Web Search, Web Tutorials & Software Applications: Characterizing and Supporting the Coordinated Use of Online Resources for Performing Work in Feature-Rich Software

by

Adam Fourney

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, 2015

Statement of Contributions

This dissertation includes first-authored peer-reviewed material that has appeared in conference proceedings published by the Association for Computing Machinery (ACM). The publisher's policy on reuse of published materials in a dissertation is as follows[1]:

> *Authors can include partial or complete papers of their own (and no fee is expected) in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included.*

The following list serves as a declaration of the Versions of Record for works included in this dissertation:

Portions of Chapter 3:

Adam Fourney, Richard Mann, and Michael Terry. (2011) Characterizing the usability of interactive applications through query log analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '11). ACM, New York, NY, USA, 1817-1826.

`DOI=10.1145/1978942.1979205`
`http://doi.acm.org/10.1145/1978942.1979205`

Portions of Chapter 4:

Adam Fourney, Richard Mann, and Michael Terry. (2011) Query-feature graphs: bridging user vocabulary and system functionality. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST '11). ACM, New York, NY, USA, 207-216.

`DOI=10.1145/2047196.2047224`
`http://doi.acm.org/10.1145/2047196.2047224`

Portions of Chapter 5:

Adam Fourney, Ben Lafreniere, Parmit Chilana, and Michael Terry. (2014) InterTwine: creating interapplication information scent to support coordinated use of software. In *Proceedings of the 27th annual ACM symposium on User interface software and technology* (UIST '14). ACM, New York, NY, USA, 429-438.

`DOI=10.1145/2642918.2647420`
`http://doi.acm.org/10.1145/2642918.2647420`

[1]`http://authors.acm.org/main.html`, retrieved: May 13th, 2015

Portions of Appendix A:

Adam Fourney, 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* (CHI '12). ACM, New York, NY, USA, 35-38.

`DOI=10.1145/2207676.2207682`
`http://doi.acm.org/10.1145/2207676.2207682`

Abstract

Web search and other online resources serve an integral role in how people learn and use feature-rich software (e.g., Adobe Photoshop) on a daily basis. Users depend on web resources both as a first line of technical support, and as a means for coping with system complexity. For example, people rely on web resources to learn new tasks, to troubleshoot problems, or to remind themselves of key task details.

When users rely on web resources to support their work, their interactions are distributed over three user environments: (1) the search engine, (2) retrieved documents, and (3) the application's user interface. As users interact with these environments, their actions generate a rich set of signals that characterize how the population thinks about and uses software systems "in the wild," on a day-to-day basis. This dissertation presents three works that successively connect and associate signals and artifacts across these environments, thereby generating novel insights about users and their tasks, and enabling powerful new end-user tools and services. These three projects are as follows:

Characterizing usability through search (CUTS): The CUTS system demonstrates that aggregate logs of web search queries can be leveraged to identify common tasks and potential usability problems faced by the users of any publicly available interactive system. For example, in 2011 I examined query data for the Firefox web browser. Automated analysis uncovered approximately 150 variations of the query "Firefox how to get the menu bar back", with queries issued once every 32 minutes on average. Notably, this analysis did not depend on direct access to query logs. Instead, query suggestions services and online advertising valuations were leveraged to approximate aggregate query data. Nevertheless, these data proved to be timely, to have a high degree of ecological validity, and to be arguably less prone to self-selection bias than data gathered via traditional usability methods.

Query-feature graphs (QF-Graphs): Query-feature graphs are structures that map high-level descriptions of a user's goals to the specific features and commands relevant to achieving those goals in software. QF-graphs address an important instance of the more general vocabulary mismatch problem. For example, users of the GIMP photo manipulation software often want to "make a picture black and white", and fail to recognize the relevance of the applicable commands, which include: "desaturate", and "channel mixer". The key insights for building QF-graphs are that: (1) queries concisely express the user's goal in the user's own words, and (2) retrieved tutorials likely include both query terms, as well as terminology from the application's interface (e.g., the names of commands). QF-graphs are generated by mining these co-occurrences across thousands of query-tutorial pairings.

InterTwine: InterTwine explores interaction possibilities that arise when software applications, web search, and online support materials are directly integrated into a single productivity system. With InterTwine, actions in the web browser directly impact how information is presented in a software application, and vice versa. For example, when a user opens a web tutorial in their browser, the application's menus and tooltips are updated to highlight the commands mentioned therein. These embellishments are designed to help users orient themselves after switching between the web browser and the application. InterTwine also augments web search results to include details of past application use. Search snippets gain before and after pictures and other metadata detailing how the user's personal work document evolved the last time they visited the page. This feature was motivated by the observation that existing mechanisms (e.g., highlighting visited links) are often insufficient for recalling which resources were previously helpful vs. unhelpful for accomplishing a task.

Finally, the dissertation concludes with a discussion of the advantages, limitations and challenges of this research, and presents an outline for future work.

Acknowledgements

This thesis is the product of collaboration and cooperation; there are so many individuals to whom I owe thanks.

First, I would like to thank my supervisor and mentor Dr. Michael Terry. Dr. Terry provided me with much guidance, but also much freedom to explore and discover research topics that captivated my interest. I am in your debt.

Dr. Meredith Ringel Morris, Dr. Ryen W. White, and Dr. Eric Horvitz – I owe you special thanks for introducing me to industry research, and for mentoring me through two very successful academic internships. These internships afforded me a privileged opportunity to pursue my research on a scale not possible at a public institution.

I also owe special thanks to Dr. Benjamin Lafreniere and Dr. Parmit Chilana. Our research is well-aligned, and I've always found discussions with you to be incredibly useful and motivational. I am very glad to have had the opportunity to collaborate with you.

I would like to thank Dr. Richard Mann who was my supervisor during my Master's degree, is a coauthor on numerous papers covered by this dissertation, and has provided me with the tools I needed to successfully transition into the doctoral program.

Likewise, I would like to thank Dr. Charles Clarke for introducing me to the field of information retrieval, and providing me with guidance and encouragement in the early stages of this research.

Finally, I would like to thank my dissertation committee for considering my candidacy for a doctoral degree. And, I would like to thank all those who have participated in my various research studies over the years.

Dedication

I dedicate this thesis to Laura, my loving wife, and to my wonderful children Owen and Ethan. None of this would have been possible without your constant love, support, flexibility, and tremendous understanding of what it means for a family member to pursue a doctoral degree in computer science.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

When faced with a new task to accomplish, or problem to troubleshoot, users of interactive systems must often navigate the *Gulf of Execution*, a conceptual barrier that Donald Norman defines as "*the gulf between the user's goals and the way they must be specified to the system*" [97]. For example, users wishing to draw a circle in the GNU Image Manipulation Program (GIMP), must correctly translate this goal into a pair of lower-level actions: First, they must make an elliptical selection with GIMP's "*Ellipse Select*" tool. Then they may either invoke the "*Stroke Selection*" command to create an outlined circle, or they may employ the "*Bucket Fill*" tool to create a filled circle. In either case, what appears to be a simple task (drawing a circle) actually involves numerous discrete steps bearing little resemblance to the original expression of the goal – the Gulf of Execution is particularly difficult to negotiate in this instance.

For many people, web search and online resources play an integral role in helping to bridge gulfs of execution. Here, three steps define a common strategy (Figure 1.1, foreground): First, users issue a web search query describing their goal in their own words (e.g., [*gimp draw a circle*]). Next, users review the tutorials and online instructional material retrieved by the search engine. These documents detail the sequence of steps needed to perform the task. Finally, users must manually carry out the instructions by manipulating mechanisms in the system's user interface. Importantly, as users gain experience with the application, they often continue to rely on these external resources, having learned the path across the bridge (i.e., learning *where* to retrieve relevant task-specific information), rather than committing specific low-level application operations to memory [16, 77, 116]. This strategy results in frequent and repeated crossings of the web-mediated execution bridge by a broad set of users, across a range of skills and experience levels.

