included in the log of each session.

3.1.1 ingimp Deployment and Distribution

ingimp was publicly released in May 2007 at an open source graphics conference, the Libre Graphics Meeting (LGM), and was freely available for download from www.ingimp.org for the duration of the project. Since ingimp was released under the GNU Public License (GPL), anyone was free to download, install, and use it. Following its announcement, ingimp was featured on a number of websites, including Slashdot, GIMP's French language project page, and a story published on an open source-themed news site. Of the various announcements, the Slashdot coverage had the greatest single impact in user uptake, rapidly increasing the user base in a week's time.

Given the factors discussed above, ingimp's user base during the study can be roughly approximated as Slashdot readers, GIMP users in general (with a slight emphasis on French users due to the French announcement), and those sympathetic to open source software. We provide more specific details of the user base as we analyze the collected data.

In the sections that follow, we first characterize the user community of ingimp, including users' locales, time zones, computing environments, session lengths, and documents. Next, we analyze command usage across the ingimp community. Finally, we use an automated cluster analysis technique to gain insights into the high-level tasks that users perform using ingimp.

Our analysis considers all log files collected in the two-year period between May 15, 2007 and May 15, 2009.

3.2 The ingimp User Base

We start our analysis by considering characteristics of the user base. One complication in doing so, however, is the presence of *curious bystanders*. By definition, open source software can be downloaded, installed, and used by anyone. As a result, it can be expected that a number of people will try out the application out of curiosity alone. These users may not use the application after an

initial test, making it worthwhile to filter them out before performing more in-depth analysis. We start by describing our criteria for filtering out these users, and then analyze the remaining users.

3.2.1 Defining Significant Users

For the ingimp dataset, we define a *significant user* as anyone who has used the application and saved a document on at least two separate days. Using these criteria, our data set consists of 211 significant users who have contributed 4198 logs. This group constitutes only 22% of all users who submitted logs, though these users produced 75% of the log files (4198 out of 5612). While this criterion may exclude some legitimate users, the fact that this minority accounts for 75% of all log files indicates that the criteria are reasonable. All subsequent data analyses use this significant user criterion to pre-filter the log files. Logs from the primary developers and researchers were also excluded from data analyses.

3.3 Characterizing Users' Sessions

As previously mentioned, we consider a *session* to correspond to activity logged between the time the application is opened and when it is closed. The median number of sessions for an individual user was 11 (IQR 15.5). As the large measures of spread indicate, the number of sessions for individual users varied widely, ranging from 1 to 168 (the significant user criteria were applied to the entire ingimp dataset, not just the dataset analyzed, leading to some single-session users in our analysis).

The duration of sessions varied widely as well, ranging from a few seconds to several hours. The median session length for the community was 9 minutes (IQR 33 minutes). However, this measure simply refers to the duration for which the application was open. To get a sense of how much of this time was spent actively using the application, we reconsider these figures with idle periods removed. Given that the mean time between commands was 19 seconds, we chose 120 seconds as a conservative threshold for idle time. Once idle time is removed, Figure 3 shows that most sessions include very little active usage time. The median active usage time across sessions was 6 minutes (IQR 15 minutes).

On the whole, while there were a small number of users who used ingimp frequently and actively for long periods, these results indicate that the majority of ingimp users use the application infrequently, to perform short tasks. This suggests that the majority of ingimp users are not expert or professional users of the application, who we would expect would use the application for longer periods of time.

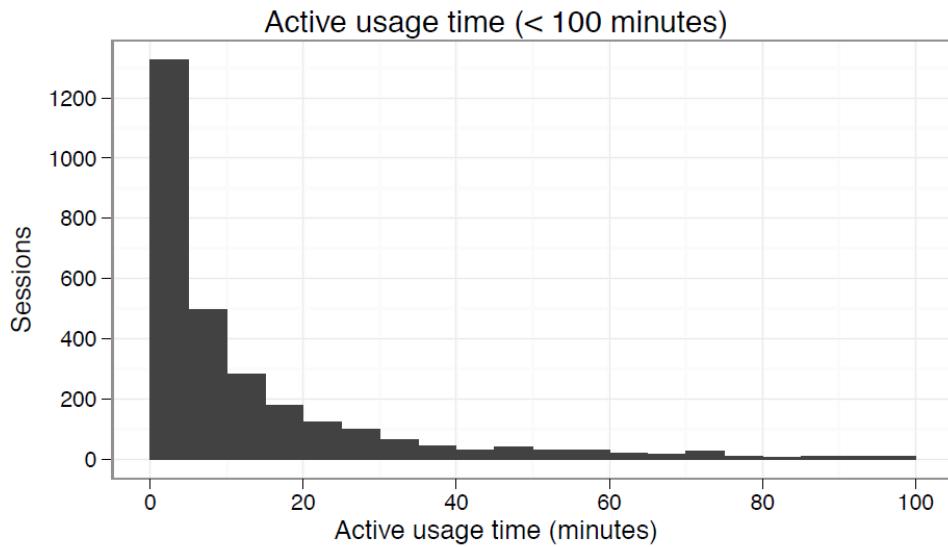


Figure 3. Histogram showing active usage time for sessions (session durations with idle time removed)

3.4 Characterizing Users' Documents

A total of 13,609 images were operated on during the data collection period. Looking at characteristics of the images, we see that the median maximum image resolution for individual documents is 800x691 (IQR 1131x1152). These results suggest that the primary uses of *ingimp* did *not* include working with high-resolution bitmap images from digital cameras. Also of interest is the maximum number of layers¹ per document, which provides an indication of the complexity of both the document and the user's task. The median maximum number of layers per document over its lifetime was 1 (IQR 1). This further implies that the majority of *ingimp* users are not professionals, since professionals tend to work on complex documents and utilize many layers as part of their workflow (Terry and Mynatt 2002).

3.5 Community Command Usage

Our data analysis thus far suggests that the majority of *ingimp* users performed relatively short, targeted tasks on documents of modest complexity. To gain a clearer picture of how participants used the application, we also analyzed command usage.

¹ Modern image editing applications allow one to define multiple *layers*, where each layer contains a unique bitmap. These layers are combined to create the visible image using compositing operations and masks.

ingimp records every command that is placed on the undo stack. It also logs Undo and Redo, the two meta-commands that operate on the undo stack. Across the entire application, ingimp provides approximately 500 different commands. These commands include those that modify entire regions at once (such as filters), direct manipulation tools (such as the Brush tool), and those that operate on properties of the document (such as layer compositing operations).

In this section, we characterize the number of commands used in individual sessions, consider which commands were most commonly used by the community, and examine the size of users' command vocabularies.

In these analyses, we consider only sessions in which at least one command was logged. This criterion excludes a number of sessions for which no commands were placed on the undo stack (1288, or 31% of all sessions). While these sessions lack commands appearing on the undo stack, this should not be interpreted as individuals not using the application. An analysis of user-supplied task descriptions revealed that many users report using the application to view images, convert images between file formats, or to obtain RGB color values of pixels, uses which would not place commands on the undo stack.

3.5.1 Command Statistics for Sessions

The number of command invocations per session ranged from 1 to 9236 with a median of 24 (IQR 107) (see Figure 4). If we consider repeated, successive invocations of a given command as only one invocation, the median number of invocations per session drops to 14 (IQR 47, min 1, max 1489), due to the common use of direct manipulation tools, which are often invoked a number of times in quick succession (e.g. a number of strokes of the Brush tool). Finally, if we consider only the number of unique commands used in a session, there is a median of just 6 commands per session (IQR 9, min 1, max 85).

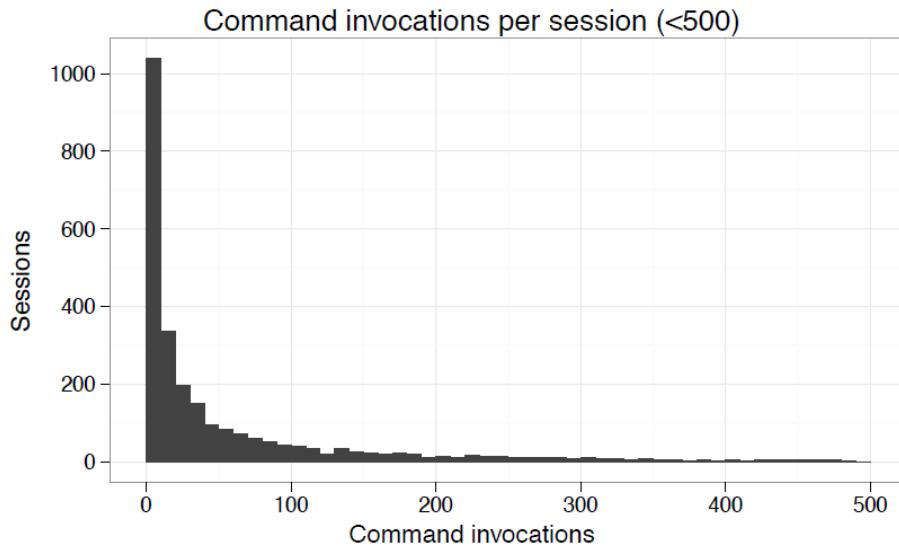


Figure 4. Histogram showing the number of command invocations per session (bins of width 10)

These results further suggest that users are engaging in relatively simple, targeted tasks; for more complex tasks we would expect to see more command invocations per session.

3.5.2 Command Coverage and Command Vocabularies

Across all sessions, there were a total of 487,308 command invocations of 352 different commands. Since there are approximately 500 different commands available in ingimp, the user community is not exercising all of the available functionality. This finding is consistent with the findings of Hanson et al. (Cook et al. 1995) for the UNIX command line, and for Linton et al.'s analysis of MS Word in which only 152 of 642 commands were observed being used (Linton, Joy, and Schaefer 1999).

We found that the size of an individual user's command vocabulary ranged from 1 to 169 commands, with a median of 27 (IQR 29), or less than 6% of the total number of available commands. This finding, that individual users' command vocabularies tend to be small in comparison to the number of available commands, is also consistent with previous findings (Greenberg 1993; Linton, Joy, and Schaefer 1999).

Considering the above observation that users' command vocabularies tend to be small, along with the previous observation that users tend to use only a small number of unique commands in a given session, we conclude that ingimp offers far more functionality than is effectively utilized by most of its user population, particularly in any individual session.

3.5.3 Overlap in Command Vocabularies

The sparse use of the application's functionality raises the question of how much overlap there is between users' command vocabularies. That is, do many users share a few small sets of commands? To answer this question, we looked at the number of commands shared by different proportions of the user community.

Proportion of users	Commands	% Commands
90–100%	1	0.3%
80–90%	0	0%
70–80%	4	1.1%
60–70%	5	1.4%
50–60%	5	1.4%
40–50%	7	2.0%
30–40%	6	1.7%
20–30%	17	4.8%
10–20%	50	14.2%
0–10%	257	73.0%

Table 1. The number of observed commands shared by different proportions of users.

As can be seen in Table 1, there is very little overlap in command vocabularies across the entire community. No single command was used by all users (Undo comes closest, being used by 91% of users). In fact, only 15 commands were used by greater than half of the population. Furthermore, 257 commands, representing 73% of the total number of distinct commands observed, were used by no more than 10% of the user community each. We conclude that users' command vocabularies have little overlap with one another. This suggests either that *ingimp* is used for widely varying tasks by different users, or that users have differing methods for achieving similar goals. This finding closely mirrors those of Greenberg (Greenberg 1993), and Sutcliffe and Old (Sutcliffe and Old 1987) for the UNIX command line domain.

3.6 Characterizing Users' High-Level Tasks

The frequency counts of commands explored in the previous sections provide perspectives on which commands are commonly used across the entire community, but they do not say much about the specific higher-level tasks performed by users.

One way to understand what tasks are performed by the community is to identify sets of commands are frequently used together. Unfortunately, this is a not a trivial task. While it is relatively easy to generate transitional probabilities from one command to another, previous work has argued that this

unit of analysis does not easily extend to describing higher-level tasks. For example, Greenberg (Greenberg 1993), and Sutcliffe and Old (Sutcliffe and Old 1987) found that this approach leads to fragile models because users' command vocabularies tend to be small and idiosyncratic—a result that we have found in our analysis as well.

In this section, we use an alternative approach based on unsupervised learning (clustering) to characterize the high-level commands being performed by the ingimp user community. We also provide evidence for the validity of our results by examining the activity tags (the optional text-based task descriptions users could enter at application start-up) provided by users. We begin by describing our method.

3.6.1 Clustering Sessions

To determine common tasks, we take the following high-level approach: First, we collect sequences of commands that correspond to separate (though unknown) tasks. We then cluster these sequences based on a measure of similarity. Finally, we extract frequently occurring commands from the resulting clusters of sequences, which serve as rough descriptions of high-level tasks.

The first step, partitioning command sequences by task, would be difficult if users performed multiple tasks per session. However, our previous analyses have demonstrated that most sessions are relatively short, have few command invocations, and include few documents. These trends suggest that individual sessions are a reasonable approximation of individual tasks. Accordingly, we apply our clustering technique on the granularity of sessions, to all sessions in which at least one command was logged (2906 sessions in total).

To perform the clustering itself, we adapted a clustering approach of Whissell et al., previously used to characterize document similarity (Whissell, Clarke, and Ashkan 2009). We first create a feature vector containing the command counts for each session. These vectors are then input into 11 well-known clustering algorithms (e.g. k -means), each of which outputs 7 clusters of sessions. From these 11 sets of 7 clusters, we would like to identify the “best” set of clusters. However, there is no objective function for making this determination. To address this, the Whissell approach works as follows. For each set of clusters produced, the clustered sessions are used as labeled training data to train a classifier for classifying sessions into clusters. The accuracy of each classifier is determined using ten-fold cross-validation. To address the problem of a trivial classifier that places everything in

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
Scale Image	Rect Select	Add Text Layer	Undo	Undo	Undo	Undo
Crop Image	Select None	Undo	Item Visibility	Bucket Fill	Eraser	Paintbrush
Undo	Undo	Move Layer	Add Layer	Select None	Move Floating Selection	Paste
Levels	Paste	Text	Paste	Rect Select	Fuzzy Select	Add Layer
Resize Image	Move Floating Selection	Move Text Layer	Add Path	Move Floating Selection	Select None	Select None
Rect Select	Anchor Floating Selection	Add Layer	Add Anchor	Paste	Add Layer	Move Floating Selection
Select None	Crop Image	Paste	Move Layer	Anchor Floating Selection	Paste	Anchor Floating Selection
Convert Image to Grayscale	Cut	Remove Layer	Select None	Clone	Anchor Floating Selection	Redo
Rotate Image	Move Layer	Select None	Remove Layer	Item Visibility	Paintbrush	Rect Select
Paste	Add Layer	Rect Select	Set Preserve Trans	Add Layer	Rect Select	Item Visibility
Rotate	Item Visibility	Item Visibility	Move Floating Selection	Move Layer	Move Layer	Move Layer
Unsharp Mask	Scale Image	Anchor Floating Selection	Drag Anchor	Paintbrush	Item Visibility	Eraser

Table 2. Task sets for the seven clusters of sessions

the largest cluster, the accuracy score is normalized by the number of sessions in the largest cluster.

The set of clusters with the highest final accuracy score is considered the best set of clusters.

From each of the resulting clusters of sessions, we identify the 12 most frequent commands in each cluster, based on the number of sessions that include that command, to create seven different *task sets*—sets of commands that typically appear with one another. Because we are clustering sessions, a

single command may appear in multiple task sets. However, this is a desirable property since we would fully expect that different high-level tasks would share commands.

Table 2 shows the seven task sets produced by applying this technique to the ingimp dataset. Inspection of the task sets suggests particular user activities. For example, the first cluster suggests fairly basic photo manipulation tasks, such as rotating, resizing and scaling. It also includes commands typically used for photo retouching, such as Levels (used to adjust an image’s brightness and color balance), and Unsharp Mask, a filter used to sharpen images. The second cluster’s task set suggests a particular operation: pasting an image into the current document from the clipboard (which results in a floating selection), choosing a place for the pasted image (Move Floating Selection), and then anchoring it to the page (Anchor Floating Selection). In the next section we find that user-supplied activity tags provide a more detailed picture of what users are doing in this cluster.

The third cluster’s task set suggests working with text in an image (evidenced by Text, Add Text Layer, and Move Text Layer), and the fourth cluster’s task set suggests use of the Paths tool (evidenced by Add Path, Add Anchor, and Drag Anchor). The fifth through seventh clusters collectively suggest painting and graphic design tasks, though more specific activities are less clear.

3.6.2 Activity Tag Keywords for Clustered Sessions

To validate the effectiveness of our clustering approach, and to provide further insights into the high-level activities in each cluster of sessions, we examined users’ activity tags. These are optional user-supplied keywords that the user was prompted to enter each time ingimp was loaded, to describe how they intend to use the application. Example tags entered by users include “photo manipulation”, “screenshot editing”, “resizing”, and “Logo creation”.

In total, 608 of the 2906 clustered sessions included activity tags entered by the user. To summarize these activity tags, we broke them into individual keywords and counted keyword occurrences for each cluster. Since many keywords occur in different tenses (e.g., “resize” and “resizing”) or singular and plural (“screenshot” and “screenshots”) we manually stemmed all keywords to their simple, non-plural present tenses (e.g., “cropped” and “cropping” both become “crop”). We also filtered out common stop words such as “for”, “and”, and “the”, as well as “image”, which occurred frequently across all clusters and does not indicate any particular activity. The ten most frequently occurring keywords for each cluster are shown in Table 3.

Our first observation is that, while some keywords such as “photo”, “web”, “crop”, and “correction” occur across clusters, the top keywords for each cluster are relatively distinct. Moreover, keywords that do occur across clusters often occur more prominently in one particular cluster (e.g., “resize” in the first cluster and “screenshot” in the second).

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
(28) resize	(29) screenshot	(6) logo	(8) design	(7) design	(7) create	(11) design
(15) photo	(15) edit	(5) photo	(7) web	(5) correction	(5) correction	(7) graphic
(13) crop	(12) web	(4) text	(6) graphic	(4) graphic	(4) web	(7) draw
(10) web	(11) crop	(4) make	(6) edit	(3) edit	(4) graphic	(6) test
(9) screenshot	(10) resize	(4) edit	(5) texture	(3) create	(4) edit	(5) texture
(9) edit	(6) photo	(3) design	(4) make	(2) web	(4) design	(5) resize
(7) scan	(6) mockup	(3) create	(4) gif	(2) test	(4) creation	(5) create
(6) screen	(6) design	(2) web	(4) correction	(2) retouch	(4) background	(4) edit
(6) correction	(6) cut	(2) up	(3) png	(2) out	(3) photo	(4) correction
(5) quick	(5) up	(2) test	(3) photo	(2) crop	(3) icon	(3) work

**Table 3. Frequently occurring activity tag keywords for each cluster
(occurrence counts in parentheses)**

These keywords appear to fit with our intuitions on the user activities represented by each cluster. For example, cluster 1’s task set includes commands for photo retouching, and it features activity tag keywords such as “photo”, “resize”, “crop”, and “edit”.

The top keywords for cluster 2 also match our interpretation from the task set (pasting an image from the clipboard and then performing some operation on it) and tell a richer story. While cutting and pasting is not mentioned directly, the most common keyword is “screenshot”. This is significant because a common method of taking a screenshot in Windows is to use the “Print Screen” key on the keyboard to copy an image of the screen to the clipboard, and then paste it into an image editing application. In fact, one activity tag for a session in this cluster was “pasting a screenshot”.

Though there are fewer total keywords in cluster 3 than in other clusters, “logo”, “text”, and “design” are featured prominently, which match with the text operations suggested by the cluster’s task set, and suggest the use of text operations to design a logo.

Finally, clusters 5, 6, and 7, whose task sets included direct manipulation tools and suggested painting and graphic design activities, include keywords such as “design”, “draw”, “graphic” and “create”. The keyword “correction” is also featured prominently, which fits with direct manipulation tools such as Paintbrush, Eraser, and Clone (a tool often used to fix small blemishes in images).

In summary, the resulting task sets both appeal to intuition and show good correspondence with users' reported intentions. More importantly, they provide a rich picture of the activities performed by an entire community of users than command counts alone.

3.7 The Typical *ingimp* User

In summary, our analysis of usage data from *ingimp* suggests that the majority of *ingimp* users are not professionals or regular users of image-editing software. Instead, our findings suggest a population of users who occasionally use the software in short, targeted sessions that exercise a small fraction of the application's full functionality. Despite this, many of the activities suggested by our high-level task analysis, such as logo creation, graphic design, and photo correction, are not trivial uses of the software, suggesting that these users do benefit from the extensive capabilities the application provides.

As we discuss in the next chapter, the current dominant interface paradigm for feature-rich applications is a poor fit for this population of users, because it is designed around supporting efficient navigation to individual commands. In contrast, task-centric interfaces are designed to explicitly support users in performing short, targeted tasks using a small subset of functionality.

Our analysis also replicates and extends the previous finding that individual users only learn and use small, personal subsets of the complete application functionality. This suggests that it is important to provide support for performing unfamiliar tasks in feature-rich software, because any given user will be unfamiliar with the majority of commands (and thus tasks) that can be performed in an application. Our task-centric interface approach is also designed to provide explicit support for this use case.

Chapter 4

Task-Centric Interfaces

In this chapter we present a high-level overview of the concept of a task-centric interface. We also position task-centric interfaces in comparison to a selection of other approaches that have been proposed and studied to support the use of feature-rich software.

We start by discussing current interfaces to feature-rich software, before presenting the components of our task-centric interface design.

4.1 Feature-Centric Interfaces

Current interfaces to feature-rich software follow a *feature-centric* organizational paradigm, in which similar commands and features are grouped into menus, submenus, toolbars, palettes, and tabs. The most common of these organizational schemes is the hierarchical menu system, which imposes a taxonomical structure on the application’s feature set.

The main advantage of feature-centric organizational schemes is that they support efficient navigation to individual commands. By grouping similar functionality into broad categories, the search space for finding a particular command is significantly reduced. For example, if a user of GIMP wishes to apply a vertical flip transformation to a layer, they can navigate to the *Layer* menu, the *Transform* submenu, and then activate the *Flip Vertically* command. The menu hierarchy guides the user’s navigation, so they do not need to consider the vast majority of functionality provided by the system (in the above example, the user would encounter only 34 menu items, which is a small percentage of the nearly 500 commands included in GIMP).

In addition, individual menus typically have a stable spatial representation, which allows users to leverage spatial memory to quickly navigate back to previously-accessed items (Hick 1952; Hyman 1953). Recent work has investigated the limits of spatial memory (Scarr et al. 2013), and how feature-centric organizations can be designed to explicitly take advantages of it (Scarr et al. 2012).

Our insight is that while the focus on navigation in feature-centric interfaces is well-suited to supporting the user in performing *familiar* tasks, it is poorly-suited to assisting the user with unfamiliar tasks. For a familiar task, the user already knows the relevant functionality and necessary procedure required to reach their goal, so the main bottleneck to performance is in navigating to the individual commands, which feature-centric interfaces are designed to support. However, for an

unfamiliar task, the user may not know (1) which commands are relevant to the task, or (2) the procedures required for completing the task. Feature-centric interfaces provide limited or no assistance with either of these concerns; the categorical grouping of functionality adopted in these interfaces can separate the commands involved in performing non-trivial tasks, making it more difficult to discover relevant functionality. In addition, the names given to tools and features may not map to the user's personal conception of a task or goal.

The end result is that the user wishing to perform an unfamiliar task is forced to (1) identify relevant functionality by inefficiently hunting through the application's interface, and (2) determine the required procedures to reach their intended goal through trial-and-error and experimentation. In principle, the user could turn to external help resources for assistance, but as we've discussed in Chapter 2, prior work has shown that users are hesitant to do so.

As well, the results of our investigation of the *ingimp* data set in Chapter 3, and existing work analyzing users' command vocabularies, suggest that users typically only learn a small percentage of the functionality available in feature-rich applications. This suggests that most users will be unfamiliar with the vast majority of tasks that can be performed in an application.

Collectively, these findings motivated us to explore an alternative to feature-centric interfaces that could explicitly help users with performing unfamiliar tasks in feature-rich software.

4.2 Task-Centric Interfaces

A *task-centric user interface* can quickly adapt itself to support the user in performing a specific intended task, presenting relevant functionality and the required procedures to reach the task goal. In effect, we are proposing a new way of working with feature-rich software, in which the main problem-solving activities involved in performing an unfamiliar task are explicitly supported by the interface of the application.

4.3 Components of a Task-Centric Interface

There are four main components that work in concert with one another to enable a task-centric interface design:

Tasks as First-Class Objects – In a task-centric interface, specific goals that can be achieved in the application (i.e. tasks) are represented as first-class objects within the interface. Contrast this with feature-centric interfaces, where commands are named entities, but tasks are not explicitly given

names or represented as objects in the interface. Providing an explicit representation for tasks enables the interface to play an explicit role in assisting the user with reasoning about or attempting to achieve specific goals in the application.

Task Awareness – Second, the task-centric interface includes a mechanism by which it can be made aware of the user’s current task. This could be through an explicit *task declaration* mechanism that enables the user to specify an intended goal, or an implicit *task inference* mechanism by which the system anticipates the user’s goal. Extending our discussion of tasks as first-class objects, task declaration or inference mechanisms provide a way to specify which task object is currently in use. This enables the interface to react to changes in the user’s current task.

For the purposes of this dissertation, we focus on systems in which the user declares their intended task, but it would be interesting to investigate task awareness mechanisms more broadly.

Task Adaptation – Third, the task-centric interface uses the user’s current task as an organizing principle for presenting the lower-level commands and features available in the application. That is, the interface restructures itself to provide explicit, step-by-step scaffolding for task completion, presenting relevant functionality in a customized interface that can be directly interacted with to perform the required actions.

Task Knowledge – In addition to organizing lower-level functionality around specific tasks, the task-centric interface provides additional information to help the user to understand how commands and features relate to achieving specific, meaningful goals in the application. This could take a range of forms, from step-by-step instructions on how to perform a task, to theory behind how a task is performed in an application, to tips or advice on using individual commands in particular scenarios. This enables the interface to provide additional guidance over and above that provided by task adaptation.

In addition to including the four components listed above, we have four additional design desiderata for task-centric interfaces:

Provide universal task support – With task-centric interfaces, we are proposing a new way of accomplishing tasks in feature-rich software. Thus, the mechanisms used for task representation, awareness, adaptation, and providing task knowledge should all be flexible enough to express any task that a user may wish to perform. For example, providing guidance by highlighting relevant interface elements could indicate how to perform some tasks, but would be insufficient to express

other tasks, such as those that involve canvas interactions. In contrast, a combination of highlighting interface elements and textual instructions could provide more universal task support.

Support specific tasks, rather than general task classes – In order to provide specific guidance to the user, the tasks supported by the interface should involve reaching particular goals. This differentiates task-centric interfaces from approaches to organizing functionality used in the Ribbon interface, or Perspectives in the Eclipse IDE, which seek to provide specialized interfaces to support large classes of related tasks (e.g. “Page Layout”, or “Debugging”). We discuss this distinction in greater detail later in this chapter.