Notably, the three steps across the web-mediated bridge largely mirror those originally described in Norman's original writings [97] defining the Gulf of Execution in 1986. Specifically, Norman's execution bridge (Figure 1.1, background) involves: forming intentions, specifying action sequences, then executing those sequences in the system's user interface. In reconciling these two execution bridges, one notices that, in this con-

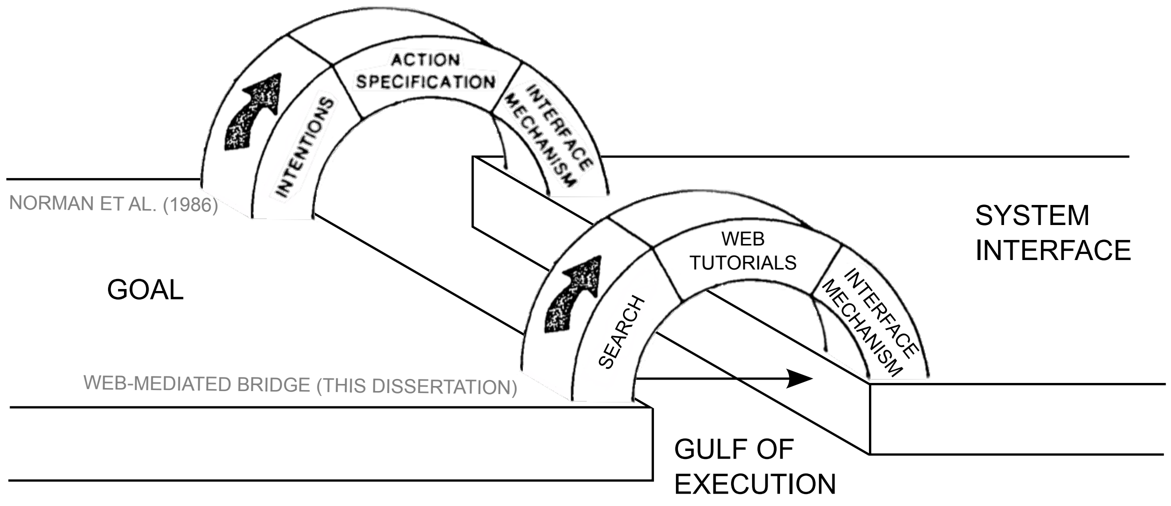


Figure 1.1: This figure, adapted from [97], depicts the *Gulf of Execution*. Users of interactive systems face the *Gulf of Execution* when they have a goal in mind, but are unsure (or cannot recall) which concrete low-level operations of the user interface will accomplish their task. Web search and online tutorials often serve an integral role in assisting users to bridge this gulf. Here, the 3 steps across the web-mediated bridge (foreground) mirror the three steps originally proposed by Donald Norman in [97] (background).

text, search queries express intentions, and online materials contain action specifications. Thus, in contrast to Norman's execution bridge where intention formation and action specification are entirely cognitive processes, each step across the web-mediated execution bridge is mediated by an interactive system. Namely, users interact with the web search engine, the retrieved document, and the application itself. Recognizing how these systems relate to one another, and collectively to the Gulf of Execution, reveals two highly significant and untapped opportunities for human-computer interaction (HCI) research. These opportunities are detailed below:

> **Web query logs, and online artifacts, afford novel opportunities for researchers and practitioners to better understand the day-to-day work practices of a feature-rich application's user base.** User actions across the web-mediated execution bridge can often be directly observed, and are *already* routinely logged and aggregated to support existing business practices and end-user functionality. For example, logs of search queries and page visits are routinely maintained to support personalized search [12, 120], or for advertising purposes [59]. Likewise, feature-rich software is often instrumented to help developers detect and remediate software bugs, as well as to measure user engagement [36]. Contextualizing these interaction logs as trails over the execution bridge affords opportunities to study hundreds of thousands of real-world, in situ, examples of users forming goals, then taking actions to accomplish those goals in feature-rich software. I argue that

these data allow researches and practitioners to study users and their work practices at a level of detail, scale, and ecological validity unprecedented in HCI research.

User goals (expressed as search queries) and action specifications (expressed in tutorials) can be reasoned about and acted upon automatically in software, enabling novel end-user tools and services. Consider again the opening example of drawing a circle in the GIMP software application. Techniques described in Chapter 3 allow one to estimate that people searched Google with queries similar to [*how to draw a circle in gimp*] nearly 10,000 times between October 2009 and August 2010 – or about once every hour on average[1]. In each of these thousands of cases, users explicitly stated their intention to use GIMP to draw a circle. And, they did so by typing text into a computer interface (i.e., web search). However, the GIMP software was never afforded an opportunity to respond to the users' clear statements of intent. Likewise, the search provider was never made aware of the actions ultimately taken by users to address their queries. I consider these to be missed opportunities. In particular, web search engines, online resources, and feature-rich desktop software exist today as separate entities that function independently from one another, requiring additional work from the user to manually link information across these three environments. I show that, by forming tighter integrations between each of the three steps across the web-mediated execution bridge, one can offer a range of novel end-user tools and services.

With this framing in mind, I now present the central thesis of this dissertation.

1.1 Thesis Statement

The remainder of this dissertation is structured around the following thesis statement:

> Web search and online resources (e.g., tutorials) serve an important role in supporting the work practices of the users of feature-rich software systems. Analysis of online interaction logs and artifacts (e.g., web query logs, or a corpus of tutorials) leads to novel and significant insights about a system's users, their tasks, and the breakdowns they experience on a daily basis. These interaction logs and artifacts can be leveraged, in an automated fashion, to support novel end-user tools and services that more directly integrate web search and online tutorials with feature-rich software.

To defend this thesis statement, this dissertation presents a series of discrete research endeavours, with each project exploring a different facet of the above claim. The remainder of this chapter briefly describes these projects, contextualizing each via the web-mediated execution bridge, as outlined in the next section.

[1]The query terms "gimp", "draw" and "circle" co-occurred in queries accounting for an estimated 9729 Google searches performed between October 2009 and August 2010.

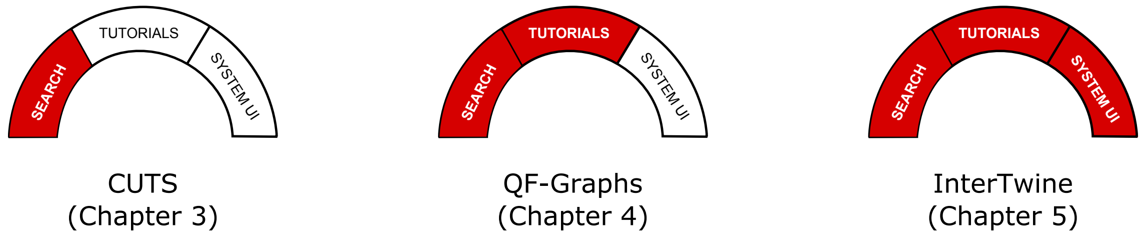


Figure 1.2: The three projects discussed in this thesis successively integrate adjacent steps along the web-mediated execution bridge which crosses the Gulf of Execution.

1.2 Specific Research Projects

This dissertation details three research projects[2] that successively connect the three steps across the web-mediated execution bridge (Figure 1.2). The journey begins in Chapter 3, which presents *CUTS* (Characterizing Usability through Search), a system that leverages web query data to characterize the day-to-day tasks and needs of a system's user population (Figure 1.2, left). These query data are then paired with a corpus of relevant web tutorials in Chapter 4, to create *Query-feature graphs*. Query-feature graphs are structures that help to addresses gulfs of execution exacerbated by the vocabulary problem [53], discussed in more detail below. Finally, Chapter 5 describes *InterTwine*, a system that explores opportunities that arise when web search, online resources, and software systems work together as a single coherent productivity system, uniting all three steps across the execution bridge (Figure 1.2, right). I describe each these systems in the sections that follow.