Provide access to a large number of tasks – Providing support for specific tasks means that a task-centric interface must be able to support the user in performing a potentially unbounded range of different tasks. In fact, the number of tasks for a given application will likely be exponentially larger than the feature set of that application. This issue of scale needs to be taken into account in designing the mechanisms for task representation, awareness, adaptation, and providing task knowledge.

Enable rapid switching between tasks – Finally, because we are providing support for specific tasks, we would also like the system to support rapid switching between tasks, to make it easy for the user to reach complex goals by performing multiple tasks in sequence.

4.4 Comparison with Existing Personalizable Interface Approaches

Task-centric interfaces can be considered to be an interface personalization mechanism, which puts them in the same class as adaptable, adaptive, and level-structured interfaces. However, as discussed in Chapter 2, the approach taken by task-centric interfaces is distinct from these existing interface personalization mechanisms. In this section, we describe how task-centric interfaces draw on ideas from existing interface personalization mechanisms to create a unique approach.

Existing work on interface personalization has considered how to optimize feature-centric interfaces on a per-user basis, but still maintains the overall feature-centric focus on supporting efficient navigation to individual commands. The interface customizations enabled by these techniques present a smaller subset of commands tailored to the individual *user*. This has the potential to provide an optimal interface for navigating to commands in the user’s personal command vocabulary (i.e. it could potentially be optimal from a feature-centric perspective). However, for any *individual* task performed by the user, the interfaces provided by these techniques may not be optimal, because they represent an aggregation of the functionality required by the various tasks that

the user performs (Figure 5a). In contrast, task-centric interfaces have the potential to provide an optimal subset and arrangement of commands for individual tasks, through task adaptation and enabling rapid switching between task-specific customized interfaces (Figure 5b).

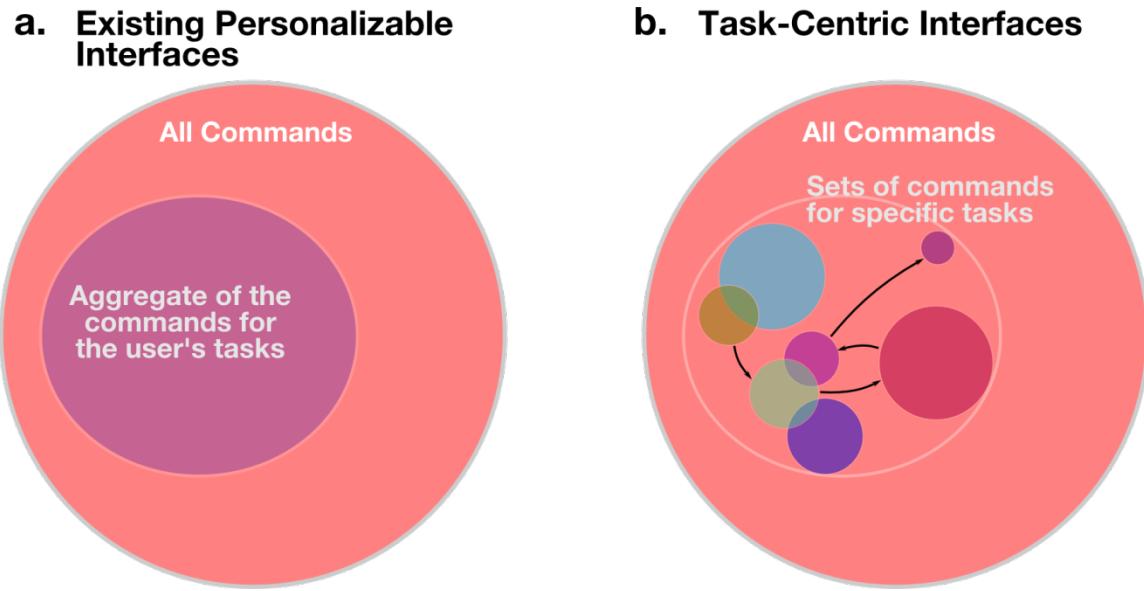


Figure 5. (a) Existing personalization approaches support all of a user's tasks in aggregate, whereas (b) task-centric interfaces support one specific task at a time, and enable rapid switching between tasks.

Another way of looking at this difference is that the primary unit of concern in existing interface personalization techniques has been the user. Task-centric interfaces instead make the user's current task the main unit of concern.

Despite this important difference, task-centric interfaces share a number of similarities with existing approaches to personalization. Like level-structured interfaces (Shneiderman 2002), in a task-centric interface the user is given a mechanism for choosing from a pre-defined set of customized interfaces. However, the set of customizations is much larger (on the order of the number of tasks that can be accomplished in the application). Like adaptive interfaces, the system is modeling the needs of the user and then initiating changes to the interface to help them meet those needs (the system takes the user's task declaration, and responds with task adaptation and guidance). However, at least in task-centric interfaces with an explicit task declaration mechanism, the user initiates this adaptive behavior. Finally, like adaptable interfaces, the user can be seen as the initiator of the change to the interface. However, the user does not need to specify the fine details of how the interface is customized.

4.5 Comparison with the Ribbon and Perspectives

Task-centric interfaces also share conceptual similarities with several techniques that have been introduced in commercial applications. In particular, we compare task-centric interfaces with the Ribbon interface in Microsoft Office, and Perspectives in the Eclipse IDE.

The Ribbon, introduced as part of the Fluent User Interface in Microsoft Office 2007 (Figure 6), provides multiple tabs, each consisting of toolbar made up of buttons, and less frequently, other interactive controls such as drop-down lists or numeric entry fields. Each toolbar is broken up into labelled, visually distinguishable areas (e.g. the Home tab is subdivided into Clipboard, Slides, Font, etc.), creating a hierarchical organization of functionality.

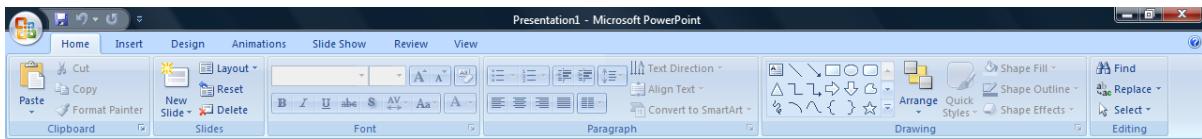


Figure 6. The Ribbon interface from Microsoft PowerPoint 2007.
Used with permission from Microsoft.

One of the goals behind the Ribbon design was to present features of the application based on the functionality they provide, rather than simply as commands (Harris 2008). This has the effect that individual tabs and areas within tabs roughly correspond to classes of tasks. For example, the Insert tab includes commands for inserting various types of objects into a document.

The specific organization of functionality into tabs for Office 2007 was developed through a complex, data-driven process that included use of card-sorting studies to determine where users felt that individual commands should fit in, and analyses of transitions between commands in usage data, in order to minimize extra clicks involved in switching tabs (Harris 2008). This extensive effort was required because the Ribbon seeks to provide a single consistent location for all of the functionality in the system (Figure 7a).

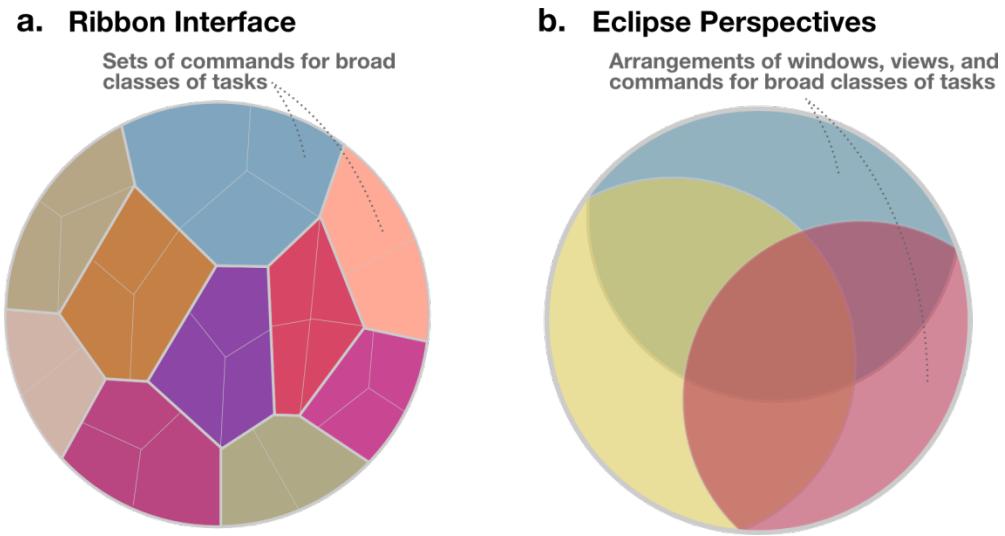


Figure 7. (a) The Ribbon interface subdivides the entire feature-set into a set of tabs, which correspond to broad classes of tasks. **(b)** Perspectives present arrangements of windows, views, and commands for broad classes of tasks, which overlap with one another.

This highlights a key difference between task-centric interfaces and the interface provided by the Ribbon. In the Ribbon, individual tabs, and areas in tabs, correspond to large and fairly general classes of tasks. In contrast, the task adaptation provided in task-centric interfaces provides interfaces for specific tasks. In addition, task-centric interfaces include task knowledge embedded in the interface, which can provide guidance on how to use commands to complete specific goals. The hierarchical organization based on functionality that is provided by the Ribbon has the effect of providing some implicit guidance, in particular for tasks that involve one command, but does not provide guidance for more complex multi-command tasks.

A similar set of differences separate task-centric interfaces from the Perspectives interface provided in the Eclipse IDE. In Eclipse, a *perspective* provides an arrangement of the windows in the application's user interface for performing broad categories of tasks. For example, the Debug perspective contains views that you might use while debugging a program, such as those for setting a breakpoint, viewing the value of a variable in running code, and viewing console output. Perspectives can be explicitly selected from a menu or are automatically loaded in response to certain actions (e.g. the Debug perspective loads automatically when running a program from within Eclipse). Unlike the Ribbon, individual perspectives often include large overlap with one another (Figure 7b). For example, nearly all perspectives include a code editor.

The major differences between task-centric interfaces and Perspectives are that task-centric interfaces provide task adaptation of the interface for specific tasks, and task knowledge that guides the user in how to accomplish specific tasks. In contrast, the task adaptation provided by Perspectives is at the much coarser granularity of large sets of related tasks, and the guidance provided is implicit in providing a relevant subset of views. Stated in another way, the user who is unfamiliar with how to set a breakpoint, or how to follow the value of a variable through program execution, is unlikely to be helped by the Debug perspective.

4.6 Comparison with Web-Based Tutorials

The idea of task-centric interfaces takes inspiration from the way that web-based tutorials are used to support use of feature-rich software. A number of studies have shown that users turn to web-based search engines to support their use of feature-rich software (Ekstrand et al. 2011; Grabler et al. 2009b; Kong et al. 2012; Lafreniere et al. 2013; Fourney, Mann, and Terry 2011a). This strategy is compelling because the user can express their high-level intent in a keyword search using natural language, then (in the ideal) be directed to a web page that describes how to achieve the desired result, step-by-step. This amounts to a declarative, task-centric approach to achieving the desired goal; the user declares what they wish to accomplish (through a web search) and is directed to web-based instructional resources that provide step-by-step instructions for attaining the desired result. These web-based instructional resources can be viewed as selective lenses onto the interface, focusing on the functionality necessary to achieve a desired outcome, and thereby providing a kind of external adaptation of the interface for particular tasks.

Task-centric interfaces bring explicit support for this practice into the application itself, and provide an interface that can be directly interacted with to complete the task at hand. Providing this functionality in-application is important for two reasons. First, as we discussed in Chapter 2, users are hesitant to spend time on concerns that don't directly advance their current goals. Second, prior work has demonstrated a number of sources of difficulty that can come from following external instructions (Knabe 1995; Lau et al. 2004): When instructions are presented in a separate window, the user is forced to switch back-and-forth between the application and documentation over the course of performing a task. The cost of switching, in addition to occlusion of the documentation window by the application, forces the user to read and remember instructions before trying to apply them in the application. This can cause users to get out of step with the instructions and miss steps. In addition, users often mistake illustrations of UI elements in external instructions for functioning interface

elements that they can interact with, or experience difficulty finding components in the interface based on the descriptions or pictures in procedural instructions.

Finally, individual web-based tutorials vary widely in the style of presentation, and the amount of expository text they include. In contrast, we envision task-centric interfaces as providing a more consistent form for presenting task knowledge.

4.7 Our Investigation of Task-Centric Interfaces

In this dissertation, we focus on presenting the overall concept of a task-centric interface, and investigating the potential impact of this approach from the perspective of the end-user. Another important area for research into task-centric interfaces is to investigate how a suitable collection of task-specific knowledge could be created for use by such a system. Though this is not our main focus, we discuss some potential approaches for this in Chapter 9.

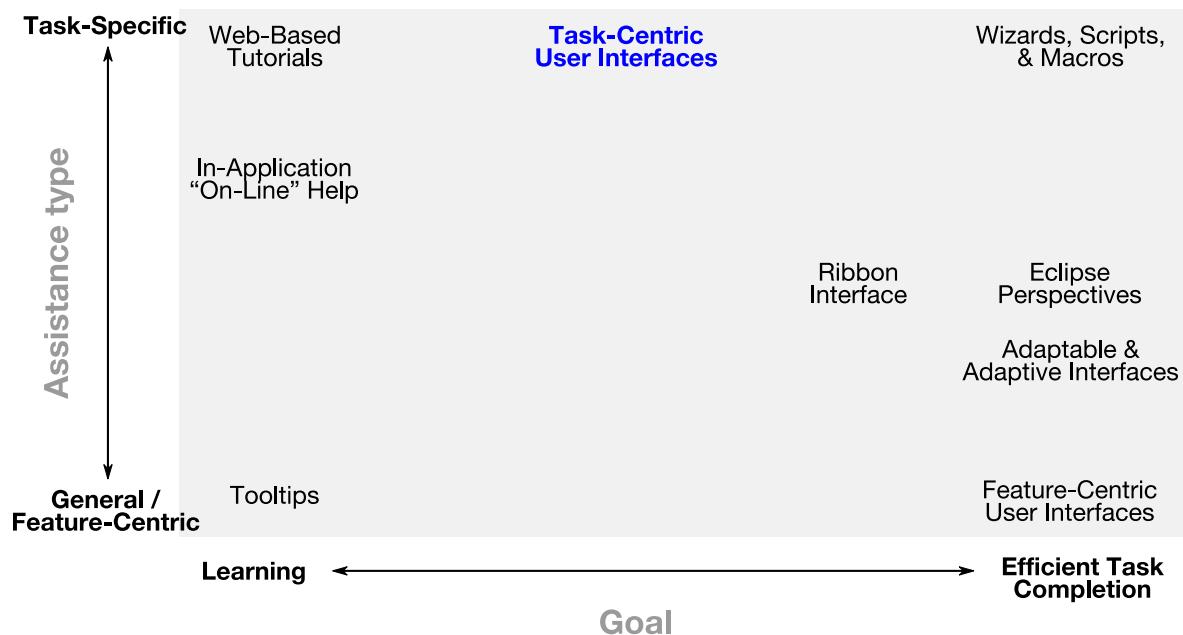


Figure 8. Design space of task-support and learning techniques.

4.8 Summary

In this chapter, we've presented the main components of a task-centric interface, and compared task-centric interfaces to a number of other approaches designed to support use of feature-rich software. In the process, we've identified two attributes that separate task-centric interfaces from similar techniques: the *generality* of the assistance provided (ranging from general guidance, to guidance on

how to perform specific tasks), and the *goal* of the support (learning vs. efficient task completion). In Figure 8, we position task-centric interfaces in the design space that is created by these attributes, and compare it to the techniques we've discussed, and other common forms of help and assistance in feature-rich software.

In the next two chapters we turn from the general concept of a task-centric interface to specific instantiations of this idea. In the next chapter we present the design of AdaptableGIMP, the first task-centric interface we designed and studied, which evolved into our Workflows prototype, described in the following chapter.

Chapter 5

AdaptableGIMP

Our investigation of task-centric interfaces, and the design of our Workflows prototype, began with AdaptableGIMP. This was an earlier project to explore the broader idea of *task-centric community customization*, which includes providing a task-centric interface, but additionally addresses how the requisite body of task-centric knowledge for such an interface could be created through contributions by a community of users.

In this chapter we first describe the task-centric interface provided by AdaptableGIMP. We then present the results of a semi-structured interview study, and a think-aloud study conducted to understand how users react to using AdaptableGIMP’s task-centric interface to complete tasks. The results of these studies suggest that participants appreciated the task-centric interface provided by AdaptableGIMP, and suggest specific ways that the task adaptation and task knowledge components of the system could be improved. Based on these insights, we iterated on AdaptableGIMP’s design to create Workflows, which we describe in the next chapter.

As mentioned above, AdaptableGIMP was additionally an experiment in task-centric community customization. To keep the focus on our investigation of task-centric interfaces, we describe the community customization aspects of AdaptableGIMP in Appendix A, including additional insights into the community customization features of AdaptableGIMP from both the semi-structured interview study presented in this chapter, and a ten-month public deployment.

5.1 AdaptableGIMP’s Task-Centric Interface

AdaptableGIMP is a modified version of the GNU Image Manipulation Program (GIMP). The most prominent user-visible change is that the traditional GIMP toolbox has been replaced with the toolbox shown in Figure 9.

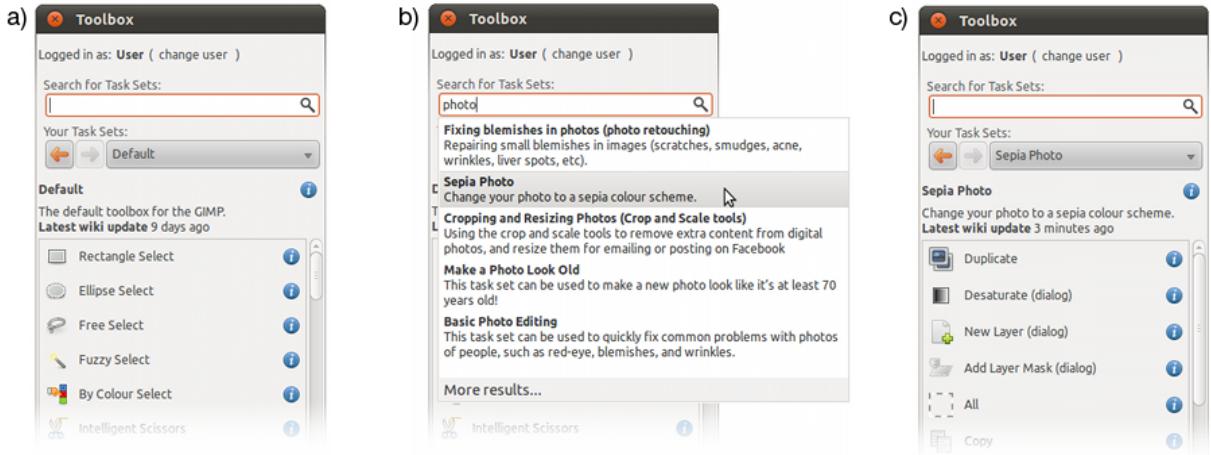


Figure 9. The task-centric interface provided by AdaptableGIMP. The user (a) enters keywords describing their intended task, (b) selects a *task set* from the set of results, and (c) the interface displays an ordered list of commands for the selected task.

In the task-centric interface provided by AdaptableGIMP, the user declares their intended task by entering keywords into a search box (Figure 9a). The system responds with a list of names and short descriptions for *task sets*—ordered sets of commands for performing specific tasks. The user selects a task set (Figure 9b) to install and load it into the toolbox (Figure 9c). They can then use the commands in the custom toolbox to perform their intended task.

For more detailed information on how to perform a task, each task set is associated with *task set documentation* (Figure 10), which can be loaded within the application by clicking the Info button next to the title of the task set.

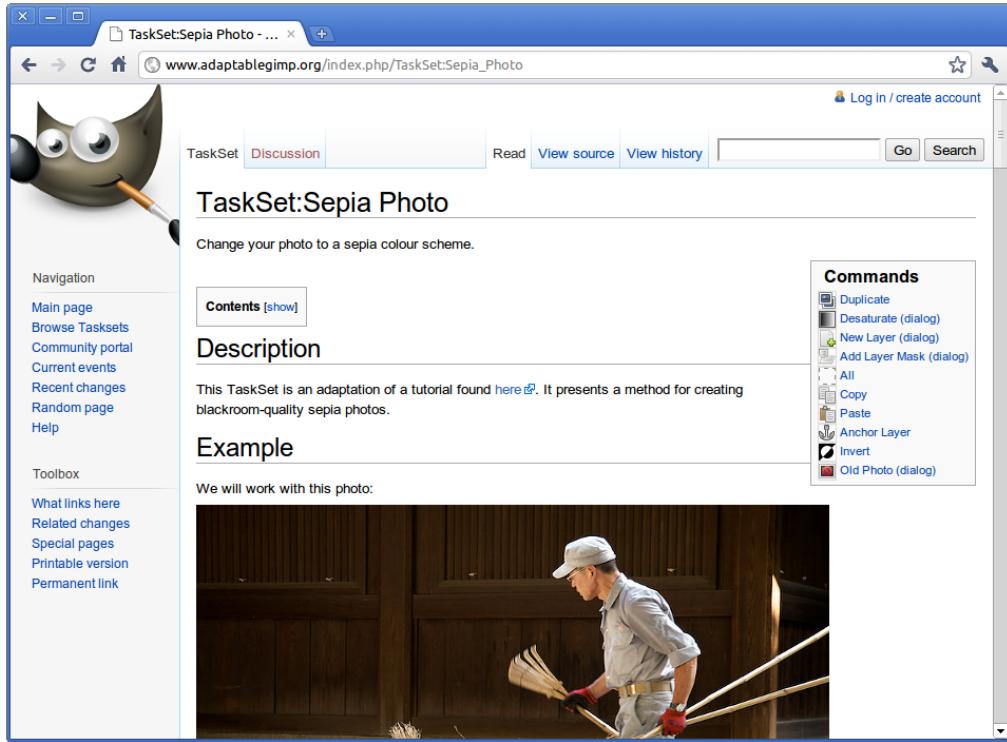


Figure 10. Task set documentation is stored on the AdaptableGIMP wiki, and can be viewed in a web-browser or a window in-application.

We now describe how AdaptableGIMP, by supporting the interaction described above, implements the four components of a task-centric interface that we introduced in the previous chapter:

Tasks as First-Class Objects – In AdaptableGIMP specific tasks that can be performed in the application are represented by task sets, each of which consists of a name, a short description, and an ordered set of commands.

Task Awareness – The user explicitly declares their intended task by searching for and installing a task set using keyword search, or by selecting from the set of previously installed task sets.

There are several features of keyword search that make it particularly well-suited as a mechanism for declaring a task. First, users performing an unfamiliar task may not know the domain-specific terminology associated with that task. Keyword search allows the user to express their intent in language that is natural to them, with the system producing the customization that it believes to be most relevant. Like a search engine, the system could also learn better associations between users' keywords and customizations, and thus continually improve its ability to direct users to relevant customizations.

Second, keyword search naturally handles providing access to a large number of items. This is important because there are a potentially unbounded number of tasks one might wish to perform in a feature-rich application.

Finally, keyword search doesn't require there to be an overarching taxonomical organization scheme on customizations (such as those used to organize commands in hierarchical menu systems). This is good, because it is unclear how one would go about classifying all the specific tasks one might want to perform in an application.

Task Adaptation – The set of commands displayed for the currently loaded task set act as an adaptation of the interface for a specific task. By indicating the commands involved in the task, and the order in which they are used, they provide step-by-step scaffolding for task completion.

Customization of an interface can be subtle or drastic. At one extreme, some applications offer customization capabilities that allow a user to completely redesign the application's interface. Applied to tasks, this approach could, theoretically, provide the user with an optimal interface for performing a given task. However, this approach comes with a number of potential disadvantages as well. First, it imposes a cost on the user when switching between customizations, because they would need to orient themselves to a completely different interface. This could act as a disincentive to using unfamiliar customizations, and could also disrupt the seamless movement between customizations to perform complex tasks. Second, creating such extensive customizations is akin to designing a user interface in its own right, which is a challenging task for professionals, let alone end-users. Finally, previous work has shown that users to be hesitant to remove features from an interface (McGrenere and Moore 2000). With all of the above as motivation, in AdaptableGIMP we opted for a more modest form of customization. In particular we:

Provide a customizable region within the full interface of the application. This is akin to the “dual interface” design advocated by McGrenere et al. for adaptable interfaces (McGrenere, Baecker, and Booth 2007).

Constrain customizations to a simple form that conveys the essence of tasks in the application. This is intended to make it simple for users to orient themselves in the interface when switching between customizations, and to make it easier for users to author customizations, while still retaining enough expressive power to communicate the range of tasks that can be performed in the application.

Task Knowledge – The order in which commands are presented in a task set provides some implicit task knowledge, but the main way that task knowledge is provided in AdaptableGIMP is through the task set documentation linked to individual task sets.

Task set documentation is stored on a wiki page associated with each task set. We selected this medium for several reasons:

The medium is rich, allowing for visual representations including images of documents and interface elements. Past work on the design of tutorials has demonstrated the importance of visual representations for demonstrating techniques in software applications (Grabler et al. 2009a).

Text content can be easily indexed algorithmically. As discussed above, using keyword search as a task declaration mechanism has the benefit that the user can express their intended goal using the language that is natural to them, and the system can infer the task the user wishes to perform, and serve appropriate customizations. However, this is only possible if the system is able to index and model the association between keyword searches and appropriate customizations. Using a text-based medium for task knowledge naturally supports this, because the problem of using keyword search to retrieve text documents is well-studied.

Wiki pages support collaboration among a large number of users. While other mediums, such as video, have been demonstrated as useful for providing succinct demonstrations of actions in user interfaces, text-based content is significantly easier to collaboratively create and refine. More specifically, wikis already provide a rich collection of tools to support collaborative refinement of content by a large number of users. We discuss the issues surrounding collaboration on task sets and their documentation in greater detail in Appendix A.

5.2 Semi-Structured Interview Study

In this section we present data gathered from a laboratory study designed to probe reactions to the idea of task-centric interfaces in general, and to the particular features included in our AdaptableGIMP design.

5.2.1 Author's Note on Contribution

I was involved in analyzing and interpreting the data gathered in this study, but the study itself was designed by Michael Terry and Andrea Bunt, and carried out by Andrea Bunt and Matthew Lount at the University of Manitoba.

5.2.2 Method

Ten participants ranging in age from 18–28 (all male) were recruited from the University of Manitoba campus and through an online community forum. The study followed a between-subjects design with three conditions. In one condition, participants used AdaptableGIMP with a task set that had only minimal supporting documentation (i.e. no instructions). In a second condition, they used a task set with a fully developed tutorial. We chose these two conditions to allow participants to experience AdaptableGIMP in the ideal case, but also in a plausible scenario of use (i.e. when there is incomplete documentation). In a third condition, we asked participants to complete a task with the stock version of GIMP, to provide participants with a baseline to compare AdaptableGIMP to.

Participants performed the following tasks: (1) cleaning up an image by reducing red-eye and wrinkles; (2) aging a photo; and (3) emphasizing a region in a photo by desaturating the rest of the image. For each task participants were provided with a before and after image and a text-based description of the differences between the two. To account for potential order effects, the order of tasks and conditions was counterbalanced using a Latin square.

The study procedure was as follows. Participants were given a brief introduction to AdaptableGIMP. They then completed the three tasks listed above, each using a different version of the system.

Tasks with AdaptableGIMP were completed in two phases. In the first phase, participants were asked to search for and install the task set that they felt best matched the task. In the second phase, they used the task set to complete the task. If after the first phase participants had not selected a task set suitable for the task, the experimenter would install one for them. This decision was made after pilot testing revealed that participants who did not find a useful task set would immediately switch to the full interface and thus would not be able to comment on AdaptableGIMP’s utility.

After performing all three tasks, participants completed a questionnaire and participated in a semi-structured interview. Each session lasted approximately 1.5 hours.

In this chapter, we discuss what our study revealed about how participants respond to the task-centric interface provided by AdaptableGIMP. In Appendix A, we discuss some additional findings from this study, regarding participants’ responses to the community collaboration features of AdaptableGIMP.

5.2.3 High-level Reactions

Overall, participants reacted positively to the task-centric interface concept in general, and to AdaptableGIMP in particular. In response to the question, “If you could choose only one version of GIMP to continue using, which would it be?”, nine of the ten participants indicated that they would choose AdaptableGIMP. The one participant who preferred to install the stock version of GIMP stated that he preferred his interface to remain constant, despite appreciating a number of AdaptableGIMP’s features.

Participants’ preferences for AdaptableGIMP are supported by their answers to a number of 7-point Likert-scale statements (summarized in Table 4) as well as their interview comments. In the sections that follow, we present a deeper analysis of participants’ reactions to AdaptableGIMP’s features.

Positive phrasing of statement	Mean	SD
S1. <i>The task sets were helpful.</i>	6.2	0.78
S2. <i>It was NOT difficult to find the right task sets.</i>	4.5	1.86
S3. <i>It was NOT difficult to switch between task sets.</i>	6.5	0.53
S4. <i>The tutorials associated with the task sets were helpful.</i>	4.7	1.88
S5. <i>Having the smaller set of commands in each task set was easier than using the full GIMP interface.</i>	6.3	0.83

**Table 4. Participant responses to 7-point Likert-scale statements
(1=Strongly Disagree, 7=Strongly Agree)**

5.2.4 Task-Centric Interface Utility

In probing participants’ perceptions of task sets, the questionnaire revealed that participants found the task sets to be helpful (S1: Mean 6.2, SD 0.78) and not difficult to manage (S3: Mean 6.5, SD 0.53). Participants also seemed to particularly appreciate being able to work in a streamlined interface (S5: Mean 6.3, SD 0.83). The following quotes provide further insight into participants’ impressions of the task-centric interface provided by AdaptableGIMP:

All the menus are horrible in GIMP. There are lots of tools available to you, but they don’t... I couldn’t find Blur, and it was like the most obvious thing to find. It’s just they’re hard to use and they’re a mile long. So [AdaptableGIMP] summarizes for you, which is beautiful. [P1]

I could definitely see that the [task sets] would be helpful for a lot of people. ...The main thing that I noticed was, well definitely streamlining the interface. Less buttons in the area made it easier to

find what I needed. I really liked having the buttons for the filters and the effects in the [task set tool palette] because it is really annoying having to search through the drop down menus all of the time. [P5]

Having all of the tools together let me know I was on the right track—I knew what tools to use. [P9]

These comments suggest a frustration with the existing feature-centric interface provided in GIMP, and suggest an appreciation for the ability of AdaptableGIMP's task-centric interface to reveal relevant functionality.

Participant also voiced support for AdaptableGIMP providing customization on the granularity of tasks, particularly for photo manipulation work, which they felt naturally decompose into distinct tasks:

You need a certain set of functions for say doing portrait photography, as opposed to a different set of functions for doing Astra photography, which is completely different. I liked that you were able to group them—just the things that you need—into a smaller subset without really losing access to the broader set of things that are available in the program overall. [P4]

Each photo manipulation can be broken down into a series of tasks. And saying, you know, “I need to remove red eye,” “I need to add a Gaussian blur,” for instance, to make a picture look old. It was trivial to search, to install, and to then start using [task sets], mostly because of the fact that the tasks can be set up into a series of steps. Whereas if you are just using the default GIMP set it was “OK, what do I do now?” because [the commands are] grouped by what they change. I liked that [task sets] made photo manipulation easier, and it also did some of the work for you, in that “OK, now I do this, then I do this.” [P2]

P2's comment above speaks to a task set's ability to provide assistance beyond simply indicating relevant functionality. In particular, it demonstrates that the ordered sequences of commands were useful guides when completing a task.

5.2.5 Task-Set Documentation

While participants were enthusiastic about the task adaptation provided by task sets, only four of the ten participants utilized the documentation included with them. As well, some participants expressed that they didn't find the task set documentation as useful as the ability to work in the streamlined interface, which itself provides a kind of guidance.

This sentiment was not universal, however, and some participants did express appreciation for being able to access additional information within the application itself:

It takes the problem of searching out of Google and brings it into the program itself, which is quicker, and providing the resources immediately. It's definitely, definitely beneficial. [P7]

Collectively, these results suggest that task knowledge in application has value, but that the manner in which it is presented or accessed in AdaptableGIMP may need to be tweaked for users to take advantage of it.

5.2.6 Searching for Task Sets

Participants also had variable responses when describing how easy it was to search for the right task set out of the roughly 30 available during our study. Some participants were very enthusiastic about the keyword search interface:

Live search for me... warms my heart whenever I see it because, at least right now, it seems to be trivial to do. It was very easy and very handy. [P2]

It was easy, really. I expected it was going to be harder, but it was easy for me. [P9]

Others expressed some difficulty in coming up with the right terminology to describe a task. In the following quote, P7 describes how coming up with the right search terms can be difficult when one is unfamiliar with how to accomplish a task:

The specifics may have more to do with my familiarity with the topic than anything else. I don't do a whole ton of photo manipulation. So it takes me a minute or two to remember exactly how it is supposed to work. [...] But if it were something more specific, I don't think it would be that bad at all. [P7]

P7's difficulty in finding the right search terms may indicate that we could do more to assist users with declaring their intended task. In our study of Workflows (Chapter 7) we gain a deeper understanding of the source of difficulty for users, and suggest some ways it could be addressed.

5.2.7 Summary of Findings

The results of our laboratory study provide encouraging evidence for the utility of customizing interfaces based on tasks. Participants expressed that it was easier to use a small set of commands than the full GIMP interface. They also indicated that the customized interfaces provided guidance on how to complete tasks, and expressed appreciation for the keyword search mechanism for locating relevant customizations. However, fewer than half of participants accessed the documentation associated with task sets, and some expressed that they did not find the documentation as useful as the ability to work in a streamlined interface.

5.3 Think-Aloud Study

To confirm the findings from the semi-structured interview study, and to gather further insights into where the task-centric interface components of AdaptableGIMP could be improved, we conducted a usability study with five participants using a think-aloud protocol. The intent of this study was to elicit participants' initial reactions to using AdaptableGIMP's task-centric interface to complete tasks, and to gain insights into how we could improve its design for this purpose.

5.3.1 Method

Each of five participants (2 female, 3 male) was asked to "think aloud" as they performed several image editing tasks using AdaptableGIMP. The experimenter observed the participants as they worked, and occasionally asked the participants to comment on their problem solving process and their reactions to features of AdaptableGIMP. The tasks that participants were asked to perform in this study were Drawing Rectangles, Making a Photo Look Old, and Selective Colorization (keeping one element of an image in color, while converting the rest to greyscale). Participants included two members of the lab where AdaptableGIMP was developed (who were not involved in the project), and three members of another lab in the same department. Thus, it is best to think of this study as "person down the hall" testing.

5.3.2 Results

Our main observations concerned locating task sets, using task sets to perform tasks, use of supplementary documentation, and reactions to replacing the stock GIMP toolbox.

5.3.2.1 Locating Task Sets

We observed that participants were generally able to quickly figure out how to use the keyword search functionality to locate a relevant task set. However, one participant initially ignored the keyword search because he felt that it only searched the set of locally installed task sets. Another participant noted that he was not confident in his ability to formulate keywords, and attributed this to his being a non-native English speaker.

These observations suggest that the keyword search interface in AdaptableGIMP could be improved to better communicate the overall intent of the system, and to better support users in formulating search queries.

5.3.2.2 Using Task Sets

Our observations also suggest that the current design of task sets is not adequately communicating that commands are presented in the order required to complete a task. This can be seen in the following quote from a participant, discussing the commands in a task set:

It's just a bunch of commands that you could use towards generating an image, so whether I select it all first, then desaturate, then copy and paste. I don't know, it's all mumble jumbled. I wouldn't imagine that they were in steps. [P3]

Another participant expressed that the commands seemed to be in the right order for the task he was currently performing, but then revealed that he is uncertain as to whether this is true for task sets in general:

You're going to duplicate it, flip it, skew it, put some sort of gradient on top of it. [Looking over task set] Oh yeah, Flip, Shear. I can see that they're in order of the sequence I need to apply them. I don't know if that's a coincidence, but that's really helpful if it's in the order. [P5]

We observed that this misunderstanding was a source of difficulty for participants. For example, P5 spent a large amount of time on the Old Photo task experimenting with the commands included in the task set in no particular order, and was ultimately unable to recreate the goal image.

These observations indicate that we can do more in our design to indicate that commands are included in a particular order for performing the task.

5.3.2.3 Use of Task Set Documentation

Related to difficulties understanding the sequence of operations from a task set, only one participant opened the documentation associated with a task set without prompting. This is consistent with findings from our previous laboratory study.

Participants gave a number of reasons for not accessing the documentation, including that they tend to disregard help (P2, P3), the placement of the Info icon used to access documentation was confusing (P1), or that they figured documentation would simply contain unhelpful concatenations of help for the individual commands in the task set (P5).

However, once P2 was shown the documentation for the task set, he commented on its importance:

So, if I'm assuming that I'm learning this [task] for the first time, and I'm using this because I'm learning it, then this is the really important thing, this button right here [the Info button], because this actually explains this stuff to me. [P2]

After directing P3 to open the documentation for a task set, she expressed that it would have been useful to open the documentation at the start of performing the task, and that the documentation included the steps she needed to complete the task:

That's what I expect that I should have done in the first place, but I never looked under 'I' [the Info button] (laughs). When I first opened it, these were the steps I was wanting to do, after you told me what my task was. [P3]

These observations indicate that the interface is not adequately communicating the availability of task set documentation and how to access it. Moreover, the similarity with traditional help systems appears to be acting as a disincentive for users to access and use the documentation.

This finding is particularly significant because, as we discussed above, it does not appear that the sequence of actions to complete the task is adequately communicated by the ordered presentation of commands alone. The task set documentation could help with this, but only if it's viewed by the user.

5.3.2.4 Replacing the GIMP Toolbox

Finally, a frequently raised concern among participants was that the AdaptableGIMP panel had replaced the stock GIMP toolbox. While AdaptableGIMP includes a Default task set with the tools from the stock GIMP toolbox, participants expressed concerns that accessing this would require switching back and forth between task sets, and that this would make it difficult to perform additional operations not directly related to their current task. For example, P4 stated:

This might be also helpful if it didn't change the original interface. If I keep having all the tools here, plus the specific tools for the task, that might help. [...] I might want to do some other stuff while doing the specific task. [P4]

P2 also suggested including AdaptableGIMP features in a separate dialog was one way to address this problem,

Well, OK, if you put [the customization features] in a separate dialog then I would have all of these tools available and I wouldn't have to switch between the two. [P2]

In addition to the time it would take to switch between task sets, this feedback may indicate that users are uncomfortable with functionality being removed from the interface. This issue was raised by one participant in our previous laboratory study, and has also been previously identified in work on interface customization (McGrenere and Moore 2000).

5.4 Summary

In this chapter, we've presented AdaptableGIMP, a modified image editing application that we developed to investigate the ideas of task-centric interfaces, and task-centric community customization.

In terms of task-centric interfaces, we discussed how AdaptableGIMP includes the four components of a task-centric interface that we described in Chapter 4 by supporting an interaction in which the user declares their intended task using keyword search, to install and use task sets—ordered sets of commands with associated documentation.

The results of two laboratory studies provided a number of insights into how users react to this interaction. Specifically:

- Participants appreciated keyword search as a task declaration mechanism, though some participants had difficulty formulating search queries;
- Participants appreciated the ability of the system to indicate relevant commands and provide a streamlined interface for task completion (i.e. task adaptation was appreciated);
- Participants had difficulty determining how to complete tasks based on the ordered list of commands in a task set alone;
- Participants were hesitant to access the documentation associated with task sets (i.e. the form in which task knowledge is presented could be improved); and
- Participants did not like that the AdaptableGIMP toolbox had replaced the stock GIMP toolbox.

In the next chapter, we discuss how we used these insights to iterate on the AdaptableGIMP design and create Workflows.

5.4.1 Contributors to the AdaptableGIMP Project

Throughout its existence, the AdaptableGIMP project has had a large number of contributors. In January 2010 it became my primary research project, under the supervision of Michael Terry and in collaboration with Andrea Bunt. Other notable contributors include Filip Krynicki and Leslie Ng, who were hired as co-op students to work on the project under the supervision of Michael Terry and I; Matthew Lount, a student of Andrea Bunt's who helped run the semi-structured interview study

reported in Section 5.2; Adam Fourney, who contributed to early brainstorming and designing the software architecture for the system; and Jordan Stinson, who worked on an earlier iteration of the design prior to 2010, which is not presented in this dissertation.

Some of the content in this chapter is based on a poster presented at the UIST 2011 conference (Lafreniere et al. 2011).

Chapter 6

Workflows

In this chapter we present Workflows, the successor to the task-centric interface design provided by AdaptableGIMP. Unlike AdaptableGIMP, which was also an experiment in task-centric community customization, Workflows was developed specifically to study the impact of a task-centric interface design on end users performing unfamiliar tasks in feature-rich software.

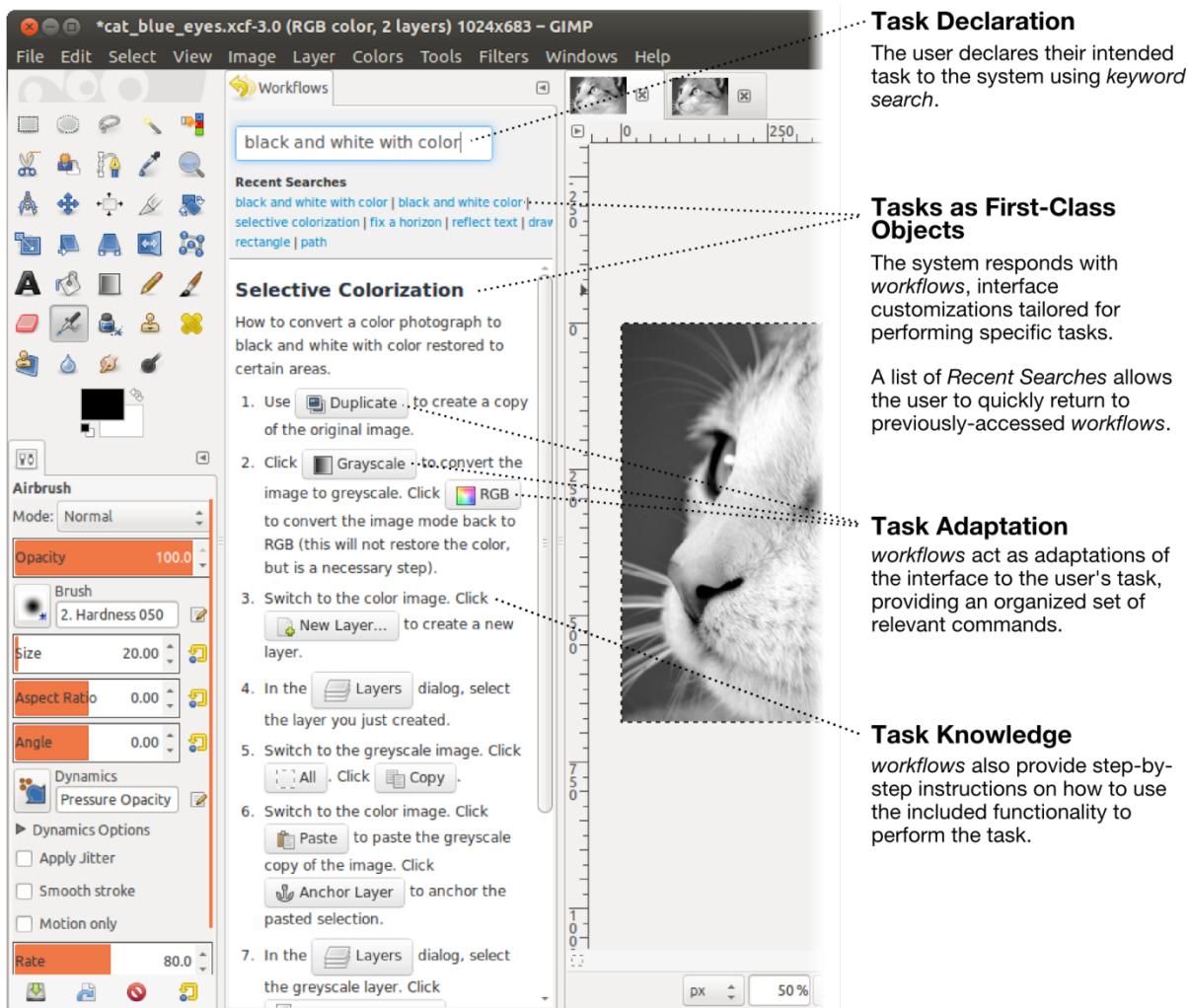


Figure 11. The basic components of a task-centric interface in Workflows.

An overview of the basic components of Workflows is shown in Figure 11. We start by briefly discussing how the interface of Workflows was developed from the task-centric interface provided by AdaptableGIMP, and the feedback we reported in the previous section. We then discuss how Workflows implements the components of a task-centric interface that we presented in Chapter 4.

6.1 From Task Sets to Workflows

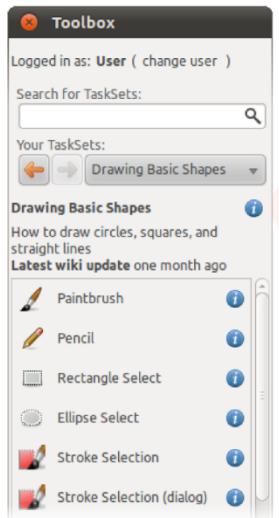
Motivated by the findings from the two user studies reported in the previous chapter, we made four major changes to AdaptableGIMP’s task-centric interface design to develop Workflows. Specifically, we:

- Simplified the interface to consist of a keyword search box and content area;
- Integrated documentation and actionable commands together into *workflows*, a more expressive form of task-specific customization than task sets;
- Removed the step of selecting a search result to install a task-centric customization. In Workflows, a keyword search directly returns step-by-step instructions with actionable references to commands;
- Added a list of “Recent Searches” to provide a quick and self-descriptive interface for returning to previously accessed customizations; and
- Restored the stock toolbox, moving the task-centric interface features into a supplementary panel.

A side-by-side comparison of AdaptableGIMP and Workflows is shown in Figure 12.

The most drastic change in this design iteration is in the form that customization takes in the two systems, which changed from *task sets*, which resemble custom button bars, to *workflows*, which are more similar to tutorials presented in-application. This was motivated by the results of our think-aloud study which suggested that ordered lists of commands can be insufficient on their own to indicate how to perform unfamiliar tasks, and the tendency of participants to not access the supplemental documentation associated with task sets.

AdaptableGIMP



Replaces the standard GIMP toolbox.

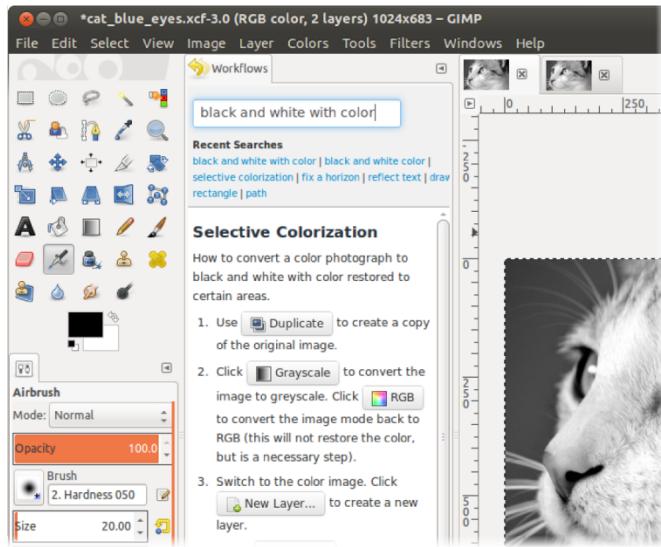
Customizations are *task sets*, similar in form to button bars.

Procedures are provided in supplementary *task set documentation*.

To perform a task, the user:

- Issues a keyword search
- Selects a task set to install
- Uses functionality in the installed task set

Workflows



Presented in its own dialog alongside the GIMP toolbox.

Customizations are *workflows*, similar in form to tutorials.

Procedures are provided in-line in each workflow .

To perform a task, the user:

- Issues a keyword search
- Uses functionality in the returned workflow(s)

Figure 12. Comparison of AdaptableGIMP and Workflows.

6.2 Workflows' Task-Centric Interface

Workflows is a modification of the GNU Image Manipulation Program (GIMP). The primary, visible modification to the interface is an additional pane displayed alongside the traditional toolbox (Figure 11). This pane allows the user to enter keywords describing their intended goal (e.g. “black and white with color”), and receive customized interfaces—*workflows*—that match the declared task. Unlike AdaptableGIMP, there is no extra step to install a customization before using it; relevant workflows

are displayed in the pane below the search box, and can be interacted with immediately. In the case where a search returns multiple matching workflows, they are displayed one after another.

For searches that return at least one workflow, the search keywords issued by the user are added to a list of “Recent Searches” that can be clicked to return to a search. This feature is intended to make it easy for the user to return to a previously used workflow, or to quickly switch between workflows.

Each workflow consists of a title, a short summary describing its intent, and succinct step-by-step text instructions. References to commands, tools, or dialogs are actionable—they can be clicked to execute a corresponding action, or to display a referenced dialog. If a dialog is already visible in the interface, the system will highlight it by briefly flashing its border. With this design, the customized interface directly describes how to perform a task using each of the necessary tools, as well as providing an interface through which to perform the required actions.

Next, we describe how this design implements the four components of a task-centric interface presented in Chapter 4.

Tasks as First-Class Objects – Tasks are represented as first-class objects in two ways. First, each workflow represents a specific task that can be accomplished in the application. Second, the entries in the “Recent Searches” area serve as a record of tasks the user has previously declared an interest in performing, and provide a way to return to the customized interfaces for these tasks.

Task Awareness – The user declares their intended task by issuing a keyword search using the search box, or by clicking a previously performed search in the Recent Searches area.

As discussed in Chapter 5, keyword search enables the user to express their intent using their own personal terminology, with the system attempting to infer relevant customizations, and is also a natural mechanism to provide access to a large, unstructured collection of items, which is the case for the set of tasks that can be performed in an application.

Task Adaptation – In response to a search, the pane below the search box presents a customized interface consisting of one or more workflows. These indicate the relevant functionality for performing specific tasks, in the order it needs to be used to reach the task goal.

Task Knowledge – The specific procedures for using included commands and functionality to complete the task are integrated into individual workflows as short, text-based instructions. Unlike the

supplementary task-set documentation in AdaptableGIMP, no additional effort on the part of the user is required to access this guidance in Workflows.

6.3 Creating Customizations

Because Workflows was created as a prototype system to explore task-centric interfaces from the user's perspective, we did not create an authoring interface for workflows. Instead, workflows are authored by creating an HTML file and adding it to a shared file repository, which is then synced to individual installations of the Workflows system.

6.4 Implementation

Workflows was implemented as a modification to version 2.8.2 of the GNU Image Manipulation Program (GIMP) in concert with an interface developed using HTML5 and JavaScript, and several Python scripts. The following sections describe the implementation in greater detail.

6.4.1 User Interface and Command Invocation

The Workflows pane shown in Figure 11 was implemented as a dockable GIMP dialog consisting of an embedded WebKit web browser widget. The embedded browser widget displays the user interface for the Workflows pane, which was implemented using HTML5 and JavaScript, and styled to appear as a seamless part of the GIMP interface. The HTML5 interface is served using a small Python web server script running on the local system.

To enable the actionable buttons within workflows to invoke GIMP commands, we modified the GIMP source code to add hooks into the JavaScript engine running in the embedded WebKit instance. The modifications would catch calls to a particular JavaScript function invoked when the user clicked the actionable buttons in workflows, and then execute the corresponding action in GIMP.

Embedding an HTML5 interface with a Python backend into GIMP provided a number of advantages when developing Workflows. First, during development it allowed us to rapidly make changes to the Workflows interface without recompiling GIMP, which made it easy to quickly test new ideas and designs. Second, using HTML5 for the Workflows UI provided greater flexibility than we would have had using the GTK+ toolkit provided by GIMP, thus removing constraints on our design. Finally, using Python as a web server backend for the user interface allowed us to use an off-the-shelf Python search engine library to implement the search engine component of Workflows.

6.4.2 Python Server Component

To serve the HTML5 user interface, we implemented a minimal web server using Bottle², a lightweight Python web framework, and cherrypy³, a minimal thread-pooled HTTP server. The server script was run on the local system at the address <http://localhost:8080>.

In addition to serving the HTML5 user interface to Workflows, the server script provided an HTTP API for making keyword search queries to the search engine component. At runtime, when the user performed a keyword search, client-side JavaScript code would make an asynchronous query to the search API and display the returned results.

6.4.3 Workflows Search Engine

The search engine component of Workflows was implemented using version 2.4.1 of Whoosh⁴, a Python search engine library.

To build an index of the library of workflows, Whoosh's StemmingAnalyzer component was used with its default settings. This component tokenizes content using the regular expression “\w+(\.?\\w+)*”, converts all tokens to lower case, removes stop words⁵ and words of fewer than 2 characters, and then performs word stemming on tokens using the Porter stemming algorithm⁶.

Queries were implemented using the standard query method provided by Whoosh with its default settings, which provides a query language similar to that provided by modern web search engines⁷, and ranks documents using the BM25F ranking function (Zaragoza et al. 2004) with B=0.75, K1=1.2.