1.2.1 CUTS: Characterizing Usability Through Search (Chapter 3)

Occupying the first step across the web-mediated execution bridge, Chapter 3 argues that aggregates of web search engine query logs can be leveraged to identify common tasks and potential usability problems faced by the users of any publicly available interactive system. This potential is demonstrated through a system entitled CUTS (Characterizing Usability through Search), an automated process for harvesting, ordering, labelling, filtering, and grouping search queries to understand the day-to-day tasks and needs of a user base.

Rather than depending on direct access to query log data, which are not generally publicly available, CUTS leverages query autocompletion services (e.g., Google Suggest)

[2]Appendix A details a 4th related, but ancillary, research project.

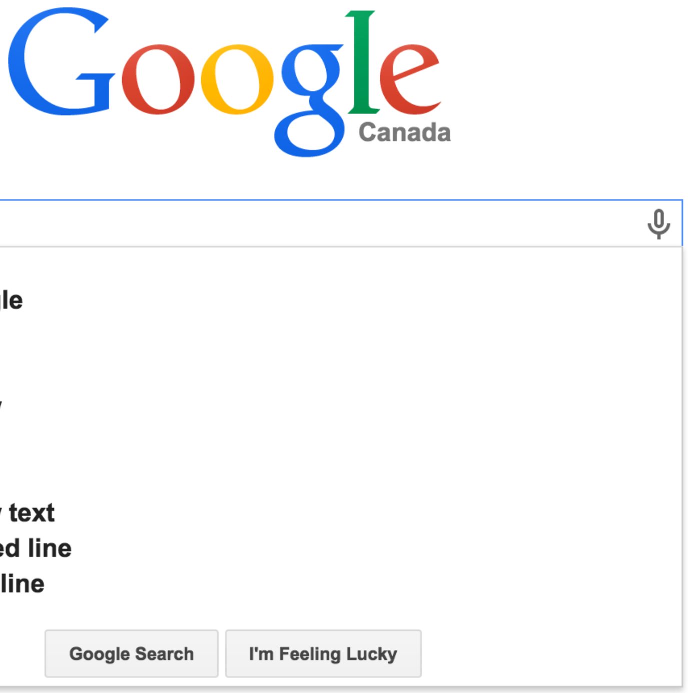


Figure 1.3: Google auto-complete suggestions for the prefix "gimp dr". Leveraging its query logs, Google correctly predicts my interest in using GIMP to draw a circle. The remaining 9 suggestions are other tasks for which GIMP users have often sought instructions in the past. (Retrieved May, 2015)

and online advertising valuations (e.g., Google AdWords) as proxies for query log aggregates. An example illustrates this potential: Consider again the goal of drawing a circle with the GIMP raster graphics editor. Simply typing the word "gimp" followed by the two characters "dr" into Google Search is sufficient for the search engine to surmise the task (Figure 1.3). This feat is possible because other users have encountered this specific gulf of execution in the past, and have leveraged web search to bridge the gap. By the same argument, the remaining suggestions depicted in figure 1.3 reveal 9 other goals, or specific execution gulfs, that users bridge with the help of web search and online documentation.

One direct application of CUTS is to identify *potential* usability problems in an application, and to provide a means of ranking issues according to the number of users impacted by these problems. For example, Chapter 3 reports that users of the Firefox web browser collectively sought information about recovering a missing menu bar (e.g., querying: [*firefox how to get the menu bar back*]) about once every 32 minutes on average in the period beginning October 2009, and ending in August 2010. The nature and frequency of this search topic suggested the existence of a software defect in the version of Firefox available at the time – why were users losing access to this important component of Firefox's user interface? An inspection of the Firefox user interface revealed that a poor design choice likely accounted for the high volume of web search on this topic. Chapter 3 describes this case study in more detail. This example demonstrates how CUTS can be leveraged to direct and target more costly HCI evaluation techniques, such as inspection by expert evaluators. The benefit of using the results of CUTS is that data are derived from the search behaviours of thousands, if not millions, of users.

Returning to the thesis statement of this dissertation, CUTS demonstrates that analyses of web search queries can lead to novel and significant insights about a feature-rich application's users, their tasks, and the breakdowns they experience on a daily basis. The claimed contributions of this work include:

- a methodology for approximating query data using publicly available interfaces
- a taxonomy characterizing the topics and likely intents of software-related search queries
- an automated method for distilling software-related query data into a ranked list of popular user tasks and breakdowns
- a series of case studies demonstrating the types of insights that can be generated from CUTS

1.2.2 Query-Feature Graphs (Chapter 4)

Chapter 4 presents query-feature graphs (QF-graphs), structures that map high-level descriptions of a user's goals to the specific features and commands relevant to achieving those goals in software (Figure 1.4). QF-graphs are motivated by the observation that there is often a discrepancy between the way users conceptualize and articulate their needs, as revealed by their search queries, and the (rather terse) technical vocabulary of the interactive system. This scenario is a modern instance of the of the more general vocabulary mismatch problem detailed in [53]. Vocabulary problems widen the Gulf of Execution, making it harder for users to map their goals to low-level actions in the software. For example, users of the GIMP photo manipulation software often want to [*make a picture black and white*]. Since GIMP has no command named "black and white," users who wish to achieve this effect must learn that commands such as "desaturate," "grayscale" or "channel mixer" will yield the desired effect. Indeed, none of these task-relevant commands shares any words in common with the user's search query.

The key insight for building QF-graphs is that, when retrieved using a search query, online tutorials serve as Rosetta stones for interaction: When a tutorial is retrieved through web search, it necessarily shares terms in common with the search query, and is thus likely expressed "in the user's language" (i.e., using terminology familiar to the user). Likewise, tutorials must also mention the specific names of commands, tools, and preferences necessary to accomplish the task in the application. As such, Query-feature

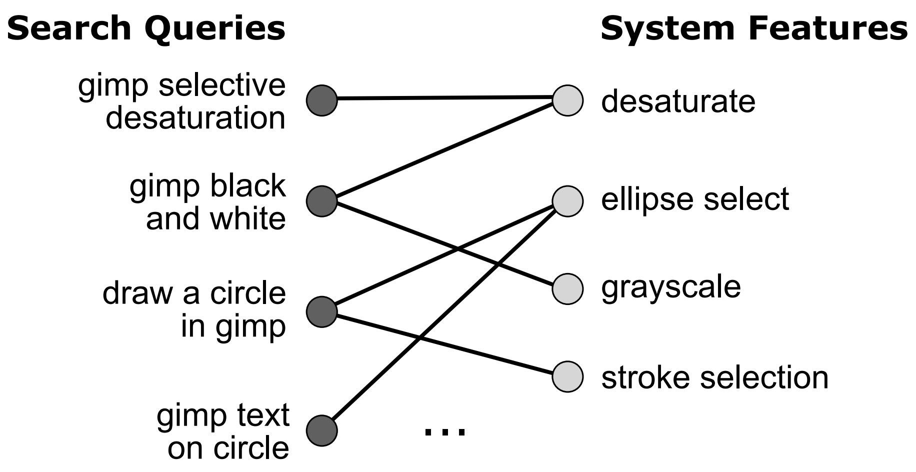


Figure 1.4: Query-feature graphs are weighted bipartite graphs with nodes representing queries on one side, and nodes representing system terminology on the other. These graphs pair tasks, as naturally expressed as user search queries, with relevant commands, features and settings in a system's user interface. This pairing enables a number of compelling end-user tools and services, many of which are discussed in Chapter 4. To construct Query-feature graphs, one leverages the co-occurrences of system commands and query keywords expressed in tutorials.

graphs are constructed automatically from mining the co-occurrences of query terms and software commands mentioned in relevant tutorials. In the context of the web-mediated execution bridge, query-feature graphs form connections between the first and second steps (Figure 1.2, center).