6.5 Summary

In this chapter, we've described our iteration on the design of AdaptableGIMP to create Workflows. In the next chapter, we present a two-session laboratory study we conducted using Workflows, to investigate our main research questions on the impact of a task-centric interface design.

² <http://bottlepy.org/>, retrieved May 19, 2014

³ <http://www.cherrypy.org/>, retrieved May 19, 2014

⁴ <https://pypi.python.org/pypi/Whoosh/>, retrieved May 12, 2014

⁵ 'a', 'an', 'and', 'are', 'as', 'at', 'be', 'by', 'can', 'for', 'from', 'have', 'if', 'in', 'is', 'it', 'may', 'not', 'of', 'on', 'or', 'tbd', 'that', 'the', 'this', 'to', 'us', 'we', 'when', 'will', 'with', 'yet', 'you', 'your'

⁶ <http://tartarus.org/~martin/PorterStemmer/>, retrieved May 12, 2014

⁷ <http://pythonhosted.org//Whoosh/querylang.html>, retrieved May 12, 2014

Chapter 7

Assessing the Impact of a Task-Centric Interface

In this chapter we present a controlled laboratory study of the task-centric interface provided by Workflows. Our analysis considers Workflows from three complementary perspectives: How it changes the problem solving process for unfamiliar tasks; how it affects performance for unfamiliar tasks; and the advantages and disadvantages of the problem-solving approach it supports over current approaches.

To answer the above questions, our study compares Workflows with a control condition representing current practices for using feature-rich software. The control condition additionally provides a close-up perspective on the strategies used to complete unfamiliar tasks using the current interfaces to feature-rich software, and resources found on the web. This complements our findings from Chapter 3 to give a comprehensive view of current practices for performing unfamiliar tasks in feature-rich software.

Designing a study to evaluate our task-centric interface design was a challenging problem in itself, because the intent of the system is to support an alternative problem-solving *process*. That is, we are designing at a higher conceptual level than interaction techniques designed to optimize performing a particular action in an interface (e.g. command selection techniques). Instead, our problem is similar to that of evaluating command-line interfaces vs. graphical user interfaces (i.e. two interface paradigms that support very different problem-solving strategies). In this setting, the choice of tasks to use in a study could easily bias the evaluation toward one system or the other, and looking at quantitative performance measures provides only so much in the way of insights. It's more important to understand how the two different systems affect the strategies employed by users to complete tasks, and the relative advantages and disadvantages of each.

With these points in mind, the primary objective of the study reported in this chapter is to investigate whether Workflows successfully supports an alternative problem-solving process for performing unfamiliar tasks, through the explicit support it provides for locating relevant functionality and determining the necessary procedures to reach an intended goal.

We also want to test whether Workflows improves performance over existing practices, and evaluate the usability of our current design. To achieve this diverse set of goals, we employ a mixed

methods experimental design that uses both quantitative and qualitative methods to derive a comprehensive picture of participants' behavior in the two conditions.

7.1 Experimental Design

To understand the potential benefits of a task-centric interface, we compared use of Workflows to existing practices, namely, use of the unmodified GIMP interface and web search. We measured task completion time, cognitive load, and subjective preference. We also used qualitative methods to analyze how participants performed the study tasks, and what activities they spent their time on, in order to better understand participants' problem-solving strategies in the two conditions.

7.1.1 Basic Study Design

Our study employed a within-subjects design with two conditions. In the control condition, participants were provided with GIMP 2.8 and a web browser initially loaded to the Google search page. Participants were instructed to use whatever strategies they would typically use (including web search if desired) to complete the study tasks.

In the experimental condition, participants were provided with Workflows and asked to try and use it as their primary method of completing the task. In addition, participants in this condition were not permitted to use the web. We imposed these two restrictions because our goal was to learn how the task-centric interface provided by Workflows would influence problem-solving strategies.

Participants performed tasks on a computer that we provided, within a virtual machine that could preserve state.

To compare the impact of Workflows for both new tasks and infrequently performed tasks, our study consisted of two sessions. In the first session, the participant performed four tasks that were new to them (two per condition). In the second session, held at least two weeks later (median 21 days, min 14, max 29)⁸, the participant returned and performed the same tasks again. The mapping of the two sets of tasks to conditions, the order in which participants experienced the two conditions, and the order of tasks within each set were all fully counterbalanced across participants. Each participant experienced the same task and condition ordering, and task-to-condition mapping, in their first and second study session.

⁸ There were no significant differences in the number of days between sessions for any of our counterbalancing factors described in the Tasks and Procedure section ($p > 0.40$ in all cases).

7.1.2 Method

Each session lasted approximately one hour. The first session began with a short demographic questionnaire. The experimenter then gave participants a brief overview of the major parts of the GIMP interface (the document, toolbox, tool settings, color picker, layers, and undo command). In the first session, a brief overview of the features of Workflows was provided immediately before the participant started this condition. In both sessions, before each condition, participants completed a short practice task to (re)familiarize them with the resources available in that condition.

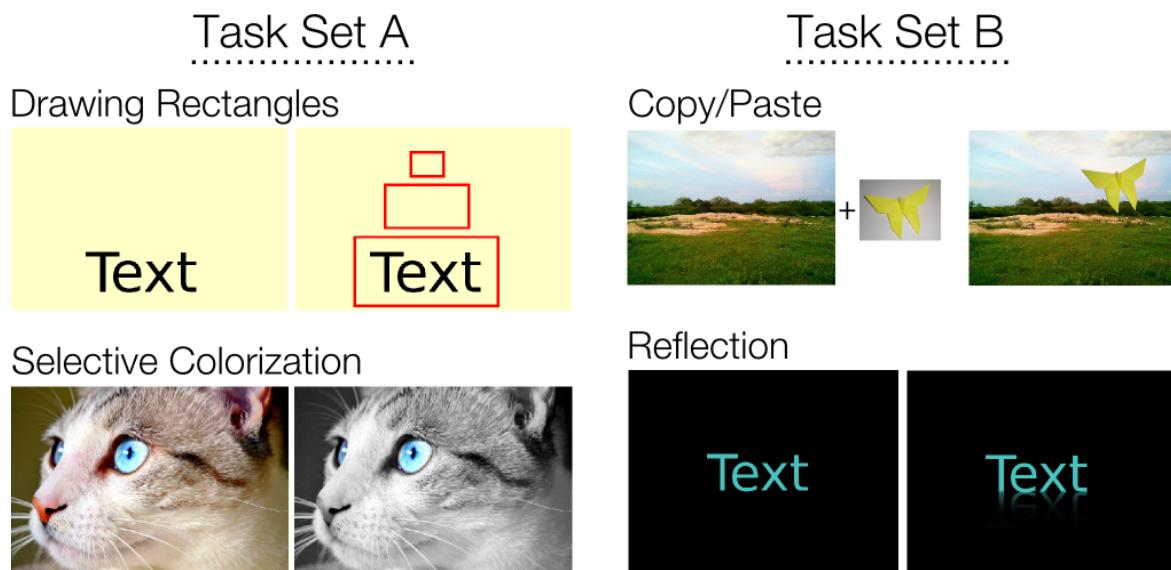


Figure 13. Before/after images for the four study tasks performed by participants.

In each session, participants completed the four tasks shown in Figure 13. Each task requires use of some direct manipulation operations, and so would be difficult to automate (making them representative of the ill-defined tasks that we imagine task-centric interfaces would be well suited for). These particular tasks were also selected to have minimal overlap in commands, to control for learning effects between tasks.

Each task was presented to the user as a before/after image displayed on a second monitor, without any text describing the task to be performed. We opted for this visual presentation of the task to avoid biasing the participant with terminology that they might use to determine what commands to use in the interface, or search terms to use when seeking help resources either on the web or in the Workflows panel. Before beginning each task, the participant filled out a short pre-task questionnaire (Appendix B.1). Following each task, the participant filled out a NASA Task-Load Index (TLX)

questionnaire to measure cognitive load (Hart and Stavenland 1988). At the end of each session, the participant filled out a post-session questionnaire designed to measure their condition preference, and their feelings on the ease of use and overall quality of the resources available to them in the two conditions (Appendix B.2).

At the end of the first session we preserved a snapshot of each participant’s virtual machine to ensure that search histories for both Workflows and the browser were preserved across sessions for each participant.

The procedure for the second session was similar to the first, except that participants were not given a tour of the interface before each condition. Additionally, a semi-structured interview was conducted at the end of the second session to elicit participants’ impressions of both conditions across the two study sessions.

An experimenter was present throughout the study to make observations and to judge when the participant had completed a task and was ready to move on to the next task. If the participant appeared stuck, or declared that they had completed the task when their current document differed significantly from the goal image, the experimenter would ask “Is there anything else you could try?” or “Is there any way you could make it look more like [the goal image]?” The experimenter capped each task at 12 minutes.

Video of the sessions was recorded using a camcorder pointed at the computer monitor, so that we could perform a secondary analysis of participants’ activities during the sessions.

7.1.3 Tasks and Workflow Authoring

To avoid bias that could arise due to our choice of tasks, or our means of authoring workflows for the experimental condition, we developed the following procedure to choose study tasks and create workflows to populate the Workflows system.

We examined data from CUTS (Fournier, Mann, and Terry 2011a) to identify common search queries for GIMP, and from these selected the set of tasks for the study. In addition to the criteria mentioned in the previous section, we selected tasks that were a reasonable length for the study, and that had minimal overlap in commands (to control for learning effects between tasks).

To create workflows for the study tasks, we executed each task’s associated search query and selected the highest ranked web page that contained step-by-step instructions for completing the task

as a basis for creating the workflow. We then used a set of predefined templates and heuristics to create a workflow with the same procedure as the tutorial. The templates in Table 5 were used to create workflows based on web-based tutorials, while preserving the overall procedure of the tutorial.

Template	Example Workflow Text
Switch to <Short Description> image.	Switch to the greyscale image.
Select the <Name> tool.	Select the Crop tool.
Click <Command or Button> [to <Desired Effect>].	Click Duplicate to create a copy of the original image.
[Hold the <Ctrl-, Shift-> key and] [<left, right>] [Click][and drag] to <Desired Effect>.	Hold the Shift key and left click where you want the line to end.
Use the following settings: <Setting: Value pairs>	Use the following settings: - Width: 200 percent - Height: 200 percent
Set the <FG,BG> color to <Color> [and the <FG,BG> color to <Color>].	Set the foreground color to <i>Black</i> .
Use one or more selection tools to <Effect>. <All selection tools>.	Use one or more selection tools to create a selection in the desired shape. Rect Select , Ellipse Select , Free Select , ...
In the <Name> dialog <Do Action>.	In the Tool Options dialog, use the following settings: ...
In the Layers dialog, select the <Short Description> layer.	In the Layers dialog, select the greyscale layer.
Use the <Name> tool to <Desired Effect> [by <Method>].	Use the Text tool to create some text.

**Table 5. Templates were used to create workflows based on procedures found in web tutorials.
Bold text in example text indicates actionable buttons.**

In addition to recreating instructions using templates, vague settings in the tutorial were made explicit (e.g. “Choose a large brush” was converted to “Choose a 150px brush”), and an “(Optional)” flag was added to steps or procedures that weren’t strictly necessary.

The above process was intended to ensure a reasonable degree of equivalence in the procedures available between the two conditions, despite the differences in form between web tutorials and workflows. We note that complete control here is not possible, because the form of workflows is qualitatively different from that of web-based tutorials.

In addition to the four workflows created for the study tasks, we included an additional workflow for a practice task and eight additional workflows for other tasks not tested in the study. These

additional tasks simulated expected real-world conditions where the user would need to locate a workflow amongst many. To facilitate replication of our study, Appendix B.3 includes all of the workflows authored for this study.

7.1.4 Participants

We recruited 16 participants from a university campus (10 male, 6 female) with ages ranging from 21–31 (median 24). Participants were recruited via postings on an email mailing list targeted toward graduate students. Participants were screened to ensure that they: (1) had minimal experience with image editing software, and (2) were native English speakers (to control for variability in strategies that may depend on ability to formulate search queries). All but one participant took part in both study sessions. In appreciation for their time, participants were given a \$10 gift certificate to an online retailer for the first session, and a \$15 gift certificate for the second session.

7.2 Research Questions

Before presenting our results, we revisit the research questions that we seek to answer in this study. A summary of research questions is included in Table 6, along with the measures associated with each.

Research Question	Measure
How does Workflows affect the problem-solving strategies used to complete unfamiliar tasks in feature-rich software?	Qualitative analysis of time spent on different activities in the two conditions. Semi-structured interviews.
Does Workflows yield faster times to complete tasks?	Recorded task times.
Does Workflows allow users to complete tasks with lower cognitive load?	NASA-TLX self-report, administered after each task.
What difficulties do participants experience in the two conditions?	Observations made during the study and secondary review of recordings. Semi-structured interviews.
What is learned in the first session in the two conditions?	Qualitative analysis of time spent on different activities in the two conditions. Semi-structured interviews.
What are participants' subjective impressions of Workflows?	Exit questionnaire. Semi-structured interviews.

Table 6. Research questions and corresponding measures.

These research questions translate into the following set of testable hypotheses. Using a task-centric interface, we expect that

- Participants will adopt measurably different problem-solving strategies.
- Participants will complete unfamiliar tasks faster.
- Participants will report lower cognitive load.
- Participants will express a preference for the task-centric interface.

7.3 Results

As mentioned at the start of this chapter, the primary objective of this study is to investigate whether the support provided by Workflows successfully supports an alternative problem-solving process to that currently used in feature-rich software, and to understand users' strategies for performing unfamiliar tasks in this paradigm. A secondary goal is to test if Workflows improves performance over existing practices.

We present our results in a different order to that described above. First, we report on the quantitative results of the study (task time and cognitive load). These results then serve as a backdrop for presenting our more holistic analysis of participants' behavior during the study, and the results of the semi-structured interviews.

7.3.1 Analysis Notes

For normally distributed data, we use t-tests to analyze significance and report means and standard deviations. Otherwise we use the non-parametric Wilcoxon Signed-Rank test and report median and interquartile range (IQR).

To check for bias introduced by the experimenter, two expert raters not associated with the research judged the completeness of participants' final images on an 8-point scale. Mann-Whitney U tests indicate that there was no significant differences in final image completeness between the two conditions for tasks completed within the time allotted (median rating 7.5 (IQR 1) for the experimental condition vs. 7.5 (IQR 1.6) for the control, $U=1127.5$, $Z=1.270$, $p=0.20$), or for tasks where the time limit was hit (median rating 4 (IQR 1.5) for the experimental condition versus 5 (IQR 2.5) for the control, $U=115$, $Z=-0.961$, $p=0.35$).

7.3.2 Task Times and Cognitive Load

In the first session, users completed 41 of 64 study tasks within the time allotted, with more tasks completed in the Workflows condition than in the control (23 vs. 18). In the second session, users completed 48 of 60 study tasks within the time allotted, with more tasks completed in the Workflows condition than in the control (26 vs. 22). Neither of these differences in number of tasks completed was found to be statistically significant. In our analysis of task times presented below, we include all tasks.

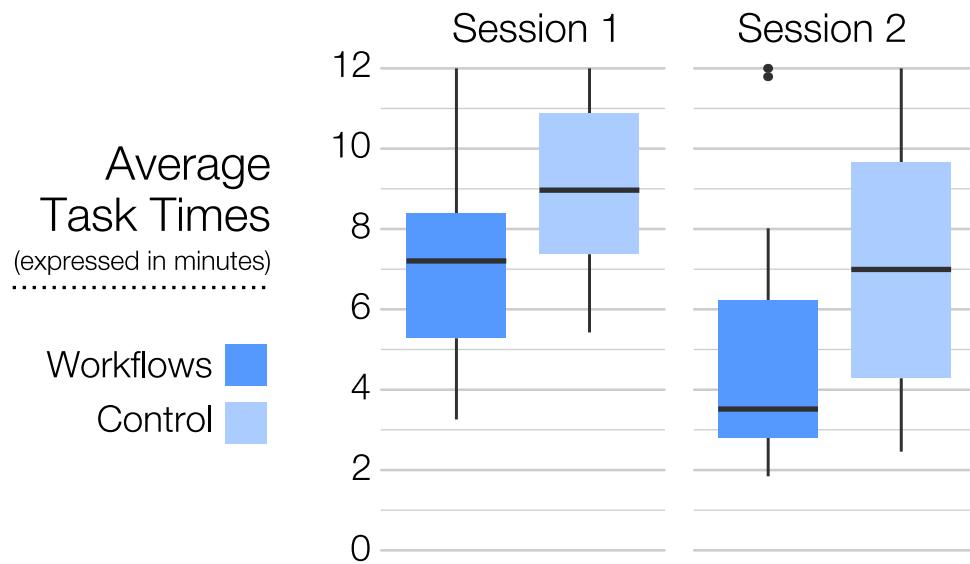


Figure 14. Average per-participant task times.

To compare task completion times, for each participant we averaged the time they spent on the two tasks for each condition / session (Figure 14). For both the first and second session, we observed significantly lower average task completion times in the experimental condition.

For Session 1, the median task time using Workflows was 432 seconds (IQR 187), versus 538 seconds (IQR 210) for the control condition. A Wilcoxon Signed-Rank test showed this difference to be significant ($W=11$, $Z=-2.783$, $p < 0.01$, $r=0.49$). This difference persisted in the second session, in which the median time for Workflows was 211 seconds (IQR 205) versus 420 seconds (IQR 322) for the control condition ($W=13$, $Z=-2.669$, $p < 0.01$, $r=0.49$).

Our findings for cognitive load mirrored the findings for task completion times. The results of the NASA-TLX, broken down by component subscales and session are shown in Figure 15.

For Session 1, a paired t-test found a significant difference in the average cognitive load for condition ($t(15)=-2.7462$, $p < 0.05$, Cohen's $d=0.69$), with the Workflows condition showing lower cognitive load than the control condition. This result was mirrored in the Session 2, ($t(14)=-2.1998$, $p < 0.05$, Cohen's $d=0.57$). In terms of the individual subscales, Workflows performed better than or was equivalent to the control condition in all cases.

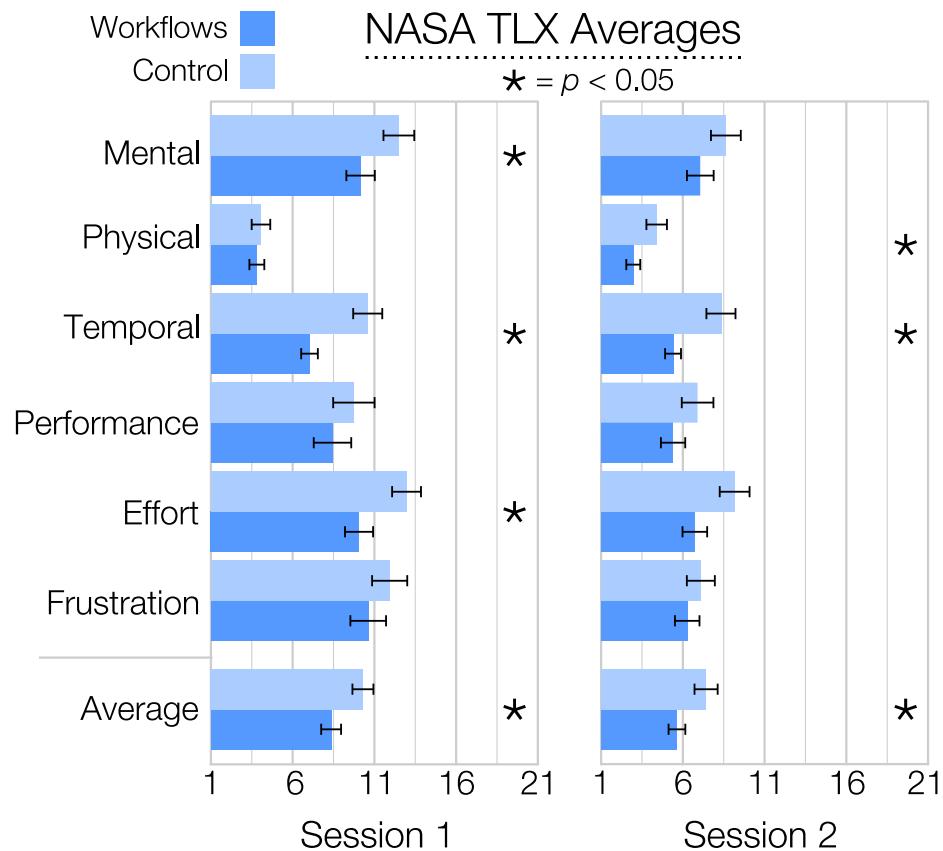


Figure 15. Average cognitive load for the six axes of the NASA TLX by condition and session. Error bars indicate standard error.

The results for task time and cognitive load indicate that use of a task-centric interface design can significantly improve performance and lower cognitive load as compared to current strategies for performing unfamiliar tasks.

Our qualitative analysis provides a more detailed account of where participants are saving time in Workflows, and the sources of difficulty and frustration in the control condition.

7.3.3 Problem-Solving Strategies

Our quantitative results indicated that participants completed tasks faster and exhibited lower cognitive load using Workflows as compared to the control condition. To gain insights into the problem-solving strategies employed by participants, we performed a qualitative analysis of the video recordings of the study sessions.

7.3.3.1 Method

We chose to analyze two of the study tasks, Selective Colorization and Reflection, in greater detail. These particular tasks were selected because they are the more complicated tasks in each of the two sets of tasks, and we reasoned that they would reveal more about participants' problem solving strategies.

The author of this thesis reviewed the video recordings of the study sessions, making notes and assigning qualitative codes to periods of time spent on various activities. Codes were developed following an open coding protocol derived from the methodology of grounded theory (Corbin and Strauss 2007). First, a small set of study videos was viewed to develop an initial set of codes. Next, the Selective Colorization task was coded, viewing the 31 study sessions for this task in a randomized order. Following this, the Text Reflection task was coded. For this task, the two sessions for each individual participant were viewed in order, one immediately following the other. However, the order of viewing individual participants was still randomized. The intent here was to reveal patterns between the first and second sessions of individual participants.

During coding, the initial coding scheme evolved in minor ways. When this happened, the codes for earlier videos were revised based on the updated scheme.

A summary of the resulting codes by study condition is shown in Table 7. Note that some of the codes were specific to one condition or the other (e.g. users in the Workflows condition sometimes performed searches that returned no results, but this was never the case when using web search in the control condition). In the sections that follow, we use the coding results to ground our discussion of observations from reviewing the video recordings.

Workflows	Control	Observation
41.2	--	Following instructions from a workflow.
17.2	27.3	Experimenting with tools or settings.
12.8	21.6	Looking for relevant functionality in menus, toolboxes, settings dialogs, etc.
8.5	21.2	Using the full interface to perform an action (excluding experimentation).
--	17.8	Viewing a page in the web browser (control condition only).
6.2	--	Keyword searches in the Workflows panel that returned no results.
4.9	5.9	Keyword searches in the Workflows panel / Web that return results.
3.9	1.3	Other (not covered by other categories, e.g. adjusting size of a window, switching between open images, reading text in the interface).
2.3	1.1	Interacting with the experimenter (e.g. experimenter encouraging participant to continue with task).
2.3	3.0	Error recovery using Undo (excluding experimentation).
1.7	--	Browsing workflows using the Recent Searches bar.
0.4	0.2	No obvious action being performed.
0.1	0.2	Using GIMP's in-application help system.

Table 7. Summary of qualitative codes for the two study conditions.
Numeric values indicate the percentage of time coded with each activity across all users.

7.3.3.2 High-Level Problem-Solving Strategies

Participants exhibited different overall strategies in the two conditions. In the Workflows condition, participants nearly universally started by using keyword search to try and find a relevant workflow. If they succeeded, they would then attempt to use the workflow to complete the task, making use of the actionable commands and taking guidance from the provided instructions. We term this the *guided-and-constrained* strategy—“guided” because participants used the workflow instructions as a framework for completing the tasks, and “constrained” because participants spent less time using the interface outside the Workflows panel.

In the control condition, participants would typically attempt to complete the task with minimal assistance from external help or documentation, even though a web browser with Google Search was open and available to them. Instead, participants would search through the interface trying to find relevant functionality, and would frequently experiment with tool and commands. We term this the *interface-exploration* strategy.

These strategies are reflected in the results of our qualitative coding. In the Workflows condition, participants spent the largest share of their time following instructions in the workflows panel (41% of time)⁹. If we also include time spent issuing keyword searches (both successful and not), and using the Recent Searches bar, participants spent 54% of time interacting with the system primarily through the Workflows panel. In contrast, the two most common activities in the control condition were experimenting with tools and settings (27%); and looking through menus, toolboxes, and dialogs (22%). This is consistent with the interface-exploration strategy discussed above.

In the Workflows condition, participants spent significantly less time looking through the interface for functionality (Wilcoxon Signed-Rank test, $W=70$, $Z=-3.029$, $p < 0.01$, $r=0.38$), and experimenting with tools and settings ($W=84.5$, $Z=-2.876$, $p < 0.01$, $r=0.37$). They also spent less time using the full interface of the application ($W=31$, $Z=-4.253$, $p < 0.01$, $r=0.54$).

These findings suggest that, by providing an explicit mechanism for locating relevant functionality and determining the procedures required to reach an intended goal, the task-centric interface provided by Workflows effectively supports an alternative problem-solving strategy to self-guided exploration of the application's interface.

Search Behavior and Use of Task-Centric Help

Combining time spent on search-related activities in each of the two conditions, we found that participants spent significantly more time on keyword search in the Workflows condition (including successful searches, failed searches, and use of the Recent Searches bar to navigate through past searches) ($W=336$, $Z=2.129$, $p < 0.05$, $r=0.27$). Again, this is consistent with participants following an interface-exploration strategy in the control condition, rather than choosing to use web search for help.

This difference in overall strategies between the two sessions was explicitly mentioned by several participants during the post-study interviews, as reflected by the following quotes:

When I was just using web search, I would really try to do it on my own first, I would try to actually figure it out before going to the web. Whereas with the workflow manager first thing I did was type

⁹ This includes time spent reading the workflow instructions and interacting with the system using the actionable buttons included in a workflow, but also includes some interactions outside of the Workflows panel, if the participant was following an instruction from the workflow (e.g. the instruction “Select the greyscale layer in the Layers dialog.” requires the user to click a layer in the indicated dialog). This was typically easy to determine because participants would sequentially follow workflow steps.

in what I wanted to do, and I didn't even pick a tool. So like, if I found the workflow, I could just pick all the tools from the workflow manager. [P11]

Like normally, [...] I would try to look at things, and then get a general sense of what things do, and then click on things and try it, and then if I didn't have success or I got confused I'd go to the web. But for this one [the Workflows panel], I'd just type it in there first. [P9]

In terms of the session where we were able to use broader tools, including the web, you know. This time I tried to do it a little bit more, without going to the web... [P10]

As the above quotes suggest, participants did still use the web in the control condition. In only about a third of cases did we observe participants not seeking help at all in the control condition (for the tasks that we qualitatively coded, 5/16 tasks included no use of the web for Session 1, and 5/15 for Session 2)¹⁰. However, whereas searching was the typical way that users would start a task in the Workflows condition, in the control condition searching typically occurred later. For the tasks in which participants used web search in the control condition, the median time before a first search was 28 seconds (IQR 173) in the first session, and 20 seconds (IQR 77) in the second session. In comparison, for the Workflows condition, the median time before the first search was 2 seconds (IQR 3) for the first session, and 2 seconds (IQR 1) for the second session. That participants started by searching in the Workflows condition is not surprising, because we asked them to try and use the Workflows panel to complete the tasks. However, this comparison shows that web search was *not* the initial strategy adopted by participants in the control condition.

Perhaps because participants typically searched after first engaging in some self-guided exploration in the control condition, we found that some users would use web search in support of the interface-exploration strategy. Resources found on the web appeared to serve as “hints” to move forward with interface-exploration, or to help locate a command the user had been searching for in the interface. Participants would notice references to commands in web content, and then jump back to the application and try them out. Overall, we got the sense that the participant was holding in their head a complex web of theories and ideas on how to proceed with the task, and constantly evaluating them for potential ways forward. When some new bit of information was found from the web, this would spur new theories and give them new directions to explore in the application.

In the control condition, participants did sometimes adopt a step-by-step *tutorial-following* strategy that was similar to how Workflows was used. In this strategy, the participant would locate a web-

¹⁰ Note that these were not the same five participants. Only one participant in this data set did not search in both sessions.

based resource that described how to perform the task, and deliberately follow its steps. Once a participant found a resource to follow, they would read the instructions, switch back to the interface and try and carry out some action, switch back to the web and read some more, etc. A notable difference between this approach and use of Workflows is that the process was often slowed by time spent switching back and forth between the web browser and interface. In particular, in cases where the instructions called for applying a dialog command, it was common for the user to switch back and forth between the web-page and the application several times in the process of opening the dialog, adjusting its settings, and finally confirming and applying the command.

In summary, the findings discussed in this section indicate that the Workflows panel effectively supports an alternative, guided-and-constrained problem-solving strategy for completing tasks in feature-rich software. In addition, participants' comments regarding their use of the web, the percentage of users that did not use the web in the control condition, and the observation that some participants use of the web sparingly to support their self-explorations of the interface, collectively suggest that participants view use of the web as a high-cost strategy.

Next, we discuss common sources of difficulty that we observed in the two conditions. This provides further insights into the relative benefits of the two problem-solving strategies we've discussed in this section.

7.3.3.3 Sources of Difficulty – Control Condition

In addition to exhibiting different problem-solving strategies in the two conditions, we observed that participants faced different common sources of difficulty in the two conditions, and that these could be traced back to the problem-solving strategies identified above. We first discuss common difficulties we observed in the control condition, followed by the Workflows condition.

Three common sources of difficulty we observed in the control condition were cognitive overload, red-herring commands, and setting state problems. We discuss each in turn.

Cognitive Overload – Participants following an interface-exploration strategy would appear to rapidly switch between trying to find relevant functionality; experimenting with functionality that they found to learn how commands and tools function; and devising, evaluating, and reevaluating strategies for progressing toward the task goal. It's difficult to imagine how juggling all of these concerns *wouldn't* impose a high amount of cognitive load on the user. This is consistent with the finding of higher self-reported cognitive load discussed in the previous section. As an anecdotal

example of the stress that this strategy put participants under, after about seven minutes of working on a task in the control condition, one participant laughed and said to herself “This is the biggest nightmare ever.” [P15]

Red-Herring Commands – Self-guided explorations of the interface would sometimes lead participants to find commands that *appeared* to be related to the current task, but were not quite right. For example, a common solution to the Reflection task is to start by making a copy of the text layer, and then flip it vertically. The vertical flip operation is best accomplished with either the *Flip* tool, or the *Layer>Transform>Flip Vertically* command. However, many participants’ explorations led them to instead find the *Image>Transform>Flip Vertically* command. This command flips the entire image, including all layers, and was a source of much frustration for participants in this task, who desired to flip only the currently selected layer.

We term these *red-herring commands*. To new users they appear to be related to a task, but they are actually irrelevant and a source of frustration. The interfaces of feature-rich software contain a huge number of commands, and many of them share close conceptual similarities with one another, so it isn’t surprising that novice users will use commands that they assume to be related to the current task, but are not quite right. Red-herring commands can be considered to be an instance of the concept of *false affordances* identified by Novick et al. (Novick, Andrade, and Bean 2009).

Setting State Problems – Finally, after experimenting with tools, participants would seldom reset their settings back to the application defaults. This would leave the system in a non-typical state, and caused frustration when the side effects of previous experimentation caused unexpected results later on. While this problem was not unique to the control condition, participants engaged in significantly more experimentation with tools and settings in the control condition.

The sources of difficulty discussed above all logically follow from use of the interface-exploration strategy.

7.3.3.4 Sources of Difficulty – Workflows Condition

The common difficulties that we observed in the Workflows condition were related to following along with workflow instructions. Specifically, some participants experienced difficulties with finding a workflow using keyword search, errors due to skimming text instructions, and understanding the dynamics of how to use tools or commands in a workflow.

Formulating Search Queries – In the Workflows condition, a number of participants had difficulty formulating keyword searches to find a workflow for their task. Because the pool of workflows available in the study was small, this often resulted in searches that would return no results, which was a source of frustration for participants. For example, during the Reflect task, one participant became frustrated and said:

Well there's obviously some super easy tool that just lets you do it, instead of some retarded [sic] way that I'm doing it, but I don't know what else to type in to the [Workflows search box]. Like I don't know what to call it, like, there's no words coming, it doesn't help you that way. It's like "fade", "blend", I don't know what to call it, to search. [P6]

Conversely, in the control condition we observed participants making extensive use of Google's query completion features; the participant would type in search terms and carefully consider the list of suggestions that was displayed, and then either select a suggested query, or refine their query based on terminology from the list of suggestions.

These observations are consistent with our findings from the think-aloud study of AdaptableGIMP reported in Chapter 5, and suggest that more can be done to assist users with declaring their intended goal. More specifically, participants' use of Google's query completion feature in the control condition suggests that providing a mechanism for helping the user to learn relevant terminology would be beneficial.

Skipping Text Instructions – For the Selective Colorization task, a common source of difficulty in the Workflows condition came from skipping over text instructions, such as “switch to the greyscale image”. In extreme cases, participants treated the workflow like it was a macro, where they could simply click through each embedded command to complete the task. One comment in the interview sums up this problem:

My first instinct is to go through really quick, you know what I mean, and I'm not necessarily reading all the text. So I click on the tool and I just think “I don't have to read what's in between” and I click the next tool. And that's me just being kind of lazy, or whatever, but that was kind of the way that I wanted to do it. But some of those tools, to make them work properly, or fully understand what I had to do, I had to read some of the text a little bit more, and that wasn't necessarily how I was going, just right out of the gate. [P10]

We suspect that this is the result of the higher visual weight and interactivity of the actionable commands, which leads users to focus on them more than the included text-based instructions. As well, in some workflows the text instructions could be safely ignored, and the correct result could be

yielded by using each of the commands in the workflow in some obvious way, while in other tasks, such as the Selective Colorization task, not following text instructions would cause serious problems.

In Chapter 8 we discuss the implications of this finding in greater detail.

Understanding the Dynamics of Tools and Steps – Finally, we observed that participants sometimes had difficulty with understanding the dynamics of how to use a tool included in a workflow. We found this was especially problematic for steps that involved use of direct manipulation tools. For example, the final steps of the workflow for the Reflect task involved using the *Eraser* tool to fade the reflected text. A number of participants reached this step, and then appeared to reject this approach and started searching for another way to fade the text. In other cases, on this step the participant used one stroke of the Eraser tool to fade the text slightly, and then started making settings adjustments to the tool (sometimes leading to setting state problems, as described in the previous section). The most straightforward approach is to make several strokes on top of one another with a soft brush. Adjusting settings, as participants often did, added extra time and effort.

This observation suggests that our text-only instructions may be too limited to fully communicate the dynamics of how to perform certain steps.

Each of the difficulties that we have discussed in this section suggests areas where the design of Workflows could be improved. In the next chapter, we revisit these difficulties and provide specific ideas for how they could be addressed in an improved design.

7.3.3.5 Guided-and-Constrained vs. Interface-Exploration

In summary, the results of our qualitative coding and observations show that Workflows successfully supports a “guided-and-constrained” problem-solving strategy in which the participant focuses on using the functionality and guidance provided by the task-centric interface to attempt to reach the task goal. We observed that this strategy helped participants to avoid common sources of difficulty associated with the common interface-exploration strategy used in feature-centric interfaces, including cognitive overload, red-herring commands, and setting-state problems. Use of Workflows was associated with a different set of common difficulties, including difficulty with determining terminology for search queries, overlooking instructions in a workflow, and understanding the dynamics of how to perform particular steps documented in a workflow.

7.3.4 Learning in Session 1

Up to this point, our analysis has looked at common patterns across both study sessions. In this section, we look at what our study revealed about what participants learned and retained between the first and second session.

In terms of overall task times, in both the Workflows and control conditions, task times were significantly faster in the second session. For Workflows, the median time for Session 1 was 376 seconds vs. 242 seconds for Session 2 (Wilcoxon Signed-rank test, $W=334$, $Z=4.026$, $p < 0.01$, $r = 0.51$). For the control condition, the median time for Session 1 was 564 seconds vs. 419 for Session 2 ($W=224$, $Z=3.165$, $p < 0.01$, $r = 0.4$). These results suggest that some sort of learning is occurring in both conditions.

We found evidence to suggest that in both conditions participants retained knowledge of relevant functionality in the second session. For the tasks we qualitatively coded, participants spent significantly less time in session 2 looking through the interface for functionality (Wilcoxon Signed-rank test, $W=289.5$, $Z=2.896$, $p < .01$, $r=0.37$) and experimenting with tools and settings ($W=271$, $Z=2.920$, $p < .01$, $r=0.38$).

However, as mentioned in our discussion of problem-solving strategies, the number of participants that searched in the second session was the same as in the first session. This would suggest that participants often do not fully recall the procedures for tasks from the first session.

Among participants who used search in both sessions (14 in the Workflows condition, 7 in the control), significantly less time was spent on search in the second session ($W=159$, $Z=2.016$, $p < .05$, $r = 0.31$). This suggests that participants employed more refined search strategies in the second session. We found additional evidence for this in the post-study interviews, where five of the 15 participants mentioned that remembering terminology from the first session helped them to return to resources in the second session. In the following quotes, participants describe how they used this strategy in the Workflows condition:

...you kind of learn how to use the workflows, I guess, you know what I mean. Like, in my experience I remembered what to search in the workflow, so that was the learning aspect of the program. [P10]

I knew, oh last time I used “greyscale”, and that worked, so I can just put “greyscale” and then boom. And I knew that worked. [P4]

I think, for me what made the difference, is I knew... having used it in the previous session I had a better idea of what words to use and what to look for, I guess, the terminology. [P5]

We also found evidence for this strategy for participants using the web in the control condition, as suggested by the following quotes:

I use the Internet to do everything in my life that way. So it's almost like, you think, "Oh I don't really need to remember it because it's on the Internet." [P10]

I think from the first session, since I knew what I was looking for, like the terms. Then my Google search seemed a lot more narrow. Like, initially, I was looking for, um, I remember looking for general things like 'how to mirror text' or I didn't know what to call how to correct a horizon. [P1]

This suggests that, in addition to learning relevant functionality, participants also learn how to return to task-centric help resources. We term this *keyword learning*.

Though we observed this phenomenon in both conditions, it is particularly significant for task-centric interface designs, where the dominant problem-solving strategy is based around search. In contrast, participants in the control condition seemed hesitant to go to the web for help. For example, P10 in the Reflect task started session 2 by saying "oh" quietly to himself as if he was trying to remember something. As he did so, twice he moved his mouse pointer to the icon for the open web-browser in the task bar, as if he was going to click, but then returned the mouse to the interface instead. This internal debate lasted for 10 seconds, after which he adopted an interface-exploration strategy, searching for commands. Only after a minute more of exploring did he switch to the web browser window and re-find a page from the first session. While this is only one specific example, the lower number of participants who used search in both sessions in the control condition, and the longer time before using search (when it was used) in the control condition, collectively suggest a hesitance to go to the web for help, and a corresponding overestimation by individuals of how much progress they can make unaided.

7.3.5 Participant Reactions

A final goal of this study was to elicit participants' subjective reactions to Workflows, and the concept of a task-centric interface. This was achieved using a post-study questionnaire, and questions asked in the semi-structured interview.

7.3.5.1 System Preference

We start by looking at system preference. In the questionnaire administered at the end of each session, we asked participants to indicate their preference for one of the two systems (5-point scale, 1=GIMP+Web, 5=GIMP+Workflows). As can be seen in Figure 16, the majority of participants expressed a preference for Workflows in both sessions.

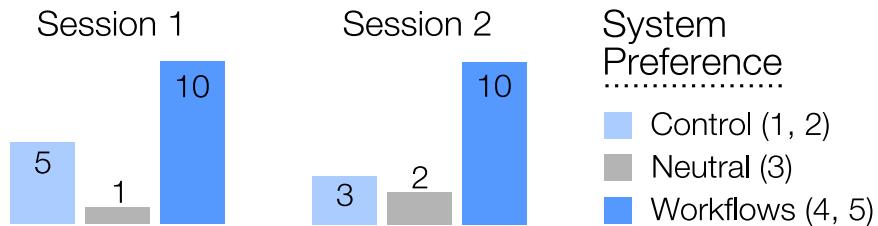


Figure 16. Results of 5-point system preference ratings, combining slight and strong preferences for each system.

7.3.5.2 Preferred Features

To elicit reactions to the features of Workflows, we asked participants to contrast the positive and negative features of the two conditions.

The most appreciated feature of the Workflows system was the actionable buttons (mentioned by 11 of 15 participants), as the following quotes demonstrate:

Sometimes when I was working on my own, I had to scroll over everything to find exactly what they meant. Whereas this says, okay 'Crop' and then it gives you the actual tool, so you're not having to look. [P7]

It saves you having to look for [the commands] after you have the instructions, you know what I mean. [P10]

(...) it gave step-by-step instructions on how to complete a task, as well as even providing the buttons right there, if I couldn't find where the buttons actually were in the program. Compared to the web, I mean you type in something and it doesn't tell you exactly where the tool is, sometimes it just explains what the process is without actually, I guess, guiding you through it. [P15]

A key reason for appreciating the actionable buttons was that they saved the effort of locating commands in the interface (mentioned by 8 of those 11 participants).

Participants also appreciated the succinct presentation of instructions in the Workflows system, and often contrasted it to the more verbose presentation of resources on the web.

I will say though that the steps in the workflow manager are way easier to follow, than having to interpret search results. [P11]

[The workflows panel] was pretty verbose, but not to the point where say, it was one of these [referring to a web tutorial on screen] where all this could be easily said within like two instructions within the workflows panel. This is what I'm looking for, but there's a whole bunch of [extra] stuff on the webpage. [P14]

Conversely, participants cited the more detailed presentation of information on the web as a source of difficulty, as in the following quotes:

There's a lot of information. There's steps, but it's not parsed like this, where it's specific [pointing to workflows panel on screen]. There's in between, like, people's dialogs, and stuff, so you have to like, there's chunks and I was like, reading it fast like 'OK what's relevant?' [P9]

There's just so much, that, it's sometimes hard to know if what you're looking at is going to be helpful or not, at the beginning. So sometimes you have to get into it a bit, before you... [...] It's not always clear at the beginning how applicable it is to what you're doing. Or how complicated it's going to be. [P16]

Participants also appreciated that workflows were presented in-application, so they did not have to switch between different windows to complete the task.

It's more convenient. It's all in the same pane rather than switching back and forth windows. [P5]

One [advantage] is that it's all in the same environment, so you don't have to open some other program to access this workflow thing. Like, you don't have to alt-tab. Like if you only have one monitor, then you're constantly switching back and forth, it's a hassle. [P11]

Collectively, the above feedback provides strong support for the task adaptation and task knowledge mechanisms provided in Workflows, as well as the overall idea of a task-centric interface providing in-application support for performing unfamiliar tasks. That the actionable buttons were mentioned by such a large percentage of participants could also be further evidence of the frustration experienced by users while searching through the interface looking for functionality in the control condition.

Greater Confidence in Instructions

An unexpected result was that participants expressed a greater confidence that the instructions presented in Workflows were correct and would work in the application, as in the following quotes:

That's like, one of the nice advantages of the workflows, because it's embedded in the program, it's created for the program, so you have confidence like, this is the right way to do it. [P14]

The first time, I think, I used one [web page], and it said something about like, the version, so like I at first didn't know if it was going to work out, because it said something about different gimp versions. And I don't even know if it was for GIMP software, for one that I used, I can't remember. [P9]

These comments suggest that participants spent less of their cognitive resources on evaluating the quality of resources in the Workflows condition, because they feel it was safe to assume that they would work. On one hand, this finding is encouraging because a greater confidence may make users more willing to utilize a task-centric interface for help. On the other, it may also suggest that users may have a higher expectation of quality for task-centric help provided in the application, which would need to be taken into account when designing the workflows to include in a system.

7.3.5.3 Areas for Improvement

Participants also identified features of the Workflows system that they felt could be improved. The most common of these concerned problems with finding a workflow that was suitable for their task.

I think I mentioned it before, but sometimes no results would pop up at all. Like when I was working on the cat one, I thought a very basic way of finding it would be writing "one color", for example, and nothing showed up! [P15]

I struggled trying to complete this task the first time around mostly because I didn't know which word to put into workflows, that would give me a workflow to get me to [the result]. [P14]

This is consistent with our observation that participants experienced greater difficulty with coming up with terminology in the Workflows condition, and that participants benefitted from Google's query completion features in the control condition.

7.3.5.4 Discoverability, Mastery, and Freedom

To gain further insights into participants' perceptions of task-centric interfaces, and the particular problem-solving strategy supported by Workflows, we asked participants what they thought of the extreme idea of replacing all menus and toolboxes with a purely task-centric interface based on keyword search.

Some participants enthusiastically embraced the notion of a purely task-centric user interface, as this quote illustrates:

Oh yeah. Yeah. I would prefer it, because then I don't have to go through, I mean there are what, four or five [counting menus in GIMP], ten, eleven, including the help box, different tabs along the top here, whereas I could just use the panel and it's all right there. [P8]

However, participants also raised questions about serendipitous discovery of commands, the potential to “master” the software, effects on performance, and the impact a purely task-centric interface could have on free-form use of the software.

In the following comment, a participant describes the perceived impact a search-only system could have on discovery of functionality:

Like for this one, I know last time I accidentally discovered this, Fuzzy Select tool. Well this time, I remember doing that, so I can go right to there immediately. I can use it without having to look it up. [P4]

The inability to “master” the software was also a concern for some participants:

I guess, like, if I had a whole batch of photos I would want to be able to master the program. And to know that I can edit whatever I need to edit immediately, without having to constantly refer to the manual. [P15]

As was the potential impact of a purely task-centric interface on advanced users:

I wouldn't get rid of the menus and the toolbox, simply for the advanced users. It would drive them nuts if they had to search for everything. If they know exactly what they're doing, they could just click on the tool. [P2]

Finally, some participants felt that they would lose their freedom, or the ability to change course in a purely task-centric interface. For example:

You would lose, for lack of a better word, your freedom to just kind of change the plan. [P11]

I think it could [replace menus and toolboxes], but I feel a lot of users of editing programs would not be happy about it. Because then you're restricted only to what the workflow allows you to do. [P15]

A commonality to the concerns cited above is that they portray the task-centric interface as a barrier to efficiently performing an action in the interface; the task-centric interface requires the user to “look [a command] up” before using it, “refer to the manual” before editing, and “search for everything” rather than “just click[ing] on the tool”. This raises the question of what exactly is creating this perception in participants. It may be that this perception is related to performance, and that participants are comparing the time it would take to invoke a command using a menu or keyboard shortcut, to the time to issue a keyword search to bring up a customized interface. It may also be related to cognitive load, and participants are comparing the cognitive cost of the largely unconscious action of issuing a familiar command, to that of formulating a keyword search query (which a number of participants experienced difficulty with). Finally, it may be that participants perceive the task-

centric interface as a help system (as suggested by the one participant referring to it as “the manual”), and thus associate it with novice and inefficient use of a system. Investigating the factors contributing to this perception is an interesting area for future work.

7.4 Summary

The results of our study show that the task-centric interface design provided by Workflows effectively supports an alternative strategy for performing unfamiliar tasks in feature-rich software. In particular, by indicating relevant functionality and required procedures to perform unfamiliar tasks, the task-centric interface enables the user to adopt a guided-and-constrained problem-solving strategy, instead of the interface-exploration strategy typically used in feature-centric interfaces.

Our study also delineated a number of advantages and disadvantages of this approach, which we summarize below.

7.4.1 Advantages

The main advantage of the guided-and-constrained problem-solving strategy is that it enables the user to focus on learning the commands and procedures for the task at hand, without having to spend time and energy on determining which commands or procedures are appropriate for the task. This is an advantage because it helps the user to avoid three common sources of difficulty we observed in the control condition: cognitive overload from searching through and evaluating the huge number of potential commands and options that may be related to the desired task; encountering red-herring commands, which appear to be related to the task, but are not; and trial-and-error experimentation with tools and settings, which can put the application into non-typical state. The end result is that the task-centric interface enables users to complete unfamiliar tasks faster, and with reduced cognitive load.

The use of keyword search as a task-declaration mechanism in Workflows also enables the user to learn and use the application at the higher conceptual level of tasks, rather than lower-level commands. For new tasks, keyword search enables the user to cross the gulf-of-execution without turning to external resources (e.g. on the web), which is significant because the web is viewed as a high-cost strategy. For tasks that have been previously performed, keyword search enables the user to engage in *keyword learning*, and learn how to return to a task-centric help resource, instead of the fine details of how to perform the task. This is significant because users have trouble remembering procedures for occasional tasks.

7.4.2 Disadvantages

We also identified a number of areas where the alternative problem-solving approach supported by Workflows' task-centric interface has potential disadvantages over that supported by feature-centric interfaces.

First, declaring an intended task using keyword search was a source of frustration for some participants, who had difficulty with determining the right terminology to use in a query.

Second, though Workflows enabled users to complete unfamiliar tasks faster than the control condition, some participants perceive the task-centric interface as a barrier to efficiently performing actions in the interface. Since our main goal is to support users in performing unfamiliar tasks, addressing this perception is somewhat outside the scope of the current work, but it could act as a disincentive for users using a task-centric interface to begin with.

Finally, use of a task-centric interface presents fewer opportunities for serendipitous discoverability of commands, as compared to exploring a feature-centric interface.

None of these disadvantages is inextricably linked to the idea of a task-centric interface, and in the next chapter we discuss how improved task-centric interface designs could address some of these difficulties experienced by participants.

7.4.3 Research Questions Revisited

Revisiting the table of research questions presented earlier in this chapter, the results of our study are shown in Table 8.

Research Question	Study Results
How does Workflows affect the problem-solving strategies used to complete unfamiliar tasks in feature-rich software?	<ul style="list-style-type: none">• Significant differences in how time was spent in the two conditions.• Observation that common sources of difficulty were different in the two conditions.
Does Workflows yield faster times to complete tasks?	Confirmed <ul style="list-style-type: none">• Significantly faster average task completion times in the Workflows condition.

Does Workflows allow users to complete tasks with lower cognitive load?	Confirmed <ul style="list-style-type: none"> Significantly lower average NASA TLX scores in the Workflows condition.
What difficulties do participants experience in the two conditions?	Control <ul style="list-style-type: none"> Cognitive overload Red-herring commands Setting-state problems Workflows <ul style="list-style-type: none"> Vocabulary for search queries Skimming instructions Interpreting dynamics of how to perform steps
What is learned in the first session in the two conditions?	Across both conditions: <ul style="list-style-type: none"> The commands required for the task, and how to use them (though often incomplete) Task procedure (though often incomplete) How to return to helpful resources (<i>keyword learning</i>)
What are participants' subjective impressions of Workflows?	Positive <ul style="list-style-type: none"> Majority preferred Workflows to control Actionable buttons Succinct task instructions In-application presentation Negative <ul style="list-style-type: none"> Doesn't help with search vocabulary Discoverability, performance, and mastery as potential issues

Table 8. Research questions and the answers we found for them in our study.

In this chapter we've focused on answering our main research questions on the impact of task-centric interfaces. In the next chapter, we revisit many of the findings presented in this chapter to discuss the design implications for task-centric interfaces.

Chapter 8

Discussion

In this chapter, we present a more detailed discussion and interpretation of the main findings reported in this dissertation. We also discuss implications of those findings for the design of task-centric interfaces, and the interfaces to feature-rich software more generally.

8.1 Implications for Task-Centric Interfaces

Our study found that use of Workflows enabled users to complete unfamiliar tasks significantly faster and with reduced cognitive load as compared to using the unmodified GIMP application and web search, and suggested a number of sources for these gains.

First, the main source of benefit appears to come from Workflows supporting a guided-and-constrained problem-solving strategy, in which users offload the work of determining relevant commands and necessary procedures to the application. This enables the user to focus on enacting instructions and learning the dynamics of a smaller set of commands, instead of spending time on self-guided exploration and experimentation—activities we observed to be time consuming and a source of frustration.

Second, the actionable buttons provided in Workflows were mentioned as an advantage of the system by a majority of participants, because they save them from having to find commands in the application’s interface. This also enables the user to focus on performing the task using the customized interface, in this case by removing the time and effort required to find commands in the toolbox or menu hierarchy.

Third, participants expressed appreciation for the succinct instructions in Workflows, and contrasted them with the more verbose presentation of content on the web. This suggests that additional exposition provided in resources on the web (e.g. providing more general instruction on how to use the application, or the theory of how to perform a task) may have made it more difficult for participants to find the procedural information they needed in the control condition.

Finally, participants also expressed appreciation that Workflows provided support in the application itself, as opposed to in a separate window. As well, we observed participants frequently switching between the application and browser when following web-based instructions in the control condition. This suggests a benefit to performance and cognitive load from reducing context switching.

These sources of gains for Workflows can all be traced back to the key components of our task-centric interface design outlined in Chapter 4. In particular, the task adaptation and task knowledge components together provide support for the guided-and-constrained problem solving strategy that is a main source of benefit; task adaptation removes the need for users to spend time finding commands in the application’s interface; and providing task knowledge on-demand allows the system to provide succinct guidance for the user’s current goal, rather than attempting to teach use of the application more generally. Thus, our study findings provide validation for the overall idea of task-centric interfaces more generally than our specific instantiation in Workflows.

A common theme in most of the sources of gains we’ve identified above is that they come from a tradeoff between some type of learning with performance or cognitive load. In the next section, we discuss the impact of our design on learning in greater detail.