Query-feature graphs enable a wide variety of tools and services, three of which are explored in this dissertation: (1) a search-driven interface in which users type the task they wish to accomplish, and the interface assembles a list of the most relevant commands for the task; (2) command tooltips that leverage query data to display how a user community typically uses the tool in practice; and, (3) app-to-app analogy search, which provides a mapping between the tools necessary to perform a task in one interface and the equivalent tools in a second (e.g., Firefox's "New Private Window" command can be matched with comparable commands in other systems, such as Chrome's "New Incognito Tab").

Returning to the thesis statement of this dissertation, Chapter 4 demonstrates how information mined from search logs and online tutorials can be linked, using automated means, with the goal of learning a mapping between query terms and specific features of the software application. This mapping is instrumental in enabling a range of novel interactions (e.g., dynamic tooltips) that are designed to help users perform tasks in feature-rich software. Specific contributions claimed in this chapter include:

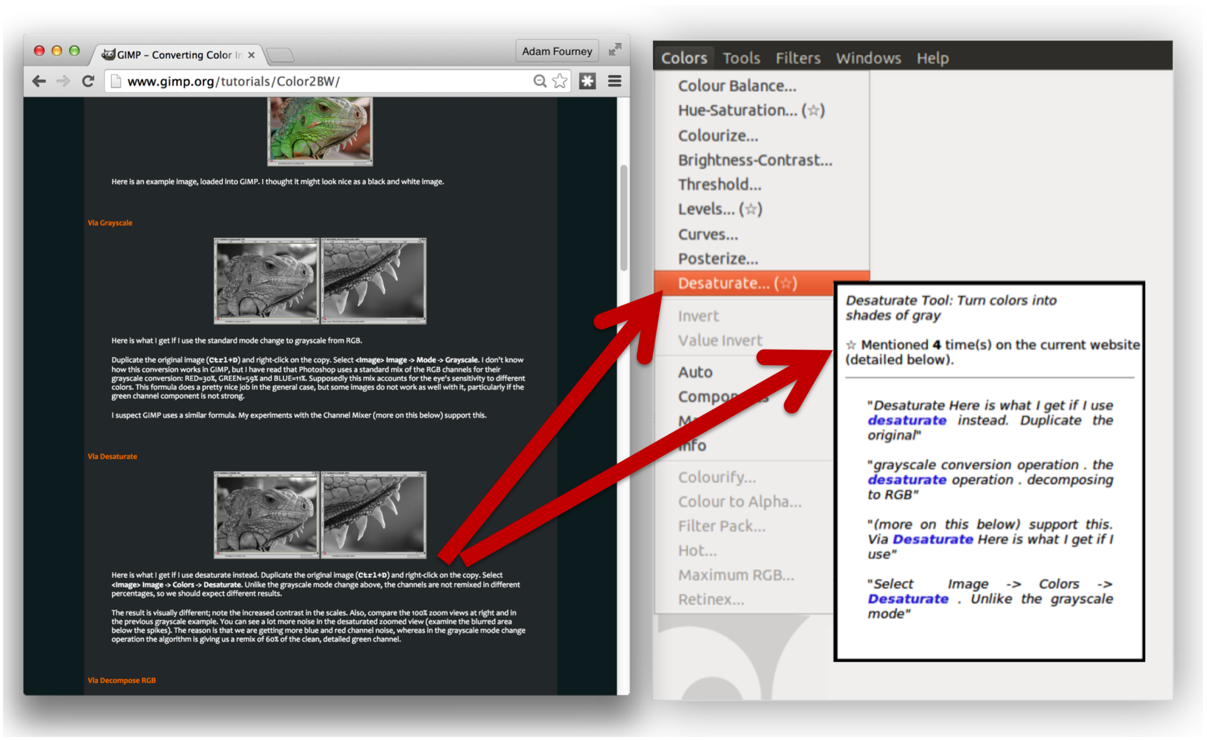
- the introduction and definition of query-feature graphs, structures that map high-level terminology to low-level features in complex software
- an automated method for constructing query-feature graphs from aggregate logs of web search data, paired with a corpus of online tutorials
- three example uses or applications of query feature graphs

1.2.3 InterTwine (Chapter 5)

Finally, Chapter 5 presents InterTwine, a research project integrating all three steps across the execution bridge (Figure 1.2, right). InterTwine explores the possibilities that arise when software applications, web search, and online support materials are directly integrated into a single productivity system. With InterTwine, actions in the web browser directly impact how information is presented in a software application, and vice versa. For example, when a user opens a web tutorial in their browser, the application's menus and tooltips are updated to highlight the commands mentioned therein (Figure 1.5). These embellishments are designed to help users orient themselves after switching between the web browser and the application. InterTwine also augments web search results to include details of past application use. Search snippets gain before and after pictures and other metadata detailing how the user's personal work document evolved the last time they visited the page. This feature was motivated by the observation that existing mechanisms (e.g., highlighting visited links) are often insufficient for directing users to resources that they previously found helpful for accomplishing a task.

Each of InterTwine's aforementioned features represents an exploration of a broader concept I refer to as *interapplication information scent*. Information foraging theory posits that people make use of information scent to guide their selection and use of information resources within "patches," or collections, of information [21]. When using the web, information scent is provided by elements such as a search engine's autocomplete service, the short page snippets shown in search results, or previously visited links rendered in a different color. In desktop software, menu hierarchies, command names, tool icons, and tooltips all provide feedforward mechanisms that can be considered forms of information scent that assist users in finding relevant functionality. While these separate systems each provide useful forms of information scent to guide the pursuit of desired information, they largely function independently of one another: the activities in one environment have no effect on the presentation of information in the next, forcing the user to manually link information patches as they move through each step across the web-mediated execution bridge. InterTwine links the separate information patches by arranging for the browser and the application to maintain, and to regularly query, a shared log of user interactions and events. In essence, browsing histories and document editing histories, are synchronized and merged.

In the context of the thesis statement of this dissertation, InterTwine observes actions across all three steps of the web-mediated bridge, and leverages these observations in an



Here is an example image, loaded into GIMP. I thought it might look nice as a black and white image.

Via Grayscale

Here is what I get if I use the standard mode change to grayscale from RGB.

Duplicate the original image (**`Ctrl+D`**) and right-click on the copy. Select **<Image> Image -> Mode -> Grayscale.** I don't know how this conversion works in GIMP, but I have read that Photoshop uses a standard mix of the RGB channels for their grayscale conversion: RED=30%, GREEN=59% and BLUE=11%. Supposedly this mix accounts for the eye's sensitivity to different colors. This formula does a pretty nice job in the general case, but some images do not work as well with it, particularly if the green channel component is not strong.

I suspect GIMP uses a similar formula. My experiments with the Channel Mixer (more on this below) support this.

Via Desaturate

Here is what I get if I use desaturate instead. Duplicate the original image (**`Ctrl+D`**) and right-click on the copy. Select **<Image> Image -> Colors -> Desaturate.** Unlike the grayscale mode change above, the channels are not remixed in different percentages, so we should expect different results.

The result is visually different; note the increased contrast in the scales. Also, compare the 100% zoom views at right and in the previous grayscale example. You can see a lot more noise in the desaturated zoomed view (examine the blurred area below the spikes). The reason is that we are getting more blue and red channel noise, whereas in the grayscale mode change operation the algorithm is giving us a remix of 60% of the clean, detailed green channel.

Via Decompose RGB

Desaturate Tool: Turn colors into shades of gray

☆ Mentioned **4** time(s) on the current website (detailed below).