8.2 Learning in Task-Centric Interfaces

One lens for interpreting the impact of our task-centric interface design is to consider how it supports different types of learning. There are many different types of learning that can occur in feature-rich software, including:

- Learning the commands that are available in the application
- Learning how individual commands work, and how they are used
- Learning the effect of the various settings for a command
- Learning the locations of commands, settings, and dialogs in the application’s interface
- Learning the conceptual relationships between commands
- Learning how commands are used in the context of particular tasks
- Learning which tasks can be performed in the application
- Learning which commands are needed for individual tasks
- Learning fundamental concepts reused throughout the application

Task-centric interfaces encourage some of these types of learning and prevent others. For example, Workflows explicitly supports learning of which tasks can be performed in the application, the commands that are needed for individual tasks, and the conceptual relationships between commands

for tasks. However, it does not help users to learn the locations of commands in the interface full interface, or how individual commands operate outside of the context of particular tasks.

Viewed through this lens, one way to interpret our study findings is that Workflows successfully supports an alternative compromise in how different types of learning are supported than that which is usually seen in feature-rich software. Moreover, our results show that adjusting these tradeoffs can make it easier for users to perform unfamiliar tasks. This framing opens up some interesting areas for future work, which we discuss next.

8.2.1 Design Implication – Exploring the Space of Learning Tradeoffs

One area for future work would be to investigate alternative task-centric interface designs that support different balances between the various types of learning that can occur in feature-rich software. For example, we could design a task-centric interface that maintains the gains provided by explicitly guiding users to the commands and procedures that are needed for a task, but additionally supports the user in learning the locations of commands in the full interface of the application (e.g. by highlighting commands in-place in the toolbox and menus, instead of listing them in step by step instructions as in Workflows).

As well as investigating individual designs based on their support for various types of learning, it would be interesting to look at how to support transitions between approaches for completing tasks. For example, we could investigate techniques that enable users to take advantage of the performance gains provided by a task-centric interface for unfamiliar tasks, but also provide a smooth transition to more efficient techniques as the user becomes more familiar with the task and performs it more frequently. As an example, in the Workflows design this could be achieved by fading away or hiding the step-by-step instructions over time, so a workflow for a commonly used task becomes a succinct toolbar of commands. As a starting point, the problem of supporting a transition to more efficient interaction techniques has previously been explored for command invocation techniques (Scarr et al. 2011; Kurtenbach, Moran, and Buxton 1994), and some of the approaches investigated in this area may be applicable to tasks as well.

8.3 Keyword Learning

Extending our discussion of different types of learning, a second major finding from our study of Workflows is that it supports *keyword learning*—participants learn relevant keywords to return to task-centric help resources using keyword search, in some cases as a substitute for learning the lower-

level commands and procedures for completing a task. This phenomenon suggests that using keyword search as a task-declaration mechanism enables the user to interact with the application at a higher conceptual level of tasks, rather than the level of individual commands.

The observation that users re-find information has previously been identified in the domain of general web search (e.g. (Aula, Jhaveri, and Käki 2005; Obendorf et al. 2007; Teevan and Adar 2007)), but it is particularly significant in the context of task-centric interfaces. Once learned, a small set of keywords could allow the user to return to an arbitrarily large set and sequence of commands to help them complete a task. Stated another way, keyword learning of tasks has the potential to *scale* more gracefully than learning the precise details of how to perform each task. For infrequently performed tasks in particular, it seems reasonable to expect that this strategy will have clear benefits over learning constituent commands and procedures.

Keyword learning also has the advantage that the keywords themselves act as a kind of description of the task to be performed. In contrast, the commands necessary to complete a task may have generic names, or may use domain-specific terminology that is foreign to the user, making them more difficult to remember and recall. Because keyword search queries are intrinsically descriptive of the task to be performed, they are likely to be more memorable than the names of commands, or the details of procedures.

Further work is needed to fully understand how keyword learning compares to learning of functionality and procedures. In particular, it would be useful to model the tradeoffs between these two approaches, to see if anything is lost by learning keywords instead of functionality and procedures. It would also be interesting to test whether keyword learning is able to enable users to use more sophisticated techniques, which would be difficult to learn at the level of functionality and procedures. The answers to these questions could further shape the design of task-centric interfaces, as well as how keyword search mechanisms are used in feature-rich software more broadly.

While the keyword search task-declaration mechanism provided in Workflows was effective in enabling participants to complete the study tasks, and has benefits in terms of learnability, it was also a source of frustration for participants. In the Workflows condition, a number of participants experienced difficulty with finding the right terminology to use in keyword searches, and cited this as a source of frustration. Conversely, in the control condition we observed that participants often used Google's autosuggest feature to help them determine relevant terminology and build search queries.

In some sense, it shouldn't be surprising that formulating queries is a source of difficulty for users, since they may not know the specific terminology surrounding an intended task, and may also be unfamiliar with the terminology surrounding the application or application domain. Next, we discuss some ideas on how this could be addressed in improved designs.

8.3.1 Design Implication – Proactive Assistance with Terminology

To improve keyword search as a task-declaration mechanism, we suggest a model in which the system provides proactive assistance with performing keyword searches, including assisting with query formulation and determining relevant terminology.

One example for how this could work is provided by Google's autosuggest feature, which suggests completions to keyword search queries as they are typed. One design possibility for task-centric interfaces is to adopt such an autosuggest mechanism, and additionally tailor it to the particular domain of the application. For instance, in the visual domain of image editing, autosuggest could display example images alongside terminology or suggested queries. This could help users to recognize relevant queries, and also help them build an understanding of domain terminology as they use the system.

A first step toward investigating designs for assisting users with task declaration would be to establish a baseline cognitive cost (e.g. the cost of issuing searches with no proactive assistance), that proposed techniques could be compared against.

8.4 Guiding Users through the Space of Solutions

Another way to frame the gains provided by Workflows is that the system is directly supporting the user in exploring the space of potential solutions for performing an unfamiliar task. Current feature-centric organizations provide minimal help with narrowing this space, because they provide help on the granularity of individual commands (e.g. through their names, icons, tooltips, and how they are organized into menus). It's up to the user to determine the proper commands to use, the proper order to use them in, and the settings to use for each. In contrast, task-centric interfaces are able to reduce the search space to a small subset of commands, and provide explicit guidance on the order to use them in, and how to use them. This framing leads to a number of design implications.

8.4.1 Design Implication – Supporting Command Settings

One area for future work would be to consider how additional dimensions of the solution space for performing an unfamiliar task could be supported in a task-centric interface design. For example, relevant setting and parameter values for commands could be explicitly represented in customizations, either by associating parameters with actionable references to tools and commands, or through separate actionable buttons to synchronize settings to a particular state.

A potential challenge raised by representing settings in a task-centric interface is that they may be more dependent on the specific situation in which a task is being performed than commands and procedures. As a result, it might be worth exploring how ranges of individual setting parameters or alternative parameters for various common situations could be represented and indicated to the user.

8.4.2 Design Implication – Guidelines for Weaving Task Knowledge into Interfaces

The inclusion of task knowledge in the interface is the mechanism by which task-centric interfaces guide the user through the solution space. However, the results of our studies have shown that providing task-centric knowledge within an interface design can be complicated, and suggest areas for improvement.

The results of our think-aloud study of AdaptableGIMP showed that an ordered list of commands was itself insufficient to indicate how to perform tasks, and that users are hesitant to access task knowledge presented external to the main interface. These observations prompted us to include instructions in-line with actionable references to functionality in our Workflows design.

While this design was successful, in our study of Workflows we observed a stubborn tendency of participants to skim documentation, attempting to pick out the minimum amount of information needed to proceed with their tasks. Moreover, the actionable commands included in workflows appear to exacerbate this by becoming a focus for participants, causing them to overlook important text instructions.

In addition to the findings discussed above, our study of Workflows found that short, text instructions were not always sufficient to communicate the dynamics of certain tools and operations, in particular those requiring direct manipulation. This is consistent with past work that has shown the value of providing checkpoint images in tutorials (Grabler et al. 2009b), and using short videos to demonstrate the dynamics of direct manipulation tools (Chi et al. 2012).

With all of these findings in mind, we have developed a short set of guidelines for including task knowledge in applications:

- Provide instructions in-line with relevant interface elements
- Keep instructions succinct and minimal
- Be aware that users will skim and focus on interactive elements
- Use rich mediums as necessary to communicate the dynamics of individual steps

Finding novel ways to best address these guidelines is an interesting area for future work. One design idea for addressing the tendency of users to skip over instructions and focus on interactive elements is to include *human action buttons* that, when clicked, would pop up text or animated instructions for the user to follow. This would provide a way to ensure that important actions can be given the same attentional weight as the interactive elements in a workflow.

We also note that a tension exists between the guideline to provide succinct instructions, and the guideline to use rich mediums to communicate the dynamics of individual steps. One idea for addressing this tension is to adopt a hierarchical format for presenting task knowledge, where initially succinct instructions are provided, but more detail on individual steps can be accessed as needed.

8.5 Use of Feature-Rich Software for Short, Targeted Tasks

Our large-scale study of ingimp (Chapter 3) found that the typical user performs short, targeted tasks, with only a small percentage of the available functionality. One way to think of this is as “*À La Carte*” usage, in which the user picks out a small subset of the full functionality provided in a feature-rich application.

This finding is important because it establishes that à la carte usage of feature-rich software is a valid use case. Moreover, as we’ve argued in Chapter 4, this use case is poorly supported by the feature-centric interfaces typically used in these applications, which focus on presenting one organization of all of the available functionality, even though the majority of it will not be required for any given task.

With task-centric interfaces, we have proposed a way that interfaces to feature-rich software could explicitly support this use case, by providing a quick way to trim down the interface for a specific intended task. However, so far we have considered this approach as applied within individual

applications, and we have not addressed considerations such as the time to install or open the application itself.

If we take a broader perspective, supporting à la carte usage could mean re-imagining how individuals use software at a fundamental level. For example, consider the scenario in which a user wishes to perform an image editing operation, but does not currently have an image editing application installed on their computer. In order to complete the task, the user must first learn that GIMP (or another appropriate application) exists and is able to support their intended task, then download and install the application, and finally engage in learning and using the application to complete the task. Considering our finding that the average active usage time in an ingimp session was only ~6 minutes, this process represents a potentially huge initial cost on the user to perform an unfamiliar task.

To support this scenario, we could imagine a task-centric interface that takes the form of a web-based search engine, into which the user types an intended task, and gets a customized interface for performing that task, with no installation required. The “application” becomes a portal to a collection of task-specific customized interfaces, rather than a monolithic application that must be installed on the user’s computer before it can be used.

Taking this vision even further, this model calls into question the distinction between separate applications. Instead of separate applications, multiple developers could contribute functionality to a library that can be drawn upon when creating task-specific interfaces.

This vision may seem like an extreme departure from the current model of computing, but in some ways this is already happening. Tasks that were once the sole purview of feature-rich applications can now be performed using simple web-based applications, or apps installed on mobile devices or tablets. For example, web-based applications such as Pixlr Express¹¹ and Photoshop Express Editor¹² offer simple sets of common photo manipulation operations, and apps such as WavePad and Android Audio Editor enable users to do basic audio editing on mobile devices or tablets.

In the next chapter, we include a more detailed discussion of some of the possibilities for future work that could come from this idea.

¹¹ <http://pixlr.com/express>, accessed Feb 21, 2014

¹² <http://www.photoshop.com/tools>, accessed Feb 21, 2014

8.6 Summary

In this chapter, we have discussed the main findings from this dissertation from a number of perspectives, and identified implications of each. Summarizing this discussion,

- The sources of performance gains we observed in our study of Workflows can be traced back to the key components of our task-centric interface design.
- Task-centric interfaces support a compromise between different types of learning and performance on unfamiliar tasks. We suggest this space of tradeoffs could be more fully explored in future work.
- Keyword search enables application learning at the conceptual level of tasks, which has potential benefits in terms of scalability and memorability, but our current design could do more to help users formulate search queries. We suggest providing proactive support for learning relevant terminology and formulating queries.
- Task-centric interfaces reduce the solution space the user needs to explore in order to perform an unfamiliar task. We suggest some ways that task-centric interfaces could further narrow the space of solutions, and guidelines for effectively presenting task knowledge.
- A valid use case for feature-rich software is to perform short, targeted tasks. We suggest that use of task-centric interfaces to support this could lead to a vision of computing in which applications are more like search engines than monolithic desktop applications.

In the next section, we provide a more comprehensive summary of the contributions of this dissertation and areas for future work.

Chapter 9

Impact and Opportunities for Future Work

In this dissertation, we make the following contributions:

1. Based on a study of two-years' worth of usage data gathered from a publicly-deployed, instrumented image editor, we provide evidence that a valid use case for feature-rich software is to perform short, targeted tasks using a small fraction of the available functionality.
2. We also replicate previous findings that individual users' command vocabularies tend to be small and idiosyncratic, and that command frequencies follow a long-tailed distribution, when considered across a community of users.
3. We argue that current *feature-centric* paradigms for organizing functionality in feature-rich software are poorly suited to the type of use that we have identified above, because they do not support users in performing unfamiliar tasks.
4. To explicitly support this scenario of use, we present the concept of a *task-centric interface*—an alternative interface paradigm for feature-rich software that is able to quickly adapt itself as needed to support the user in performing specific tasks.
5. We present AdaptableGIMP, an initial task-centric interface design that we have developed. Based on two user studies of AdaptableGIMP, we present evidence that users appreciate the features of a task-centric interface, and derive insights into how the design can be improved.
6. We present Workflows, an iteration on the task-centric interface design in AdaptableGIMP, and the main task-centric interface prototype that we have developed to evaluate the potential of task-centric interfaces.
7. Based on the results of a laboratory study held over two sessions, we find evidence that a task-centric interface design can provide advantages over current practices for using feature-rich software to perform unfamiliar tasks. In particular, we present evidence that:
 - a. A task-centric interface design can enable users to complete tasks significantly faster and with reduced cognitive load, both when performing an unfamiliar task for the first time, and re-performing that task at least two weeks later.

- b. Users express a subjective preference for the task-centric interface design over current practices.
- 8. We present evidence for three main mechanisms by which task-centric interfaces provide the benefits listed above:
 - a. The task adaptation provided by the task-centric interface helps users to locate relevant functionality for a task.
 - b. The task knowledge provided by the task-centric interface helps users to form a plan for how to achieve their intended goal.
 - c. By explicitly supporting the process of performing an unfamiliar task, the task-centric interface reduces self-directed exploration of the interface, which is a common source of frustration and errors.
- 9. We present evidence that a task declaration mechanism based on keyword search allows users to engage in *keyword learning* to efficiently return to task-centric help resources. This enables the user to learn the interface at a higher conceptual level of tasks, rather than learning lower-level functionality and procedures. Keyword learning also has potential advantages in terms of scalability, because a few keywords can map to an arbitrarily complex set of commands and procedure, and memorability, because the keywords themselves are descriptive of the task being learned.
- 10. Based on the results of all of the studies described above, we outline specific areas for future work to continue to develop the idea of task-centric interfaces.

9.1 Conclusions

Based on the research we have presented in this dissertation, we reach the following conclusions about task-centric user interfaces. For our chosen scope and use scenario (non-expert users performing unfamiliar tasks), we found that a task-centric user interface can effectively enable a guided-and-constrained problem solving strategy that confers advantages in terms of improved task performance and reduced cognitive load. The main sources of benefit provided by the task-centric interface come from (1) revealing the relevant commands for an intended task; (2) providing guidance on how to use these commands together to reach the task goal; and (3) by way of 1 and 2, constraining and guiding the user to a relevant subset of the interface, making this activity more

productive and avoiding frustration and errors that stem from self-guided exploration and trial-and-error experimentation in a feature-centric interface.

Based on the findings above, we conclude that a task-centric interface design can provide benefits over the current feature-centric interfaces to feature-rich software.

In the next section, we discuss open research problems, and opportunities for work to further understand and develop the task-centric interface concept.

9.2 Opportunities for Future Work

In the previous chapter we suggested a few ways to further develop the task-centric interface provided by Workflows. In this section, we highlight additional areas for future work, with a focus on investigating the idea of task-centric interfaces more broadly.

9.2.1 Understanding User Behavior in Feature-Rich Software

Our findings raise a number of questions about user behavior in feature-rich software that could be investigated with additional studies. In particular, it would be interesting to further investigate the phenomenon of keyword learning, influences on users' choice of problem-solving strategies, and how users react to task-centric interfaces when they can only provide imperfect task support. We discuss each of these areas in detail below.

9.2.1.1 Keyword Learning and Use of Feature-Rich Software

As discussed in the previous chapter, we found evidence that users learn keywords that allow them to return to task-centric help resources when they need to re-perform tasks in feature-rich software. We have also argued that this type of application learning has the potential to be more scalable and more memorable than learning lower-level commands and procedures.

Additional studies could be conducted to fully understand the keyword learning phenomenon in feature-rich software, and how it compares to learning lower-level commands and procedures.

9.2.1.2 Understanding Choice of Problem-Solving Strategies

In this work, we've investigated how task-centric interfaces change use of feature-rich software *when they are adopted and used*. Having now established that a task-centric interface can provide advantages in this case, there are a number of further questions we could investigate about users' motivations to use this kind of system.

First, there is the question of how motivated users will be to use a task-centric interface that is presented alongside a feature-centric interface, and the related question of how to motivate users to use the task-centric interface when it is appropriate to do so.

Second, comments by some participants in our study of Workflows suggested that they perceive the task-centric interface as a barrier to efficiently performing actions in the interface. This raises the question of what exactly is creating this perception in users. It may be that this is purely related to performance. Along these lines, it would be interesting to investigate how much a task-centric interface can close the gap between novice and expert use of an interface. It could also be that the act of formulating a keyword search to enter into the task-centric interface requires more conscious effort than navigating to known tools in an interface, which is perceived as being effortless. If this latter explanation is true, it would be interesting to see if techniques could be developed to reduce the perceived cognitive cost of issuing searches to the point that participants no longer felt this was a problem. Finally, it could be that conceptual similarities between task-centric interfaces and traditional help systems (which are typically not perceived as being helpful (Rettig 1991)) are creating a negative perception of task-centric interfaces as an efficient method of accomplishing tasks.

Finally, more work could be done to understand why users engage in inefficient and error-prone interface-exploration strategies in feature-rich software. Some initial work has been done to understand users' preference for these strategies: Trudel and Payne (Trudel and Payne 1995) suggest that the cause is "unthinking interaction" in which the user rapidly tries actions without reflecting on the results of each, and Novick, Andrade, and Bean suggests four potential factors behind this preference (Novick, Andrade, and Bean 2009). However, there is currently no work that provides a comprehensive characterization of the factors that influence a user's choice of problem-solving strategy, or the specific triggers that cause a user to switch between strategies while performing a task. A deeper understanding of this phenomenon could potentially feed into the design of systems to persuade users to use task-centric interfaces or other assistance mechanisms.

9.2.1.3 Understanding and Designing for Imperfect Task Support

In this work, we've investigated the scenario where a user wishes to perform a particular task for which a single appropriate customization is available through the task-centric interface. We chose this as our focus because it is a necessary precondition for the task-centric interface approach working

more broadly. However, having now established benefits in this scenario, more work could be done to understand the impact of task-centric support in other situations.

A number of variables could be manipulated to create a range of situations users might encounter when using a task-centric interface, including: the number of task-specific customizations that are available for the intended task (none, one, multiple); how closely the available customizations fit with the user's intended task (good fit, imperfect fit but can be adapted, poor fit that leads to incorrect results); and whether multiple customizations must be used in concert with one another to complete the intended task. Investigating how users react in these situations could yield insights into the human factors surrounding use of task-centric interface, and how these could be supported in refined designs. For example, how good does the support provided by a task-centric interface need to be to ensure that users trust and regularly use the system?

9.2.2 Usability at the Task Level

Once tasks are included as first-class objects within the interface, we can take any existing concerns that can be applied to lower-level functionality (e.g. discoverability, learnability, efficiency) and apply it to tasks as well. For example, we could consider *task discoverability*, or how the interface helps the user to discover the tasks that can be performed in the application. Task discovery mechanisms could communicate tasks that are similar to the user's current task, or help users to understand the range of different goals that can be achieved using the application.

As another example, consider the concern of efficiency as applied to tasks. Efficiency in this context could mean supporting rapid access to specific task adaptations, or supporting fast switching between adaptations. In our Workflows design, we provided one mechanism to support this (the Recent Searches bar), but one could imagine a range of other mechanisms, such as more efficient keyboard shortcuts to directly access, or cycle through, commonly used workflows.

We can also consider how concerns such as discoverability or efficiency apply *across* the conceptual levels of tasks and lower-level functionality. Continuing the example of discoverability above, the interface could support discovery of which tasks are related to particular commands, or vice versa. For this particular example, work by Fourney, Mann, and Terry on Query-Feature graphs has looked at how task-command relationships could be discovered, and argued for a number of potential benefits that this information could provide (Fourney, Mann, and Terry 2011b).

9.2.3 Exploring the Space of Task-Centric Interface Designs

Returning to the components of a task-centric interface that we discussed in Chapter 4, our Workflows prototype represents only one point in a space of potential task-centric interface designs. In particular, we've focused on a task-centric interface design in which the user initiates task adaptation through a keyword-search based task-declaration mechanism. A rich area for future work would be to explore other areas of this design space.

For example, we could imagine a task-centric interface design in which task knowledge, including specific uses, and *incorrect uses*, of commands was presented in the tooltips for those commands in the interface. In such a design, the tooltip for the *Image>Transform>Flip Vertically* command could include a statement such as “This command flips the *entire image*, including all layers. To flip an *individual layer*, see *Layer>Transform>Flip Vertically* instead.”, and include an actionable reference to the *Layer>Transform>Flip Vertically* command. Further extending this idea, these “task aware” tooltips could be customized based on a model of the current task the user is engaged in (e.g. as inferred by the commands they’ve recently used).

This approach would provide a natural way to address the problem of red-herring commands that we observed in our study, because users would naturally come across information to lead them back on track while exploring the interface.

Extending this idea even further, elements of the application other than commands could be made task-aware as well. For example, sliders for parameter values could be altered to highlight regions most likely to yield desirable results for the current task.

Though the design described above represents a very different vision for supporting use of feature-rich software than Workflows, it is also based on the four components of a task-centric interface that we discussed in Chapter 4:

Tasks as First-Class Objects – In the task-aware tooltips, the system explicitly represents the connection between specific goals, and the tools that are (and are not) applicable to those tasks.

Task Awareness – The system infers the user’s current task, based on their recent actions.

Task Adaptation – The contents of tooltips (which act as simple interface customizations) are changed in response to the user’s current activities.

Task Knowledge – The content of the tooltips provides additional information to help the user to understand how commands and features relate to achieving specific, meaningful goals in the application.

As this example illustrates, varying these four components can lead to a wide range of different task-centric interface designs.

9.2.3.1 Purely Task-Centric Interfaces

We could also imagine a purely task-centric interface that consists solely of a search box when the application is first opened, and acts like a search engine for accessing self-contained applications for performing particular ill-defined tasks.

A simple form of this kind of interaction is already supported on the web. For example, typing mathematical expressions (e.g. “ $2+3$ ”) or unit conversions (e.g. “70 Fahrenheit in Celsius”) into Google search returns simple interactive interfaces for performing these operations. In addition, third party web pages exist that provide step-by-step instructions and customized interfaces for performing more complex operations, such as particular statistical tests (e.g. the GraphPad chi square calculator¹³).

Extending this idea, we could imagine web-based, task-centric applications that forgo providing a feature-centric interface entirely, and instead present a portal for accessing a vast repository of task-specific interfaces. Such an interface would provide a number of avenues for further developing task-centric interfaces:

Supporting À La Carte Usage – As discussed in the previous chapter, time spent on finding and installing software acts as a barrier to à la carte usage. A web-based task-centric application could be loaded in a matter of seconds on any computer, and so would be naturally suited to supporting this usage scenario.

Supporting à la carte usage on the web raises additional questions about how to best orient a user to a custom interface that they’ve never used before, in a domain they may not be familiar with.

One Application, Many Users – Web-based software has a different conceptual relationship with its user community than desktop software. Instead of each user having their own installation of the software, web-based applications act as one monolithic application that many users connect to. A

¹³ <http://graphpad.com/quickcalcs/chisquared1.cfm>, accessed May 18, 2014

potential advantage to this architecture is that users may be more accepting of subtle changes in an interface from visit to visit, which would allow task-specific customizations to be refined incrementally over time.

This architecture also creates a rich set of possibilities for how a task-centric interface could be improved based on usage data gathered from a huge number of users. For example, relevance feedback could be used to refine which task-centric customizations are most appropriate for a given keyword search. The application could also model which tool settings are most commonly used for particular tasks, and present this data back to the user.

We have explored some ideas along these lines in work on Community Enhanced Tutorials (Lafreniere, Grossman, and Fitzmaurice 2013), which explored how video demonstrations could be gathered from individual users following a tutorial on the web, and then presented as additional demonstrations to subsequent users.

A Social Domain for Feature-Rich Software – The web is accepted as a social domain, and the one-application-many-users model naturally lends itself to features that support collaboration and social interaction. This opens up possibilities for embedding community feedback and refinement mechanisms into task-centric interfaces. For example, individual customizations could include Q&A features, so their content could be refined by the community over time.

Some work has already been done in this area. Chilana et al. created the LemonAID system, which integrates Q&A help directly into the interfaces of web-based applications (Chilana, Ko, and Wobbrock 2012; Chilana et al. 2013), and Matejka et al. explored the idea of integrating Q&A functionality into the interface of AutoCAD (Matejka, Grossman, and Fitzmaurice 2011). Finally, our own work on task-centric community customization, discussed in Appendix A, explored how a community of users could collectively create and refine a set of task-specific interface customizations.

9.2.4 Toolkit Support

Another interesting area for future work would be to develop a *task-centric interface toolkit* that could be used to add a task-centric interface component to existing feature-rich applications. To provide an interface of the type presented in Workflows, a toolkit would need the following components:

Workflow Database – The main component of the toolkit would be a database to store the library of task-centric interface customizations accessible through the interface.

User Interface – This component would present an interface for searching for customizations and displaying a currently loaded customization. In our Workflows prototype, the user interface was implemented using an embedded browser widget, so only minor changes were needed to integrate the interface into the application.

Keyword Search – When the user issues a keyword search, this component would return matching workflows from the Workflows Database.

Command Execution – When users interact with a workflow (e.g. by clicking an actionable command), this component would handle executing the corresponding command in the application.

What makes the idea of a task-centric interface toolkit compelling is that almost all of the components above could be implemented once, and then reused across multiple applications with only subtle changes. The main component that would need to be modified for each particular application is the Command Execution component, which would need to interact with the application at a deeper level to invoke individual commands.

A task-centric interface toolkit could even conceivably be implemented using a cloud-based, software as a service model, where the task-centric application is implemented on a central server, and application developers simply need to make minor adjustments to their applications to hook into the service.

9.2.5 Building a Comprehensive Library of Task-Centric Customizations

One aspect of a task-centric interface that we have not yet discussed is how a comprehensive library of task-centric interface customizations might be created. In this section, we discuss some ideas for future work in this area.

9.2.5.1 Task-Centric Community Customization

AdaptableGIMP was created as part of a larger investigation into the idea of *task-centric community customization*. The idea was to provide users with a task-centric interface, but to also provide a means for the user community of an application to collectively create, refine, organize, and document a set of interface customizations used by the task-centric interface.

In Appendix A, we discuss the idea of task-centric community customization, and our investigation of this idea in AdaptableGIMP. We also present the results of a semi-structured interview study and a ten-month public deployment, which provided insights into these capabilities of AdaptableGIMP.

Continuing to develop the idea of task-centric community customization offers one route toward creating a library of task-centric customizations.

9.2.5.2 Converting Web-Based Tutorials

For applications with large, established user bases, rich collections of tutorials already exist on the web. Thus, another approach to creating task-centric customizations is to investigate how these existing task-centric help resources could be converted into customizations. For the laboratory study reported in Chapter 7, we developed a principled approach for converting web-based tutorials into task-centric customizations, which could be used as a starting point for further work in this area.

It may also be possible to develop algorithms that perform some portion of the conversion automatically. Recent work by Fourney et al. has demonstrated the use of machine learning techniques to accurately identifying references to interactive interface elements in web-based tutorial content (Fourney et al. 2012), which could act as a starting point for more detailed task modeling. However, additional work by Fourney and Terry suggests that there are many challenges to overcome before a system could automatically derive accurate and complete representations of tasks from web-based content (Fourney and Terry 2014). This suggests that, for now, machine learning techniques may need to be combined with contributions from humans. Human contributions could come from skilled experts or the “crowd” on micro-task marketplaces such as Amazon’s Mechanical Turk.

9.2.5.3 Prioritizing Customization Creation Efforts

Regardless of the approach used to create customizations, a great deal of effort will be involved in creating a comprehensive library. Thus, another area to explore is how this effort could be prioritized based on how likely a customization is to be used by the community of users. This could be used to prioritize efforts of contributors, or to best allocate funds if directly paying for content.

Work on CUTS (Characterizing Usability Through Search) by Fourney, Mann, and Terry (Fourney, Mann, and Terry 2011a) has explored how search query logs can be used to derive a prioritized list of tasks being performed by a community of users of interactive applications. This technique could be applied to web search (the approach used by Fourney, Mann, and Terry) to derive a list of common tasks for an application planning to add a task-centric interface. If a task-centric interface is already deployed and being used by a community of users, the keyword searches issued through its interface could be analyzed in this way as well.

Finally, prior work has shown that creators of web-based tutorials often do so for extrinsic motivations, including earning money from ads (Lafreniere et al. 2012). Given this, a market-based approach could be adopted to prioritize creation of customizations, by providing rewards to creators of customizations in proportion to how much a customization is used by the community.

9.2.6 Applications for Task-Centric Interfaces

Finally, an interesting area for future work would be to investigate how task-centric interfaces could be applied to very different application domains than feature-rich software for design and creation.

One domain in particular that seems well-suited for task-centric interfaces is statistical modeling and data analysis. Experts in this area have developed a rich collection of techniques and sophisticated processes for deriving insights from data, but these increasingly need to be put to use by non-experts¹⁴. Moreover, non-experts may only occasionally need to apply statistical analysis techniques, such as when analyzing the results of an experiment or study. Finally, speaking anecdotally, a common current practice for performing statistical data analysis is to consult cookbook-like sites, such as *Statistics for HCI Research*¹⁵ to learn the particular steps required to analyze data in a particular scenario. Together, this describes a scenario very similar to that which we have focused on in this thesis, and designed task-centric interfaces to support.

Task-centric interfaces for statistical data analysis could help occasional practitioners of statistical data analysis to use the proper techniques to analyze data in a given scenario. They could also help make advanced analysis techniques, such as Bayesian data analysis, accessible to a wider group of practitioners.

9.3 Summary

In this chapter, we've reviewed the contributions of this thesis, and outlined seven areas where future work could be done in this area, including:

- User studies to understand the keyword learning phenomenon, users' choice of problem-solving strategies, and users' reactions when task-centric interfaces cannot provide ideal support.

¹⁴ It has been argued that the vast majority of statistical data analysis is no-longer being performed by experts, because data has become cheap and easy to collect: <http://simplystatistics.org/2013/06/14/the-vast-majority-of-statistical-analysis-is-not-performed-by-statisticians/>, February 23, 2014

¹⁵ <http://yatani.jp/HCIstats/HomePage>, Accessed February 23, 2014

- Reconsidering usability concerns such as discoverability or learnability at the higher conceptual level of tasks.
- Exploring the design space of task-centric interface designs, including purely task-centric interfaces.
- Creating a toolkit for easily integrating task-centric interface into existing applications, including offering task-centric interface as a cloud service.
- Investigating techniques for building a comprehensive library of task-centric interface customizations, for use in a task-centric interface.
- Applying task-centric interfaces to additional domains, such as statistical data analysis.

Overall, continuing to develop the idea of task-centric interfaces constitutes a rich space for future work, and we hope that the specific avenues we've highlighted will guide future work in this area.

Appendix A

Task-Centric Community Customization

As discussed in Chapter 5, our first task-centric interface design in AdaptableGIMP, was part of a larger investigation of the idea of *task-centric community customization*. The idea was to provide users with a task-centric interface, but to also provide a means for the user community of an application to *create, refine, organize, and document* a set of interface customizations used by the task-centric interface (i.e. to directly engage the user community of an application in building the body of task-centric knowledge necessary to power a task-centric interface).

The promise of this approach is that it could enable a task-centric interface to naturally adapt to the needs of the user community. In the ideal, task-centric community customization:

- Enables the user community to direct their efforts toward creating the task-centric knowledge that best reflects their needs;
- Allows the task-centric interface of an application to adapt and follow the needs of a user community as they change; and
- Documents task knowledge in the terminology and vocabulary used by members of the user community.

In the rest of this chapter, we start by briefly reviewing prior research to motivate and position the idea of task-centric community customization. Next, we present a set of functional requirements that we derived for the design of this type of interface, and describe how these functional requirements were met in AdaptableGIMP. We then present some additional results from the semi-structured interview study we presented in Chapter 5, and findings from a ten-month public deployment of AdaptableGIMP, which provided some initial insights into the idea of task-centric community customization.

A.1 Motivation and Related Work

A.1.1 The Social Practices of Customization

As discussed in Chapter 2, a robust finding from the literature is that users are hesitant to spend time on concerns not directly related to their current task, including time spent on interface customization. However, past work has also indicated that when users *do* choose to spend time customizing, social

factors often play a role. Users with greater knowledge often help less knowledgeable users by sharing their expertise, or by creating and sharing customizations (Gantt and Nardi 1992; Kahler 2001a; Mackay 1990; Mackay 1991; MacLean et al. 1990); and the act of seeing or hearing about another user's customizations can trigger a user to customize their own software (Mackay 1991).

Motivated by the social practices surrounding customization, Kahler proposed the notion of collaborative tailoring (Kahler 2001a), or the idea of providing explicit support for sharing customizations in an application, and synthesized a list of design desiderata for collaborative tailoring (see (Kahler 2001b)). Task-centric community customization can be considered to be a collaborative tailoring approach, but it addresses a number of issues beyond Kahler's design recommendations, including long-term refinement of customizations by multiple users, and the design of mechanisms for finding and managing customizations that scale to the size of the entire user community of an application.

A.1.2 Community-Generated Documentation

The web has enabled grassroots support communities to arise around popular feature-rich applications. As two examples of this phenomenon, the tutorial aggregator site Tutorialized includes links to more than 16,000 tutorials for Adobe Photoshop¹⁶, and Gimptalk, a GIMP user forum, has more than 17,000 registered members¹⁷ and active forums devoted to user-created tutorials and tips. The size of these support communities indicates the willingness (and ability) of users to provide support to one another. Moreover, past work has shown that users make use of community created resources for support while performing unfamiliar tasks (Lafreniere et al. 2013), making these resources a kind of de-facto help system.

The intent of task-centric community customization is to create a system that provides explicit support for this practice. In pursuing this goal, we are inspired by the example of StackOverflow¹⁸, which has demonstrated that a carefully designed system can enable users to build up a rich and reusable collection of task-specific information in the programming domain (Mamykina et al. 2011).

¹⁶ <http://www.tutorialized.com/tutorials/Photoshop>, Accessed February 12, 2014

¹⁷ <http://www.gimptalk.com>, Accessed February 12, 2014

¹⁸ <http://www.stackoverflow.com>, Accessed February 13, 2014

A.2 Community Customization in AdaptableGIMP

The AdaptableGIMP application works in conjunction with a publicly accessible central wiki at www.adaptablegimp.org (the AdaptableGIMP wiki). The wiki acts as a central location to coordinate the creation and editing of task sets and task set documentation by the community.

In Chapter 5 we described how task sets can be searched for, installed, and used in AdaptableGIMP. In this section, we describe how task sets can be created, shared, documented, and refined by a community of users. We also briefly discuss our rationale for the design decisions we made in AdaptableGIMP.

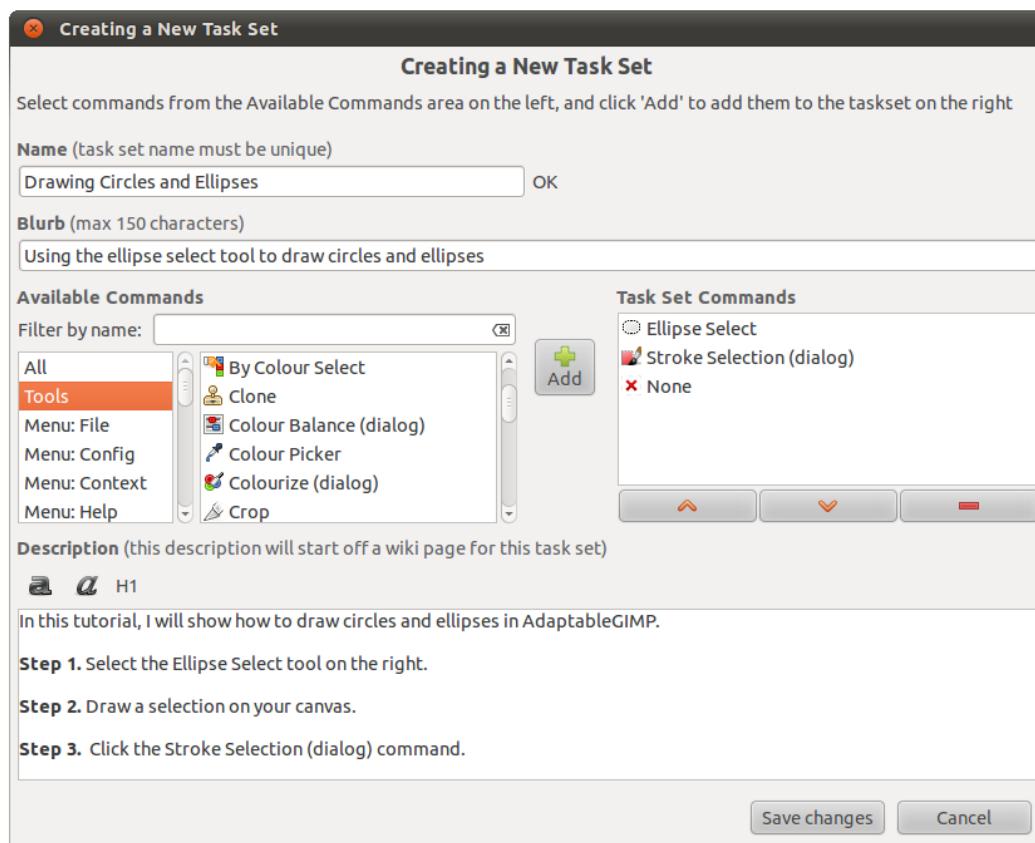


Figure 17. Creating a task set in AdaptableGIMP.

A.2.1 Creating and Sharing Customizations

To create a new task set in AdaptableGIMP, the user chooses the commands for the task, names the task set, and manually enters documentation (Figure 17). When the task set is saved, it is automatically uploaded to the AdaptableGIMP wiki, where it is represented by a new wiki page. Once

on the wiki, the task set is available to all other users of the application and will begin to show up in searches. From this point on, any user is free to modify the task set and its documentation, using a similar model to large-scale wikis such as Wikipedia.

Our choice to make sharing the default for newly-created customizations was intended to emphasize the community nature of the application, and to frame customizations as being primarily a community resource.

Our main overriding concern in designing mechanisms for creating and sharing task sets was to make the process as easy as possible to use, to encourage this behavior. With that in mind, we tried to make the above process as straightforward as possible. That having been said, our approach does require manual actions on the part of the user. Perhaps the ideal mechanism for creating customizations would be completely automatic; the user would perform a task, and the system would record their actions and create a fully-documented customization for performing that task. Promising work in this direction has explored the automatic creation of tutorials for photo-manipulation applications (Grabler et al. 2009a).

While we focused on reducing disincentives to creating and sharing customizations, an equally valid way to motivate users would be to provide them with compelling incentives, for example through a reputation system. This approach has been employed to good effect in other social computing systems, such as StackOverflow (Mamykina et al. 2011).

A.2.2 Community Refinement of Customizations

As well as supporting creation and sharing of customizations, a task-centric community customization system should enable the community to collaboratively refine customizations and their documentation over time. This enables the community to share improvements to techniques for performing particular tasks, and allow the system to adapt to the specific needs of the user community.

Our AdaptableGIMP design satisfies this requirement by providing tools for collaborative editing of customizations and their documentation, and awareness mechanisms to notify users of changes made by the community.

In this section, we present these features of AdaptableGIMP, as well as the model we developed for how changes to a customization on the wiki affect current users of a customization the application.

A.2.2.1 Supporting Collaboration

In considering the question of how to support a community of users collaboratively creating a body of task-specific knowledge about a software application, we wanted a solution that would:

- *Provide a flexible medium for storing, structuring, and linking together task-centric knowledge about a software application;*
- *Support discussion around customizations*, to allow users to engage in collaborative sense-making about methods for performing tasks in the software, as well as to build a sense of community around the project;
- *Provide a user account system*, to give users an identity within the community and to allow them to be recognized for their contributions;
- *Enable users to create derivations (“forks”) of a customization*, to resolve differences of opinion regarding how to accomplish a task; and
- *Provide history or revision tracking mechanisms*, to encourage users to edit without fear of doing something irreparable, and to minimize damage by antisocial users.

Given the complexity of this set of design desiderata, and the known challenges of developing groupware systems (e.g. see (Grudin 1994)), we decided not to develop our own groupware system from scratch, and instead explored existing solutions that could be adapted. This led to our decision to use a modified wiki as the repository for customizations and their documentation in AdaptableGIMP.

Because wikis have been honed over time to support the collaborative creation, linking, and structuring of massive amounts of information, they natively support the majority of the requirements listed above. Wikis also come with a set of social norms and practices for collective refinement of knowledge, though it’s an open question whether these norms will smoothly transfer to this new domain.

The AdaptableGIMP wiki was based on a modified version of the popular MediaWiki¹⁹ software used by Wikipedia and other large wiki sites. Customizations and their documentation are represented as standard wiki pages with XML embedded in the markup. For example, a task set for drawing rectangles (a task not natively supported by GIMP) is represented by the following XML:

¹⁹ <http://www.mediawiki.org>, February 13, 2014

```

<taskset blurb="How to draw rectangles in GIMP">
  <command gimpname="gimp-rect-select-tool"/>
  <command gimpname="select-stroke"/>
  <command gimpname="select-none"/>
</taskset>

```

Because customizations and their documentation are represented using text-based formats, the practice of “forking” a customization (i.e., duplicating to create a derivation of the original) simply requires creating a new wiki page and copying the markup text from an existing page.

A.2.2.2 Awareness Mechanisms

Inspired by MediaWiki’s dynamic “Recent Changes” page, we created a dynamic “Browse Task Sets” page on the AdaptableGIMP wiki that clusters task sets based on real-time community usage data. For example, we list task sets according to the length of time people typically use the task set for, the number of times it has been used, and the number of installations of the task set.

We also modified the built-in change tracking system in MediaWiki to distinguish between edits to a task set’s documentation and changes to the set of commands included in a task set (see Figure 18). Our rationale for this feature was that minor edits to the text of a wiki page may be less important to a contributor than edits that add or remove commands from a task set.

21 April 2011	
• (diff hist) . .	N TaskSet:Paintbrush and Pencil; 20:23 . . (+174) . . User (Talk contribs) - Failed to load diff data
• (diff hist) . .	N TaskSet:Simple tools for drawing; 20:20 . . (+191) . . User (Talk contribs) - Failed to load diff data
• (diff hist) . .	TaskSet:Paths; 19:10 . . (-84) . . Filkr (Talk contribs)
Commands added/removed:	
+ Copy Path	
+ Paste Path	
- Lower Path	
- Lower Path to Bottom	
- Raise Path	
- Raise Path to Top	
• (diff hist) . .	TaskSet:Sepia Photo; 19:09 . . (-21) . . Filkr (Talk contribs)
• (diff hist) . .	TaskSet:Drawing Basic Shapes; 19:09 . . (-79) . . Filkr (Talk contribs)
• (diff hist) . .	TaskSet:Drawing Rectangles and Squares; 19:09 . . (-22) . . Filkr (Talk contribs)
• (diff hist) . .	TaskSet:Drawing Complex Shapes (Path Tool); 19:09 . . (-109) . . Filkr (Talk contribs)

Figure 18. Change tracking for a task set stored on the AdaptableGIMP wiki.

A.2.2.3 Community Improvement vs. Providing a Stable Interface

At the heart of the concept of task-centric community customization is the goal of making it easy for the community to learn from one another and increase their efficiency with the software. When one user learns a more efficient way to perform a task, we want to facilitate the dissemination of that new knowledge to other users.

While the sharing of customizations provides the primary vehicle for achieving this goal, we also felt that any sharing mechanism should not cause undue disruption to end-users. For example, users may react badly if community changes to customizations were immediately reflected in AdaptableGIMP.

This led to a set of goals and constraints that are in conflict with one another. We wanted to:

- Encourage continuous improvement of task sets by the user community, but give users a stable, predictable user interface;
- Allow users to make small personal modifications to task sets to fit their individual use of the application, while still encouraging dissemination of improvements to the user community; and
- Enable users to easily form an accurate mental model of how changing a customization affects the copy of a customization in their interface, and the shared copy in the community repository

After exploring a range of potential ways to meet these goals and constraints, we adopted a model for AdaptableGIMP in which *edits on the wiki stay on the wiki, while edits in the application stay in the application*. The one exception to this general rule is newly created customizations, for which a copy is automatically uploaded to the wiki, as we mentioned previously.

Because changes to community customizations are not automatically pushed to users, it is necessary to provide mechanisms to notify users when the community version of customizations they have installed in AdaptableGIMP have been updated, and provide a way to merge changes from the community into locally installed task sets. AdaptableGIMP addresses these needs by displaying a “Latest Wiki Update” status for the currently selected task set, and by providing a mechanism to view and merge changes from the community, using a similar interface to that for creating task sets in Figure 17.

In the preceding sections, we've described how AdaptableGIMP supports community collaboration on creating task sets and their documentation. We now present insights that we gained into these features from a laboratory study, and a ten-month public deployment.

A.3 Feedback on Task-Centric Community Customization

In addition to the results presented in Chapter 5, our semi-structured interview study probed participants' attitudes towards creating customizations in AdaptableGIMP, and sharing customizations with the rest of the user community.

The majority of participants reported that they could envision themselves creating task sets, with only three of the ten participants expressed reluctance to do so. Participants stated that their willingness to create task sets stemmed from desires to increase personal efficiency, to document a workflow, to provide help to other users, and to keep with the open source spirit. The following quotes reflect these attitudes:

Yes I can [envision creating task sets]. Specifically, a while ago, a friend who was going to get married, she asked me to get a slideshow ready for her. So she e-mails me a ton of pictures, all different sizes, different ratios, different shapes... So I wanted to first get them into all the same size and [was] doing the same task repetitively over and over. So in that case – I could see a small set would be very helpful... So, in some scenarios, I would definitely create a task set. [P6]

For myself, it seems like a good way of keeping track of processes. For whatever reason, I often find myself figuring out something and immediately writing it down, so it is kind of part of the process anyways, so then you might as well put it out there. [P7]

A sense of philanthropy I suppose. It is just fun to see people using things that you've created. It gives you a sense of pride I guess. [P4]

In addition to expressing a willingness to contribute task sets, most participants responded positively to AdaptableGIMP's policy of automatically publishing task sets, and the ability for anyone in the community to edit a task set. Participants expressed appreciation for the concept of open information access and the notion of community refinement, particularly with the goal of learning from others:

It would be nice too, if you found somebody that was good, being able to download their tool configurations and see how they work and maybe that would help you improve. So I don't see why anybody should have a problem with that. It's not really an invasion of privacy or anything. [P5]

Well, it would be cool if somebody else finds it helpful and improved it hopefully. [P6]

I'm okay with that, as long as I don't have to download their updates. I'd like to be able to see when that happens and to be notified and [I] could check it out and see that this person [...] has used a series of actions that actually makes things a lot easier. [P2]

However, participants also raised some concerns about sharing customizations. P1 felt that individuals would develop a sense of ownership over task sets, which would distinguish the AdaptableGIMP wiki from sites like Wikipedia. This sense of ownership made him uncomfortable with the idea of other users modifying a customization that he had created:

It's a little weird. Wikipedia got popular because it's information – everybody can share information. But this is a tool I created and published in a way and that I have more ownership over. If somebody made a branch of it: "I took it and added some stuff to it" – that would be neat. But I would be a little bit uncomfortable with somebody changing the tool set... It is an ownership thing. [P1]

P4 also echoed this sense of ownership, and felt uncomfortable with the idea of other users making modifications without some form of approval process:

It would depend on what the specific modifications were and how I felt they fit into the scope of what the task set was trying to do. But that's just the risk that you take with a wiki-based community like that. It's a community effort, not a collection of individual efforts. If somebody legitimately decides, "Oh I think this set of functions should be in this task set, maybe this and this shouldn't be." Maybe the new version shouldn't be accepted outright as the best possible solution, but it is definitely something that merits discussion. I guess I have mixed feelings about it. [P4]

P4 was also uncomfortable with the automatic publishing of task sets, and wanted the option to prototype a task set before releasing it. In particular, he cited concerns about misleading other users, or getting a reputation for sharing low quality task sets with the community:

I would prefer to have the option to save it locally because if you are prototyping a task set, you might want to experiment with it yourself a little while first before releasing it to people who might get confused by it. If they don't understand what you were intending to accomplish when making the set, then it might be confusing and people might associate your username with low quality sets that are not ready yet. [P4]

A final concern raised in regards to automatic publishing of task sets was that users might create task sets that do not correspond to particular tasks, but are simply personal collections of tools.

In summary, participants responded well to the idea of contributing to a collection of task sets, but raised concerns regarding ownership over task sets, and the automatic publishing of newly created task sets to the community. Specifically, the feedback provided by participants suggest some of the social dimensions to task-centric community customization, including the desire to help other users,

and keep with an open source ethic, but also issues of personal reputation, and not wanting to cause harm users by sharing a poor task set.

In the next section, we discuss the results of a ten-month public deployment of AdaptableGIMP, in which observed further evidence for some of these phenomena.

A.4 Deployment and Response

AdaptableGIMP was publicly released in March 2011. In the ten months after its initial release, it was installed more than 1,600 times, and more than 700 users registered accounts on the AdaptableGIMP wiki. In this section, we provide some observations on the actions of users during this initial deployment period.

Despite enjoying a large response in installations and user accounts, only 21 task sets were created by users who were not associated with the project, and we did not observe any collaboration between users on task sets (i.e. none of the 21 task sets was edited by a user other than its original author). As well, we noted a trend where authors would include their username in the title of the task sets that they created. For example, the task set “Normen Kontrast” was created by user Normen, and “DlwFigColorToEnhancedGrayscale” was created by DLWood60. In total, 6 of the 21 task sets contained some variation on the author’s username in their title.

These findings provide further evidence of concerns regarding sharing and ownership over task sets to those reported in the previous section.

The majority of task sets created by users also included minimal documentation, though this was not universal. A “Torn paper / ripped paper effect” task set was created by a user who made edits several times over a four day period to ultimately create a seven step long tutorial including detailed step-by-step instructions and images documenting each step. However, this was the exception, rather than the rule.

This finding suggests that the design of AdaptableGIMP may not be placing enough emphasis on the creation of task knowledge. This may also be a result of the task-centric interface component of AdaptableGIMP not placing enough emphasis on the documentation associated with task sets (recall that fewer than half of the participants in our semi-structured interview study utilized task set documentation).

A.5 Summary and Discussion

In this chapter, we have presented the concept of task-centric community customization, which seeks to enable a community of users to collectively create and refine a collection of task-centric interface customizations. We also describe our implementation of this idea in AdaptableGIMP, and insights into this idea gathered in a semi-structured interview study, and a ten-month public deployment.

The results of our study and deployment suggest that the idea of task-centric community customization is promising. In principle, participants expressed a willingness to create and share customizations, and we observed that some users of AdaptableGIMP did take the time to create and share task sets, including one user that created a fully-documented task set.

Despite these encouraging results, the user community as a whole only created a relatively small number of customizations, and our study found evidence that users have concerns regarding the mechanisms for community contribution. Without further study it's difficult to say the exact reasons for these observations, but we can offer a few speculations:

First, the design of AdaptableGIMP, and its community contribution mechanisms in particular, are undoubtedly playing a role in shaping user contribution. In particular, the results of our semi-structured interview study suggest that the automatic sharing of customizations may not be sensitive to the social implications of publicly sharing a customization. As well, by not sufficiently emphasizing documentation in the task-centric interface provided by AdaptableGIMP, we may have led users to focus on creating task sets, but not their documentation.

Second, it may be that a larger user community is required to create the critical mass necessary to spur contribution. In AdaptableGIMP, the chief incentive for creating a task set is the possibility that it might be used by other people, which is proportional to the size of the user community.

Finally, it may be that more time is needed for a set of social norms and practices to develop to fit this new medium. While the Wikipedia community collaborates on a body of factual information, methods of performing tasks in an application are more subjective. This may lead users to feel protective of their own methods of performing tasks, and conversely hesitant to correct other users' methods of performing tasks. An interesting open question is how to design tools to enable productive cooperation in this type of medium.

Overall, the insights from our semi-structured interview study and initial ten-month public deployment are encouraging, and suggest avenues to continue to develop the idea of task-centric community customization.

Appendix B

Study Materials

To facilitate replication, this appendix includes materials developed for the study of Workflows presented in Chapter 7.

B.1 Pre-Task Questionnaire

Consider the before/after image for the task, and the resources that are available to you in the study, and then indicate how much you agree or disagree with the following statement:

“I am confident that I can complete this task.”

Strongly Disagree					Strongly Agree
1	2	3	4	5	

B.2 Post-Study Questionnaire

Questions for all tasks

“I preferred completing tasks using the...”

GIMP + Web browser		GIMP + Workflows panel		
1	2	3	4	5

Questions for the GIMP + Web tasks

“It was easy to find help using the Web.”