*"Desaturate Here is what I get if I use **desaturate** instead. Duplicate the original"*

*"grayscale conversion operation . the **desaturate** operation . decomposing to RGB"*

*"(more on this below) support this. Via **Desaturate** Here is what I get if I use"*

*"Select Image -> Colors -> **Desaturate** . Unlike the grayscale mode"*

Figure 1.5: One of InterTwine's features includes embellishing an application's menu items with beacons (star icons) when those items are mentioned on the web page a user is reading in their browser. Likewise, menu tooltips gain snippets describing the context in which each menu item is mentioned in the web page.

automated means to enable feedforward mechanisms and information scent cues designed to facilitate users in performing their work. Specific contributions claimed in this chapter include:

- a formative study that informed the need for, and design of, mechanisms that provide interapplication information scent
- defining the concepts of a shared interapplication history and interapplication information scent
- demonstration of three types of interapplication information scent: application bridges, history snippets, and history digests
- implementation of a system that ties together the separate information spaces of a web browser (i.e., Firefox) and a feature-rich application (i.e., GIMP)

1.3 Thesis Outline and Terminology

The remainder of this dissertation is structured as follows: The document begins by presenting related work, then details each of the three aforementioned projects in turn. As noted above, CUTS is presented in Chapter 3, Query-feature graphs are presented in Chapter 4, and InterTwine is presented in Chapter 5. The dissertation concludes with a general discussion of the challenges and limitations of the techniques proposed in this document, followed by an outline for future work. However, before continuing forward, it is worthwhile to step back, and more precisely define some terminology that is used throughout this dissertation.

***goal* (or *task*)** – In the context of interactive systems, Donald Norman defines a *goal* as "*the state (of the system) the person wishes to achieve*" [96, p. 37]. A goal becomes a *task* when users contemplate, or begin taking actions, in service of achieving a goal. In this dissertation, the distinction between tasks and goals is rarely meaningful, and the terms are used interchangeably.

online resource – The term *online resource* broadly refers to any *primarily text-based* web page that a user may consult for the purpose of achieving a goal or performing a task in software. Such resources include: user manuals, online tutorials, answers posted to question answer websites, forum posting, and other materials of a similar instructional nature. For the purposes of this dissertation, I exclude online instructional videos – though video tutorials are discussed in the discussion and future-work chapters (Chapters 6 and 7, respectively).

***feature* (of software)** – The term *software feature* is used to refer generally to elements or functionality of a software application with which users may interact. This includes, but is not necessarily limited to: tools, commands, settings, dialogs, and other interactive elements.

feature-rich application – The term *feature-rich application* refers to any software application or interface whose features, as defined above, number in the hundreds or thousands. Adobe Photoshop, Microsoft Word, GIMP, and AutoCAD are all examples of feature-rich applications.

web-mediated (execution) bridge – The web-mediated bridge (Figure 1.1, foreground) refers to the process of leveraging web search and online resources in service of accomplishing a task in a feature-rich software application. The three steps across the web-mediated bridge include: (1) issuing a search query that describes a user's goal, (2) reviewing online resources and online instructional material detailing the sequence of steps needed to perform the task, and (3) carrying out the instructions by manipulating mechanisms in the system's user interface.

(raw) query log – A raw search query log, or simply a *query log*, refers to a chronological record of all search queries issued, by all users, to a web search engine.

At a minimum, each query log entry contains a timestamp, the text of the user's search query, and a unique identifier attributing the query to a distinct individual (or computer).

aggregate query log – The term *aggregate query log* refers to a type of query log that serves as a ledger, totalling the number of occurrences of each search query issued to the web search engine.

The aforementioned definitions and terminology are used throughout the remainder of this dissertation, which begins with a review of related work next in Chapter 2.

Chapter 2

Background

When people rely on web resources to support their work in feature-rich software, their interactions are distributed between the search engine, the retrieved documents, and the user interface of the feature-rich application itself. Together, these three environments create a rich research space affording many opportunities to study and effect how work gets done with feature-rich software. In this chapter, I review existing research that falls within or around this space.

The discussion of related work begins in section 2.1 with a review characterizing the extent to which people leverage online resources to support complex computer-mediated tasks. Mirroring chapter 3, section 2.2 reviews work that derives insights about people and tasks through static analysis of query logs, and other online artefacts (e.g., a corpus of customer reviews). Mirroring chapters 4 and 5, section 2.3 reviews existing research systems that form connections between environments, leveraging online actions and materials to improve interactions with the software. The chapter concludes with a discussion of the set of open problems addressed in this dissertation.

2.1 Searching and re-finding

The introduction of this document made an important claim: Users of feature-rich software often adapt to ubiquitous access to web search by adopting a strategy of learning where to find procedural information rather than learning the specific operational details necessary to directly perform their tasks. In other words, users learn to retrace their paths across the web-mediated execution bridge. This section begins by reviewing research that directly argues for this main effect, then discusses work demonstrating how this behaviour is manifest in various computer-mediated tasks including systems administration, software development, and work with a feature-rich raster graphics application. The section concludes by discussing some strategies people employ to re-access previously visited resources – a task that becomes necessary when users learn to rely on these external resources.

2.1.1 Search as external memory

In a 2011 paper published in the journal Science [116], Sparrow et al. performed a series of experiments investigating how people adapt in response to having convenient and ubiquitous access to information (i.e., a reliable Internet connection and web search). In a series of experiments, the authors found that: (1) when faced with a gap in knowledge, people are primed to think about web search; (2) when people believe that information will continue to be available in the future, they perform worse at recalling its details; and (3) in those same situations, participants perform better at recalling the locations where the necessary information can be accessed. From these findings, the authors conclude:

> *The Internet has become a primary form of external or transactive memory, where information is stored collectively outside ourselves. [116]*

Here, transactive memory refers to the phenomena where individuals in a group (e.g., a family or a group of co-workers) leverage one another as external memory stores or memory aids, thus serving to benefit from each other's individual expertise [125]. In these environments, people quickly learn *who knows what*, and to which information one must personally attend. Sparrow argues that transactive memory provides the theoretical framework for characterizing the processes that underly people's increasing reliance on web search and online resources:

> *Relying on our computers and the information stored on the Internet for memory depends on several of the same transactive memory processes that underlie social information-sharing in general. (...) Just as we learn through transactive memory who knows what in our families and offices, we are learning what the computer "knows". (...) We have become dependent on them to the same degree we are dependent on all the knowledge we gain from our friends and co-workers. [116]*

Given this general effect, the next section characterizes how dependence on online resources is manifest in various computer-mediated tasks.

2.1.2 Search for supporting programming, and systems administration tasks

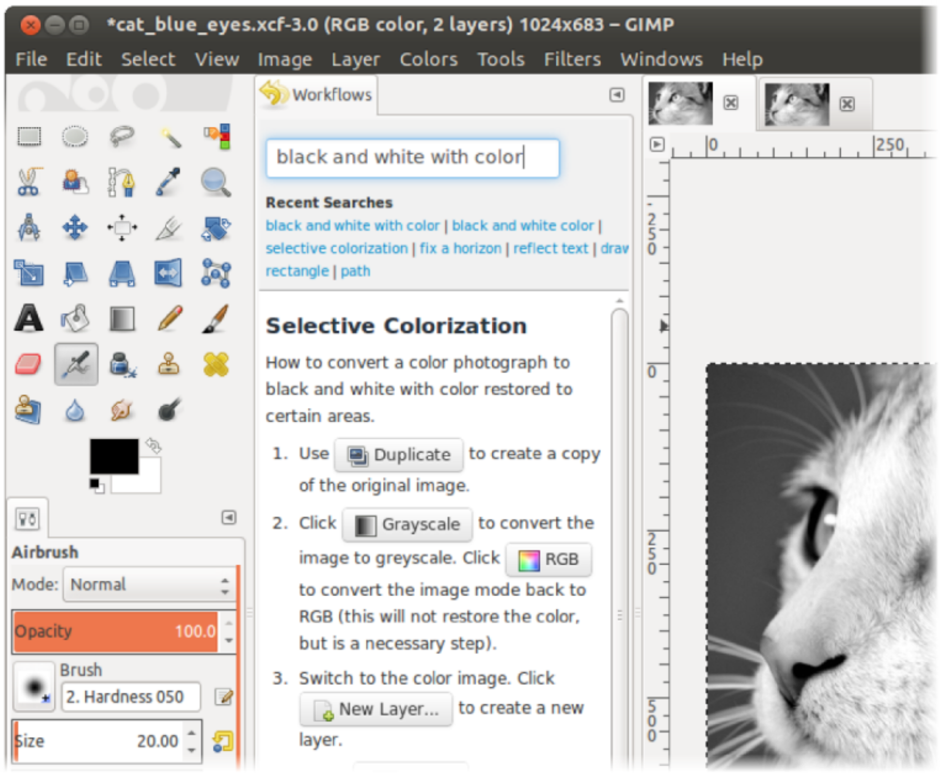
Recent work in HCI has explored the extent to which knowledge workers leverage online resources to support their work practices. In 2004, Barret et al. performed a series of observational field studies of system administrators in their places of work [10]. The researchers reported that general-purpose web search was the primary tool used by administrators on a day-to-day basis.