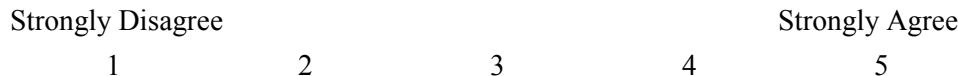
Strongly Disagree		Strongly Agree		
1	2	3	4	5

The overall quality of the Web for helping me to complete the tasks:

Poor		Excellent		
1	2	3	4	5

Questions for the GIMP + Workflow panel tasks

“It was easy to find help using the Workflows panel.”



The overall quality of the Workflows panel for helping me to complete the tasks:



B.3 Workflows

In this section, we present the thirteen workflows we authored for this study. Commands that were clickable in the individual workflows are indicated in grey rectangles below.

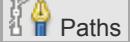
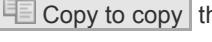
Straighten a Horizon

How to straighten a photo with a crooked or skewed horizon.

1. Use  Open... to open the photo with the crooked horizon.
2. Use the  Rotate tool to fix the tilt in the horizon by clicking and dragging the image.
3. Use the  Crop tool to trim the edges of the image by selecting the area you want to keep, and hitting **Enter**.

How to Copy & Paste

How to copy and paste objects from one image to another.

1. Click  Open... to open the images you want to work with.
2. Use the  Paths tool to outline the item you want to copy by left-clicking around the item's border. To close the path, hold the **Ctrl** key and click the first point you added.
3. Click  Path to Selection to create a selection from the path you just created.
4. Click  Copy to copy the selection.
5. Switch to the other document.

6. Click Paste to paste the selection.
7. **(Optional)** Use the Move tool to position the pasted object. Use the Scale tool to scale the pasted object.
8. Click Save As... to save your work.

Selective Colorization

How to convert a color photograph to black and white with color restored to certain areas.

1. Use Duplicate to create a copy of the original image.
2. Click Grayscale to convert the image to greyscale. Click RGB to convert the image mode back to RGB (this will not restore the color, but is a necessary step).
3. Switch to the color image. Click New Layer... to create a new layer.
4. In the Layers dialog, select the layer you just created.
5. Switch to the greyscale image. Click All. Click Copy.
6. Switch to the color image. Click Paste to paste the greyscale copy of the image. Click Anchor Layer to anchor the pasted selection.
7. In the Layers dialog, select the greyscale layer. Click Add Layer Mask... to add a layer mask.
Use the following settings:
 - o Initialize Layer Mask to: **White (Full Opacity)**
8. Set the foreground color to black. Use the Paintbrush tool to paint the parts of the image you want color to show through.

Borders on Selections

How to draw shapes (circles, squares, rectangles, etc.).

1. Use one or more selection tools to create a selection with the desired shape.
 Rectangle Select,
 Ellipse Select, Free Select, Fuzzy Select, By Colour Select,
 Intelligent Scissors
2. **(Optional)** In the Brushes dialog select a brush you want to use to draw the outline.
3. Click Stroke Selection... to create an outline around the selection.

Easy Reflection

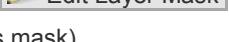
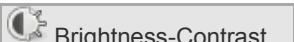
How to create a mirrored text effect.

1. Click New... to create a new image.
2. Use the Bucket Fill tool to create a black background.
3. Set the foreground color to the desired color for your text.
4. Use the Text tool to create the text that you want to mirror.
5. Use the Move tool to position your text.
6. **(Optional)** Use the Dodge / Burn tool to add some texture to the text.
7. In the Layers dialog, select the text layer. Click Duplicate Layer to duplicate the layer.
8. In the Layers dialog, select the duplicated layer. Click Flip Horizontally. Click Rotate 180°.
9. Use the Move tool to position the upside down text directly below the original.
10. Select the Eraser tool. In the Tool Options dialog, set the following settings:
 - Opacity: **50**
 - Brush: **Hardness 025**
 - Size: **150**
11. In the Layers dialog, select the layer with the upside down text. Use the Eraser tool to erase the bottom half and darken the reflection.

Simple Vignetting

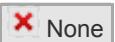
How to add a vignette effect to photographs in which the image appears darker near the corners.

1. Click Open... to open your source photograph.
2. Click Duplicate Layer to duplicate the layer containing the image.
3. Click Add Layer Mask... to add a layer mask to the newly created layer. Use the following settings:
 - Initialize Layer Mask to: **White (Full Opacity)**
4. Click Default Colors to set the colors to their default value.

5. Select the  Blend tool. In the  Tool Options dialog, use the following settings:
 - Gradient: **FG to BG (RGB)**
 - Shape: **Radial**
6. Click and drag in the image window to create a radial gradient on the layer mask (you'll see the gradient in the layer mask icon in the  Layers dialog, but not in the image window).
7. Click  Edit Layer Mask to toggle editing of the layer mask (once clicked, you should be editing the layer, not its mask).
8. Click  Brightness-Contrast... to decrease the brightness and increase the contrast of the layer (because of the layer mask, you will only see a change at the edges of the image).

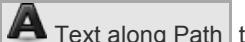
How to Produce Glossy Text

How to create text with a Web 2.0 glossy text style.

1. Use the  Text tool to create some text.
2. In the  Layers dialog, select the text layer. Click  Duplicate Layer to duplicate the layer.
3. Click  Alpha to Selection to create a selection with the same shape as the text.
4. Select the  Rectangle Select tool. Hold the **Ctrl** key and click and drag to create a rectangle that covers the bottom half of the text (this will deselect the bottom half of the text).
5. Select the  Blend tool. Set the foreground color to **black** and the background color to **grey**. Click and drag to draw a gradient over the top half of the text.
6. Click  None to clear the selection.

Text Along Path

How to make text follow a path (for example, a curve or a circle).

1. Use the  Text tool to create your text somewhere on the canvas (it doesn't matter where).
2. Use the  Paths tool to draw the curved path that you want the text to follow.
3. In the  Layers dialog, select the layer containing your text.
4. Click  Text along Path to create a new path in the shape of your text, along the path you created above.

5. Click  Path to Selection to create a selection from the path.
6. Click  New Layer... to create a new layer for the curved text. Use the following settings:
 - o Layer Fill Type: **Transparency**
7. Set the foreground color to the desired color for your text. Click  Fill with FG Color to fill the selection with the foreground color.
8. Click  None to clear the selection.
9. In the  Paths dialog, hide the two paths you created earlier by clicking the eye icon beside each.

How to Outline Text

How to add an outline around text.

1. Use the  Text tool to add some text to the canvas.
2. Click  Text to Path to convert the text to a path.
3. Click  New Layer... to create a new layer for the outline. Use the following settings:
 - o Layer Fill Type: **Transparency**
4. In the  Layers, drag the layer you just created so it is below the text layer.
5. Click  Path to Selection to create a selection from the path you created earlier.
6. Click  Grow... to grow the selection; this will determine how thick your outline is.
7. Use the  Bucket Fill to fill the selection with the desired color.
8. Click  None to clear the selection.

Simple Scanlines

How to create a simple scanline effect.

1. Click  Open... to open the images you want to work with.
2. Click  New Layer... to create a new empty layer.
3. Set the foreground color to white. Use the  Bucket Fill tool to fill the layer you just created.
4. Click  Erase Every Other Row... .

5. Use the  Scale tool to enlarge the layer by clicking the document. Use the following settings:
 - o Width: **200 percent**
 - o Height: **200 percent**
6. In the  Layers dialog, set the Mode to **Overlay**.

Straight Line Tutorial

How to draw straight lines.

1. Use the  Paintbrush tool to set the start location for the line by left clicking once in the desired location.
2. Hold the **Shift** key and left click where you want the line to end.

Reduce the number of colors in a raster image

How to reduce the size of the palette for an image.

1. Click  Indexed... to convert the image to Indexed mode. Use the following settings:
 - o Colormap: **Generate optimum palette**
 - o Maximum number of colors: **(your choice)**
2. Click  RGB to convert the image back to RGB color mode (your changes from the last step will remain).

Simple Animations

How to create animated GIFs.

1. In the  Layers dialog, create a layer for each frame of the animation.
2. **(Optional)** In the  Layers dialog you can set the duration for each frame by adding the desired duration in milliseconds to the layer's name (e.g. "Layer 1 (500ms)").
3. Click  Export... to save the animation as a **GIF Image** with a filename ending in **.gif**. Use the following settings:
 - o GIF Options: **As animation**

References

- “Adobe Labs Tutorial Builder.” 2012. <http://labs.adobe.com/technologies/tutorialbuilder/>.
- Andrade, Oscar D., Nathaniel Bean, and David G. Novick. 2009. “The Macro-Structure of Use of Help.” In *Proceedings of the 27th ACM International Conference on Design of Communication*, 143–50. New York, NY, USA: ACM.
- Aula, Anne, Natalie Jhaveri, and Mika Käki. 2005. “Information Search and Re-Access Strategies of Experienced Web Users.” In *Proceedings of the 14th International Conference on World Wide Web*, 583–92. New York, NY, USA: ACM.
- Bergman, Lawrence, Vittorio Castelli, Tessa Lau, and Daniel Oblinger. 2005. “DocWizards: A System for Authoring Follow-Me Documentation Wizards.” In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, 191–200. New York, NY, USA: ACM.
- Bernstein, Michael S, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David R Karger, David Crowell, and Katrina Panovich. 2010. “Soylent: A Word Processor with a Crowd inside.” In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology*, 313–22. New York, NY, USA: ACM.
- Berthouzoz, Floraine, Wilmot Li, Mira Dontcheva, and Maneesh Agrawala. 2011. “A Framework for Content-Adaptive Photo Manipulation Macros: Application to Face, Landscape, and Global Manipulations.” *ACM Trans. Graph.* 30 (5): 120:1–120:14.
- Bunt, Andrea, Cristina Conati, and Joanna McGrenere. 2004. “What Role Can Adaptive Support Play in an Adaptable System?” In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, 117–24. New York, NY, USA: ACM.
- . 2007. “Supporting Interface Customization Using a Mixed-Initiative Approach.” In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, 92–101. New York, NY, USA: ACM.
- Carroll, John M., and Caroline Carrithers. 1984. “Training Wheels in a User Interface.” *Commun. ACM* 27 (8): 800–806.
- Carroll, John M. 1990. *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. MIT Press.
- Carroll, John M., Robert L. Mack, Clayton H. Lewis, Nancy L. Grischkowsky, and Scott R. Robertson. 1985. “Exploring Exploring a Word Processor.” *Hum.-Comput. Interact.* 1 (3): 283–307.
- Carroll, John M., and Mary Beth Rosson. 1987. “Paradox of the Active User.” In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, 80–111. MIT Press.
- Carroll, John M., Penny L. Smith-Kerker, James R. Ford, and Sandra A. Mazur-Rimetz. 1987. “The Minimal Manual.” *Hum.-Comput. Interact.* 3 (2): 123–53.
- Chi, Pei-Yu, Sally Ahn, Amanda Ren, Mira Dontcheva, Wilmot Li, and Björn Hartmann. 2012. “MixT: Automatic Generation of Step-by-Step Mixed Media Tutorials.” In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, 93–102. ACM.

- Chilana, Parmit K., Andrew J. Ko, and Jacob O. Wobbrock. 2012. “LemonAid: Selection-Based Crowdsourced Contextual Help for Web Applications.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1549–58. New York, NY, USA: ACM.
- Chilana, Parmit K., Andrew J. Ko, Jacob O. Wobbrock, and Tovi Grossman. 2013. “A Multi-Site Field Study of Crowdsourced Contextual Help: Usage and Perspectives of End Users and Software Teams.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 217–26. New York, NY, USA: ACM.
- Cockburn, Andy, Carl Gutwin, and Saul Greenberg. 2007. “A Predictive Model of Menu Performance.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 627–36. New York, NY, USA: ACM.
- Cook, Ronny, Judy Kay, Greg Ryan, and Richard C. Thomas. 1995. “A Toolkit for Appraising the Long-Term Usability of a Text Editor.” *Software Quality Journal* 4 (2): 131–54.
- Corbin, Juliet, and Anselm Strauss. 2007. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. 3rd ed. Sage Publications, Inc.
- Csikszentmihalyi, Mihaly. 2008. *Flow: The Psychology of Optimal Experience*. Harper Perennial Modern Classics.
- Draper, Stephen W. 1984. “The Nature of Expertise in UNIX.” In *Proceedings of INTERACT '84 IFIP Conference on Human-Computer Interaction*, 465–71. Elsevier North-Holland.
- Dryer, D. Christopher. 1997. “Wizards, Guides, and beyond: Rational and Empirical Methods for Selecting Optimal Intelligent User Interface Agents.” In *Proceedings of the 2nd International Conference on Intelligent User Interfaces*, 265–68. New York, NY, USA: ACM.
- Dyck, Jeff, David Pinelle, Barry Brown, and Carl Gutwin. 2003. “Learning from Games: HCI Design Innovations in Entertainment Software.” In *Proceedings of Graphics Interface 2003*, 237–46. A K Peters.
- Ekstrand, Michael, Wei Li, Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2011. “Searching for Software Learning Resources Using Application Context.” In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 195–204. New York, NY, USA: ACM.
- Fernquist, Jennifer, Tovi Grossman, and George Fitzmaurice. 2011. “Sketch-Sketch Revolution: An Engaging Tutorial System for Guided Sketching and Application Learning.” In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 373–82. New York, NY, USA: ACM.
- Findlater, Leah, Karyn Moffatt, Joanna McGrenere, and Jessica Dawson. 2009. “Ephemeral Adaptation: The Use of Gradual Onset to Improve Menu Selection Performance.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1655–64. New York, NY, USA: ACM.
- Fischer, G. 1993. “Shared Knowledge in Cooperative Problem-Solving Systems---Integrating Adaptive and Adaptable Components.” In *Adaptive User Interfaces: Principles and Practice*, 49–68. Elsevier Science Publishers.
- Fourney, Adam, Ben Lafreniere, Richard Mann, and Michael Terry. 2012. “‘Then Click Ok!’: Extracting References to Interface Elements in Online Documentation.” In *Proceedings of the*

- SIGCHI Conference on Human Factors in Computing Systems*, 35–38. New York, NY, USA: ACM.
- Fourney, Adam, Richard Mann, and Michael Terry. 2011a. “Characterizing the Usability of Interactive Applications through Query Log Analysis.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1817–26. Vancouver, BC, Canada: ACM.
- . 2011b. “Query-Feature Graphs: Bridging User Vocabulary and System Functionality.” In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 207–16. New York, NY, USA: ACM.
- Fourney, Adam, and Michael Terry. 2014. “Mining Online Software Tutorials: Challenges and Open Problems.” In *CHI ’14 Extended Abstracts on Human Factors in Computing Systems (submitted)*. New York, NY, USA: ACM.
- Gajos, Krzysztof Z., Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. 2006. “Exploring the Design Space for Adaptive Graphical User Interfaces.” In *Proceedings of the Working Conference on Advanced Visual Interfaces*, 201–8. New York, NY, USA: ACM.
- Gajos, Krzysztof Z., Katherine Everitt, Desney S. Tan, Mary Czerwinski, and Daniel S. Weld. 2008. “Predictability and Accuracy in Adaptive User Interfaces.” In *Proceedings of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems*, 1271–74. New York, NY, USA: ACM.
- Gantt, Michelle, and Bonnie A Nardi. 1992. “Gardeners and Gurus: Patterns of Cooperation among CAD Users.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 107–17. New York, NY, USA: ACM.
- “Google SketchUp Training.” 2012. <http://sketchup.google.com/intl/en/training/index.html>.
- Grabler, Floraine, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. 2009a. “Generating Photo Manipulation Tutorials by Demonstration.” In *SIGGRAPH ’09*, 1–9. ACM.
- . 2009b. “Generating Photo Manipulation Tutorials by Demonstration.” *ACM Trans. Graph.* 28 (3): 66:1–66:9.
- Greenberg, Saul. 1993. *The Computer User as Toolsmith: The Use, Reuse, and Organization of Computer-Based Tools*. New York, NY, USA: Cambridge University Press.
- Grudin, Jonathan. 1994. “Groupware and Social Dynamics: Eight Challenges for Developers.” *Commun. ACM* 37 (1): 92–105.
- Hanson, Stephen José, Robert E. Kraut, and James M. Farber. 1984. “Interface Design and Multivariate Analysis of UNIX Command Use.” *ACM Trans. Inf. Syst.* 2 (1): 42–57.
- Harris, Jensen. 2008. “The Story of the Ribbon” presented at the UX09, April 6.
- Hart, SG, and LE Stavenland. 1988. “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research.” In *Human Mental Workload*, edited by PA Hancock and N Meshkati, 139–83. Elsevier.
- Hick, W. E. 1952. “On the Rate of Gain of Information.” *The Quarterly Journal of Experimental Psychology* 4: 11–26.

- Höök, K. 2000. "Steps to Take before Intelligent User Interfaces Become Real." *Interacting with Computers* 12 (4): 409–26.
- Hyman, Ray. 1953. "Stimulus Information as a Determinant of Reaction Time." *Journal of Experimental Psychology* 45 (3): 188–96.
- Kahler, Helge. 2001a. "More than WORDs - Collaborative Tailoring of a Word Processor." *Journal of Universal Computer Science* 7 (9): 826–47.
- . 2001b. "Supporting Collaborative Tailoring (PhD Thesis)". PhD thesis, Roskilde University.
- Kay, Judy, and Richard C. Thomas. 1995. "Studying Long-Term System Use." *Commun. ACM* 38 (7): 61–69.
- Kelleher, Caitlin, and Randy Pausch. 2005. "Stencils-Based Tutorials: Design and Evaluation." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 541–50. New York, NY, USA: ACM.
- Knabe, Kevin. 1995. "Apple Guide: A Case Study in User-Aided Design of Online Help." In *CHI '95 Conference Companion on Human Factors in Computing Systems*, 286–87. New York, NY, USA: ACM.
- Kong, Nicholas, Tovi Grossman, Björn Hartmann, Maneesh Agrawala, and George Fitzmaurice. 2012. "Delta: A Tool for Representing and Comparing Workflows." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1027–36. New York, NY, USA: ACM.
- Kurtenbach, G., T. P. Moran, and W. Buxton. 1994. "Contextual Animation of Gestural Commands." *Computer Graphics Forum* 13 (5): 305–14.
- Lafreniere, Ben, Andrea Bunt, Matthew Lount, Filip Krynicki, and Michael A. Terry. 2011. "AdaptableGIMP: Designing a Socially-Adaptable Interface." In *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology*, 89–90. New York, NY, USA: ACM.
- Lafreniere, Ben, Andrea Bunt, Matthew Lount, and Michael Terry. 2012. "'Looks Cool, I'll Try This Later!': Understanding the Faces and Uses of Online Tutorials". Technical Report CS-2012-01. University of Waterloo, Technical Report CS-2012-01.
- . 2013. "Understanding the Roles and Uses of Web Tutorials." In *Proceedings of the 7th International AAAI Conference on Weblogs and Social Media*, 8 pages. AAAI.
- Lafreniere, Ben, Andrea Bunt, John Whissell, Charles L. A. Clarke, and Michael Terry. 2010. "Characterizing Large-Scale Use of a Direct Manipulation Application in the Wild." In *Proceedings of Graphics Interface 2010*, 11–18. Canadian Information Processing Society.
- Lafreniere, Ben, Tovi Grossman, and George Fitzmaurice. 2013. "Community Enhanced Tutorials: Improving Tutorials with Multiple Demonstrations." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1779–88. New York, NY, USA: ACM.
- Laput, Gierad, Eytan Adar, Mira Dontcheva, and Wilmot Li. 2012. "Tutorial-Based Interfaces for Cloud-Enabled Applications." In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, 113–22. ACM.

- Lau, Tessa, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. 2004. "Sheepdog: Learning Procedures for Technical Support." In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, 109–16. New York, NY, USA: ACM.
- Lazar, Jonathan, Adam Jones, and Ben Shneiderman. 2006. "Workplace User Frustration with Computers: An Exploratory Investigation of the Causes and Severity." *Behaviour and Information Technology* 25 (3): 239–51.
- Leshed, Gilly, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. "CoScripter: Automating & Sharing How-to Knowledge in the Enterprise." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1719–28. New York, NY, USA: ACM.
- Linton, Frank, Deborah Joy, Hans-Peter Schaefer, and Andrew Charron. 2000. "OWL: A Recommender System for Organization-Wide Learning." *Educational Technology & Society* 3 (1): 62–76.
- Linton, Joy, and Schaefer. 1999. "Building User and Expert Models by Long-Term Observation of Application Usage." In *UM '99 Proceedings of the Seventh International Conference on User Modeling*, 129–38. Secaucus, NJ, USA: Springer-Verlag.
- Mack, Robert L., Clayton H. Lewis, and John M. Carroll. 1983. "Learning to Use Word Processors: Problems and Prospects." *ACM Trans. Inf. Syst.* 1 (3): 254–71.
- Mackay, Wendy E. 1990. "Patterns of Sharing Customizable Software." In *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work*, 209–21. New York, NY, USA: ACM.
- . 1991. "Triggers and Barriers to Customizing Software." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 153–60. New York, NY, USA: ACM.
- MacLean, Allan, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. 1990. "User-Tailorable Systems: Pressing the Issues with Buttons." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 175–82. New York, NY, USA: ACM.
- Mamykina, Lena, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. 2011. "Design Lessons from the Fastest Q&A Site in the West." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2857–66. ACM.
- Matejka, Justin, Tovi Grossman, and George Fitzmaurice. 2011. "IP-QAT: In-Product Questions, Answers, & Tips." In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 175–84. New York, NY, USA: ACM.
- Matejka, Justin, Wei Li, Tovi Grossman, and George Fitzmaurice. 2009. "CommunityCommands: Command Recommendations for Software Applications." In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, 193–202. New York, NY, USA: ACM.
- McGrenere, Joanna, Ronald M Baecker, and Kellogg S Booth. 2002. "An Evaluation of a Multiple Interface Design Solution for Bloated Software." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 164–70. New York, NY, USA: ACM.
- McGrenere, Joanna, Ronald M. Baecker, and Kellogg S. Booth. 2007. "A Field Evaluation of an Adaptable Two-Interface Design for Feature-Rich Software." *ACM Trans. Comput.-Hum. Interact.* 14 (1).

- McGrenere, Joanna, and G. Moore. 2000. “Are We All in the Same ‘Bloat’?” In *Proceedings of the Graphics Interface 2000 Conference*, 187–96.
- Mori, Giulio, Fabio Paternò, and Carmen Santoro. 2002. “CTTE: Support for Developing and Analyzing Task Models for Interactive System Design.” *IEEE Trans. Softw. Eng.* 28 (8): 797–813.
- Murphy, G.C., M. Kersten, and L. Findlater. 2006. “How Are Java Software Developers Using the Eclipse IDE?” *IEEE Software* 23 (4): 76 –83.
- Norman, Donald A, and Stephen W Draper. 1986. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc.
- Norman, Kent L. 1991. *The Psychology of Menu Selection: Designing Cognitive Control at the Human/computer Interface*. Edited by Ben Shneiderman. Greenwood Publishing Group Inc.
- Novick, David G., Oscar D. Andrade, and Nathaniel Bean. 2009. “The Micro-Structure of Use of Help.” In *Proceedings of the 27th ACM International Conference on Design of Communication*, 97–104. New York, NY, USA: ACM.
- Obendorf, Hartmut, Harald Weinreich, Eelco Herder, and Matthias Mayer. 2007. “Web Page Revisitation Revisited: Implications of a Long-Term Click-Stream Study of Browser Usage.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 597–606. New York, NY, USA: ACM.
- Oppermann, Reinhard. 1994. “Adaptively Supported Adaptability.” *Int. J. Hum.-Comput. Stud.* 40 (3): 455–72.
- Rettig, Marc. 1991. “Nobody Reads Documentation.” *Commun. ACM* 34 (7): 19–24.
- Rieman, John. 1996. “A Field Study of Exploratory Learning Strategies.” *ACM Trans. Comput.-Hum. Interact.* 3 (3): 189–218.
- Scarr, Joey, Andy Cockburn, Carl Gutwin, and Andrea Bunt. 2012. “Improving Command Selection with CommandMaps.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 257–66. New York, NY, USA: ACM.
- Scarr, Joey, Andy Cockburn, Carl Gutwin, and Sylvain Malacia. 2013. “Testing the Robustness and Performance of Spatially Consistent Interfaces.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 3139–48. New York, NY, USA: ACM.
- Scarr, Joey, Andy Cockburn, Carl Gutwin, and Philip Quinn. 2011. “Dips and Ceilings: Understanding and Supporting Transitions to Expertise in User Interfaces.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2741–50. New York, NY, USA: ACM.
- Schön, Donald A. 1983. *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books.
- Shneiderman, Ben. 2002. “Promoting Universal Usability with Multi-Layer Interface Design.” *SIGCAPH Comput. Phys. Handicap.* (73-74): 1–8.
- Sutcliffe, and Old. 1987. “Do Users Know They Have User Models? Some Experiences in the Practice of User Modelling.” In *Proceedings of INTERACT '87 IFIP Conference on Human-Computer Interaction*, 35–41. Elsevier North-Holland.

- Teevan, Jaime, and Eytan Adar. 2007. "Information Re-Retrieval: Repeat Queries in Yahoo's Logs." In *In SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 151–58. ACM.
- Terry, Michael, Matthew Kay, B. Van Vugt, O. Slack, and Terry Park. 2008. "Ingimp: Introducing Instrumentation to an End-User Open Source Application." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 607–16. New York, NY, USA: ACM.
- Terry, Michael, and Elizabeth D. Mynatt. 2002. "Recognizing Creative Needs in User Interface Design." In *Proceedings of the 4th Conference on Creativity & Cognition*, 38–44. New York, NY, USA: ACM.
- Trudel, Carol-Ina, and Stephen J. Payne. 1995. "Reflection and Goal Management in Exploratory Learning." *Int. J. Hum.-Comput. Stud.* 42 (3): 307–39.
- Tuck, Robin, and Dan R. Olsen. 1990. "Help by Guided Tasks: Utilizing UIMS Knowledge." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Empowering People*, 71–78. New York, NY, USA: ACM.
- Whissell, J. S., C. L. A. Clarke, and A. Ashkan. 2009. "Clustering Web Queries." In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, 899–908. New York, NY, USA: ACM.
- Whiteside, John, Norman Archer, Dennis Wixon, and Michael Good. 1982. "How Do People Really Use Text Editors?" *ACM SIGOA Newsletter* 3 (1-2): 29–40.
- Yeh, Tom, Tsung-Hsiang Chang, and Robert C. Miller. 2009. "Sikuli: Using GUI Screenshots for Search and Automation." In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, 183–92. New York, NY, USA: ACM.
- Zaragoza, Hugo, Nick Craswell, Michael Taylor, Suchi Saria, and Stephen Robertson. 2004. "Microsoft Cambridge at TREC-13: Web and HARD Tracks." In *IN PROCEEDINGS OF TREC 2004*.