Software developers also make extensive use of online resources in their day-to-day activities. For example, Goldman and Miller studied patterns of web use by programmers [58], and found that 23% of websites visited while users were authoring code were directly related to the software project they were developing. Likewise, when Brandt et al. [16] tasked experienced student-programmers with developing a web chat application, the researchers noted that participants spent 19% of their time conducting online research. In Brandt's study, use of online material was frequent and brief, with half of all web sessions lasting 47 seconds or less. Notably, many participants explicitly reported using search as an alternative to memorizing programming details, with one participant noting that they had retrieved the same database connection code hundreds of times previously for past projects. This echoes the aforementioned findings of Sparrow's study on web search and external memory, and foreshadows comments made by participants who took part in the iterative design and evaluation of the InterTwine system, as described in Chapter 5.

Building on the work of Brandt et al., Meredith Ringel Morris and I examined the role of web search in answering programming questions on popular programming question-answer websites such as Stack Overflow and MSDN Forums [50]. In this work, we examined the web browsing logs of 120 users who had recently answered Stack Overflow questions, and found that 56% of these users performed relevant online research prior to posting their answers. This proportion was replicated in a second study, which solicited survey responses from users of MSDN Forums who had recently answered questions on this website. Again, 52% of the 107 developers surveyed reported that they had relied on web resources when composing their solutions. Additionally, in 70% of uses, respondents reported that they employed web search to retrieve material they already hand in mind and had previously relied upon. These results suggest that professional software developers often rely on the ability to re-find relevant online documentation rather than on committing technical details to memory.

2.1.3 Search for supporting use of feature-rich applications

The work outlined in the previous section reflects strategies adopted by professional software developers and systems administrators. This thesis considers these web-centric strategies in the more general context of supporting use of *any* feature-rich software. Here, too, research suggests that users often adapt a strategy that relies on finding and re-finding helpful web resources. In 2013, Ben Lafreniere et al. conducted a laboratory study evaluating the effectiveness of *Workflows* [77, 79], a task-centric interface to the GIMP image manipulation software. A key component of their system is that the task-specific adaptations enabled by *Workflows* resemble tutorials, and are indexed and retrieved through a built-in search field displayed prominently in the interface (Figure 2.1). The researchers compared their system to that of an unmodified installation of GIMP, paired with a web browser open to the Google search engine. Sixteen participants were asked to perform tasks with each system, and then to return two weeks later


*cat_blue_eyes.xcf-3.0 (RGB color, 2 layers) 1024x683 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Workflows
black and white with color
Recent Searches
black and white with color | black and white color | selective colorization | fix a horizon | reflect text | draw rectangle | path
Airbrush
Mode: Normal
Opacity 100.0
Brush
2. Hardness 050
Size 20.00

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.

Figure 2.1: A screenshot of Lafreniere et al.'s *Workflows* system, a task-centric interface to the GIMP image manipulation software. In *Workflows*, task-specific adaptations are retrieved using a built-in search interface. (Reproduced with permission from [77])

to repeat the experiment. Relevant to this dissertation, the researcher's observed *keyword learning* in both experiment conditions. Specifically, participants were significantly faster at retrieving relevant resources in the second experiment session, with five participants explicitly attributing task successes to their abilities to recall keywords from the first session performed two weeks earlier. The authors hypothesize that keywords are more memorable than low-level operational procedures because keywords bear a closer resemblance to how one describes their task or goal in their own words. Consequently, the authors propose that keyword learning is better able to scale as applications become more complex.

In summary, work by Sparrow et al. strongly suggests that, when people have ubiquitous access to the Internet, they adapt a strategy of recalling where information can be found rather than memorizing specific details of the information itself. This is consistent with observations of how developers, system administrators, and users of feature-rich creative software utilize web resources to manage task complexity. An important implication of this web-centric strategy is that people must revisit online information when faced with repeating a task in the future. I now review research examining how web content is revisited, and discuss the role of web search in this second-order process.

2.1.4 Re-visitation and re-finding

Developers and users of feature-rich applications leverage online resources to support their work. Often this requires them to revisit material that they previously found to be useful. This concept of information re-visitation has received considerable attention in previous research (e.g., see [2] for a good review). In particular, Obendorf et al. examined browsing logs for 25 users, and reported that *re-finding* was the most common strategy for returning to previously accessed information [98]. Re-finding is the act of using general-purpose web search to locate online information that has been previously visited. Notably, Obendorf reported that bookmarks, browsing history, and manual entry of URLs accounted for very few instances of re-access. Similar findings were reported by Aula et al., who examined the re-visitation strategies employed by expert web users [6]. It is therefore unsurprising that re-finding queries account for a large portion of web search traffic: Teevan et al. performed an analysis of one year of Yahoo!s search query logs, and found that 40% of all queries could be classified as attempts by users to re-find previously accessed information [119]. These results were corroborated and extended by Sanderson and Dumais, who examined a log of 3.3 million queries, and found that slightly more than 50% of all searches could be accounted for by users repeating previously issued queries [111]. In the context of this dissertation, this tendency towards re-finding, together with the strategic use of online resources as a form of external memory, suggest sustained use of the web-mediated execution bridge in supporting tasks performed with feature-rich software. The remainder of this chapter examines the implications of this sustained use of web search and online instructional materials.

2.2 Learning about People from Static Analysis of Web Search Logs and Web Documents

When people rely on web search and online resources to support their use of feature-rich software, it follows that their tasks are reflected in the web search queries they issue, and in the documents they retrieve. In this section, I review research that demonstrates the types of insights about people, tasks, and products that can be gained from static analysis of web query logs, and other online content (e.g., product reviews, bug reporting forums, etc.)

2.2.1 Using Query Logs to Learn about Real-World Phenomena

In [106], Matthew Richardson describes web query logs as resembling surveys "*sent to millions of people asking them to, every day, write down what they were interested in,* *thinking about, planning, and doing.*" This section reviews research that demonstrates how query log analysis can be used to model a wide range of real-world phenomena. Chapter 3 then extends these ideas to the field of human-computer interaction, demonstrating that aggregates of the logs of web search queries can be used to infer common tasks and breakdowns experienced by the users of any publicly available product.

Recent work has demonstrated the value of query logs for investigating matters of public health. For example, Ginsberg et al., demonstrated how query data can be used to estimate the prevalence of influenza in a population [56]. White et al. developed methods to detect adverse drug interactions by monitoring a population's medication and symptom-related search queries [129, 128]. West et al. [127] leveraged search data to investigate the association between dietary sodium, inferred from analyses of recipes retrieved by users, and the rates of hospital admissions for congestive heart failure. Paul et al. demonstrated that query logs can be used to model the typical progression of breast cancer [99] and prostate cancer [100]. Finally, working with Ryen White and Eric Horvitz, I demonstrated that it is possible to estimate population-level birth statistics by detecting queries issued by expectant mothers in search logs, then aligning their long-term query histories with the 40 gestational weeks of pregnancy [52].

In addition to epidemiology, web search logs have also been used to measure a number of economic metrics including: consumer confidence, claims of unemployment, automobile demand, vacation destinations, and inflation (See [23] for a good review). These uses of query data are examples of the more general problem of nowcasting. Nowcasting can be defined as predicting the present state of the world from indirect measures, in cases where direct methods of measurement exhibit long lag or delays. Query log analysis can also be used to predict the future behaviors of a population, as far as weeks in advance. For example, Goel et al. [57] demonstrated that query analysis can be used to predict: the opening weekend box office revenue of theatrical movies, the first month sales of video games, and a song's trajectory on the Billboard Hot 100 music chart.

To summarize, past work has demonstrated that query log analysis can be used to model a wide range of real-world phenomena. Chapter 3 extends this list of phenomena to include issues of concern for human-computer interaction research. In this latter case, search queries ultimately direct people to relevant web documents, and these documents can be studied to learn about users, tasks and breakdowns. The next section examines the types of usability information that can be mined from online documents.

2.2.2 Using Online Documents to Learn about Systems and Tasks

Thus far, the discussion has focused on learning about people through query log analysis. The documents that users ultimately retrieve provide another lens to learn about people and their tasks. This section outlines relevant research in this space, and describes the types of interactive system-related information that can be mined from these online resources.

2.2.3 Mining usability data from web pages

Since publishing my work on characterizing application usability through search (Chapter 3), a number of researchers have sought to gather similar information by analyzing other online datasets. In 2012, Andrew Ko developed *Frictionary*, a system which extracts and organizes issues reported on software support forums [75]. *Frictionary* parses forum postings using a probabilistic grammar, and then identifies common problems using template matching and normalization techniques similar to those outlined in Chapter 3. Once data has been mined, the system provides data visualizations and faceted browsing facilities that can be accessed by developers and designers. The system was populated using data for the Firefox web browser, and was presented to the Firefox support lead, and to Firefox's principal designer. Participants felt that the tool was impressive, and could help prioritize efforts, but suffered from imperfect topic extraction.

In 2013, Hedegaard and Grue Simonsen characterized the types of usability information contained in online reviews of software applications and video games [61]. The authors began by segmenting reviews into sentences, and then manually categorized each sentence according to the types of usability information contained therein. Here, the authors considered 4 competing sets of usability criteria, giving a total of 24 individual (and often over-lapping) dimensions of usability. The authors found that reviews do indeed contain considerable usability information, and that these data can be manually labeled with a high degree of agreement between raters. This motivated an automated approach. To automate the labeling process, the authors trained a SVM classifier for each of the 24 dimensions, using a one vs. rest strategy. The effectiveness of the classifiers varied from dimension to dimension, but 14 of the classifiers achieved F1 scores of 0.50 or higher. The authors concluded that their technique could determine if users were "preoccupied" with certain usability dimensions (e.g., learnability), but that the SVM classifiers rendered it difficult to determine which specific features of the product itself were relevant to a given dimension.

More generally, there has been considerable work in the area of opinion mining and sentiment analysis in the context of online reviews (e.g., [64]). Indeed, several survey papers now describe this research space [88, 118, 123]. As with the work by Hedegaard et al., past re-search in this space often encountered challenges when trying to identify the product features, or specific reasons, that give rise to a positive or negative review. To address this problem directly, Kim and Hovy, developed a maximum entropy classifier to extract the detailed pros and cons mentioned in online product and restaurant reviews [74]. Likewise, Yatani et al. [131, 132] preformed sentiment analysis on adjective-noun pairs extracted from online restaurant reviews, allowing their *Review Spotlight* tool to report the positive and negative characteristics of a venue (e.g., "best sushi", "long wait", etc.). In an evaluation, the researcher found that the data and visuals of Review Spotlight allowed users to quickly form detailed opinions of restaurants, enabling them to make restaurant selections much more quickly than when using a more traditional set of patron reviews.

To summarize, past work has demonstrated that online documents, such as postings to support forums and product review websites, do contain information about the usability of interactive systems, and in some cases these data can be extracted through automated means. Identifying the product features that give rise to a positive or negative sentiments can be a challenge, though specific phrase templates (e.g., adjective-noun pairs) can be of value. In Chapter 3, similar phrase-based templates are used to categorize product-related search queries, with the goal of identifying searches that are related to troubleshooting problems, or retrieving instructions or tutorials. The types of task-related information that can be extracted from online tutorials is reviewed in the section that follows.

2.2.4 Mining task details from tutorials

The previous section outlined the types of data that can be mined from online reviews, support forums, and other dynamic web resources. There has also been considerable interest in examining software tutorials, manuals, and static reference material. Here, the primary interest is in extracting the procedural details described therein. Such procedural details have numerous applications including task modeling [17], software automation [14, 84, 101], machine-guided help [85], and interface search (chapters 4 and 5).

When mining procedural information from tutorials, a natural first step is to extract the names of all tools, commands, menu items or interface widgets mentioned therein. In this document, I refer to this problem as named *widget recognition* – a domain-specific instance of the more general named entity recognition problem of natural language processing. Numerous research papers [84, 85, 101], including the work presented in appendix A of this dissertation, demonstrate that such elements can be reliably extracted from web documents. Here, my co-authors and I demonstrated that these entities can be detected with an accuracy of 95% using a naive Bayes classifier, with carefully chosen features that reflect common conventions employed by tutorial authors (e.g., using the sequence `->` to denote a menu item, as in "`File -> Save`"). More recently, researchers have achieved accuracies as high as 97% using conditional random fields [84, 101]. Unfortunately, the parameters of these operations are much harder to extract, with past work achieving F1 scores as low as 0.36 [84]. For example, a recognizer might detect mentions of a system's "*Gaussian Blur*" tool, but is unlikely to be able to correctly associate the operation with any mentions of the blur *Radius*, a key parameter which may or may not be mentioned in a tutorial's text.

Beyond simply extracting commands and parameters from text, there has been some research seeking to recover higher-level task details from written material. In 2002, Brasser and Linden strived to automatically extract full task models from written scenarios [17]. The authors manually crafted a natural language grammar, which was implemented as a 25-state augmented transition network. Unfortunately, the hand-built grammar did not perform particularly well, achieving an accuracy of 48% for detecting entities mentioned in text. More recently, Branavan et al. [14] demonstrated the potential for reinforcement learning approaches for interpreting natural language instructions.

In contrast to previous work using supervised learning, Branavan et al.'s system learned how to interpret instructions by repeatedly testing hypotheses within a virtual machine. This approach has the advantage of being able to interpret some high-level instructions that lack details of the specific low-level operations needed to perform the action in the interface. The authors reported that their method was able to correctly interpret 62% of the high-level actions in their dataset.

Despite steady progress, numerous challenges remain, preventing complete and accurate machine interpretation of written instructions. Consider for example, the tutorial excerpt:

Place it (an object) underneath the original text, as if it were a reflection [40]

The correct interpretation of this instruction requires: (a) coreference resolution, to determine to which object the pronoun "it" refers; (b) spatial reasoning, to determine approximately where the item is to be placed; and (c) an understanding of the purpose clause "as if it were a reflection" to further constrain the final placement. In this case, three challenges arise from a single tutorial sentence. When examining full tutorials, these and other challenges quickly accumulate. To this end, my work in [51] presents a roadmap for the research challenges that must be tackled for more complete machine understanding of instructional materials for interactive systems.

In summary, past work detailed in sections 2.2.3 and 2.2.4 have sought to leverage online documents to learn about products and interactive systems. Online support requests and reviews are an obvious first choice for analysis, and past work has achieved limited success in extracting usability data from these sources. Web tutorials are another common target for analysis, with many researchers seeking to extract procedural information from these documents. In both cases, automated efforts are stymied by the imprecise nature of written language, with work in this space progressing in lockstep with advancements in more general natural language processing.

2.3 Interactions and Interventions

By recognizing that people rely on search and on web resources to support their use of technology, human-computer interaction researchers can learn a great deal about users, tasks and breakdowns; but, they can also develop new interactions and interventions that optimize the use of online materials. The remainder of this chapter reviews research investigating how search and online materials can be integrated into client software and vice versa. Chapters 4 and 5 of this dissertation develop these ideas further.

2.3.1 Integrating (local) search into feature-rich software

Local search is becoming a popular method of finding documents, settings, and other functionality, in modern feature-rich applications. For example, in all modern operating systems, users are able to enter a few keywords into a search field to launch applications and load documents. In particular, the Mac OS X operating system offers a search field in the system-wide help menu that provides users with a means of searching and executing commands found in the top few levels of any application's menuing system. A similar system is now available in the Chrome web browser, the Ubuntu Unity heads-up display (HUD), and in the Windows operating system. Similarly, programs like Quicksilver [115], and Ubiquity [76] allow keyword search to be used to both issue commands and to specify command parameters in a range of applications and usage scenarios.

Notably, in many of the aforementioned examples, the search services are locally hosted on the user's machine. This limits the extent to which user behavioural data [3, 12] can be used to refine and improve search results, and limits opportunities for performing the types of log analyses described in 2.2.1 as well as in chapters 3 and 4. These limitations are dissipating: In December 2014, the online version of Microsoft Word [54] incorporated a search bar which, as with previous systems, searches over the elements of the application's user interface. However, Word Online is distinguished from past systems by its use of the centrally-hosted Bing.com search engine to derive search results. In this dissertation's discussion (Chapter 6), I elaborate on the potential of these types of web-scale integrations of search technologies and interactive systems.

2.3.2 Integrating web resources into feature-rich software

In addition to search, researchers have begun to explore how best to incorporate more general online resources into feature-rich software systems. Several research projects [60, 58, 15] have sought to directly support programmers' tendencies to refer to online material while writing code. For example, HyperSource [60] helps developers document and track the origin of the code they copy and paste from websites, by automatically embedding referenced URLs directly into the project code. Codetrail [58] is similar to HyperSource, but associates websites with project code through purely automated means – it compares a developer's recently written code to pages in their web browsing history, forming associations when similarities are identified. Blueprint [15] is a system that takes these ideas one step further by integrating web search directly into a software development environment (IDE). Here, web searches can be executed within the source code editor using an interaction technique similar to the code-completion services of modern IDEs. Blueprint facilitates the process of adapting retrieved code listings to a user's project, and maintains the association between these two entities. Finally, my past work with CiteHistory [50], also enhances a developer's ability to leverage their search and browser history, albeit in service of the specific task of adding citations to Q&A forum posts.

Moving beyond tools for software developers, the AdaptableGIMP and Workflows projects [78, 77, 79] demonstrate how web resources can be leveraged within a feature-rich client application. The AdaptableGIMP [78] comprises two components: (1) an online wiki where users can create task-specific tool pallets (a.k.a., "task sets") for the GIMP image manipulation program, and (2) a version of the GIMP client application that can load these customizations, together with their task-specific tips and instructions. The Workflows project is an evolution of AdapableGIMP. Workflows organizes the task-specific tools into small instructional documents that resemble tutorials, which then are embedded directly in the system's user interface. In both AdapatableGIMP and Workflows, users retrieve task-specific customizations by querying a built-in search service. The Workflows system was extensively evaluated, and Lafreniere et al. reported that incorporating workflows into GIMP's user interface substantially improved user performance compared to a classical web browser / web tutorial baseline. As such, AdaptableGIMP, and in particular Workflows, demonstrates one vision of how search, and instructional materials can be integrated into an application to support users in their daily tasks.

2.3.3 Integrating application context and functionality into web resources

The previous section described how web resources can be incorporated into desktop applications. It is also possible to do the reverse: to move application context and components into web services. For example, Michael Ekstrand et al. [42] explored how web search engines might leverage contextual cues provided by client software in order to provide more relevant search results. The authors modified a vector graphics drawing application (Inkscape) to communicate with a web search engine. The search engine was then able to incorporate various details of the user's session into its rankings (e.g., user queries are expanded to include mentions of the last 5 Inkscape commands issued, and the type of Inkscape object currently selected). Here, search results were generated using an ensemble method, which involved issuing a large number of queries to the Google search engine. Unfortunately, the ensemble approach failed to significantly outperform the unmodified version of Google in all test scenarios explored by the researchers.

While Ekstrand's work demonstrates how application context can be leveraged in an online environment, other work has examined how application functionality can be embedded in an online environment. For instance, Laput et al. [84] developed tutorial-based applications, which are specially crafted web documents that directly control feature-rich software applications (e.g., Photoshop). Here the concept is to use the tutorial as an alternative interface to the software application, allowing users to issue commands, modify parameters, and review results, all within the context of a web document. In this system, the feature-rich application is hosted in a remote virtual machine, and is (in many cases), invisible to the user. While these tutorials need to be specially authored to enable interactivity, the authors present tools to semi-automate the process of adapting existing online materials to versions supported by their system.

Finally, Lafreniere et al.'s community enhanced tutorials take this concept one step further, by directly embedding a full photo manipulation application directly into web tutorials. Here, the photo editor's entire UI appears alongside the tutorial material [81]. The colocation of the application with the tutorial allows the system to fully record a reader's actions across both, in a manner comparable to InterTwine's shared interaction history (described in Chapter 5). In particular, the system detects and logs any deviations between the user's actions and the steps described in the original tutorial instructions. These variations can then be echoed back to the community as alternative methods for completing each tutorial step. The authors conducted a user study of this system, and found that this setup was especially beneficial in cases where the original tutorial was of low quality, or when a user's chosen work document differed significantly from that of the original tutorial instance. This work clearly demonstrates the value that arises when a system is aware of a user's actions in both the web and application contexts.

2.4 Summary

In this chapter I detailed related research which argues that people – even experts – leverage online resources to support their computer-mediated tasks. This strategic use of search serves as a mechanism to cope with the ever-increasing complexity of our interactive systems. As a consequence of this behaviour, the query logs of major search providers serve as centralized repositories cataloguing the day-to-day tasks and interests of a user population. While query logs have demonstrable ecological validity and utility in characterizing, monitoring and predicting many real-world phenomena, to the best of my knowledge they have not been directly applied to characterizing how people interact with feature-rich software. To this end, Chapter 3 details how analysis of aggregate query log data can be used to characterize the day-to-day tasks and needs of a system's user community. Importantly, this in-situ data is vast in scale and in scope, and is arguably less prone to self-selection bias compared to other data sources.

Past work has also attempted to garner similar insights through analysis of written documents such as forum postings, product reviews, and instructional material – though these efforts have been stymied by imperfect topic extraction, and other challenges arising from the need to do extensive natural language processing on these longer documents. The systems described in Chapters 4 and 5 sidestep this problem by leveraging behaviour data to contextualize the contents of these written documents. For example, the QF-Graphs described in Chapters 4 derive value from written tutorials by pairing them with the search queries that a user might issue to access those documents. Likewise, Chapter 5 pairs tutorials with the commands issued while accessing the documents on previous occasions.

Finally, this chapter reviewed other work which has combined or integrated web search or online documents with feature-rich software. Here, two general strategies have been employed: In the first, feature rich software is updated to leverage online material. In the second, online material is updated to leverage context and functionality provided by feature-rich software. The final project described in this dissertation, InterTwine, demonstrates a system where such enhancements are bidirectional – where online resources influence the software's user interface, *and vice versa.* InterTwine is presented in detail in Chapter 5.

The next chapter begins the journey across the web-mediated execution bridge by exploring the types of insights about users and tasks that can be derived from logs of web search queries.

Chapter 3

CUTS: Characterizing Usability Through Search

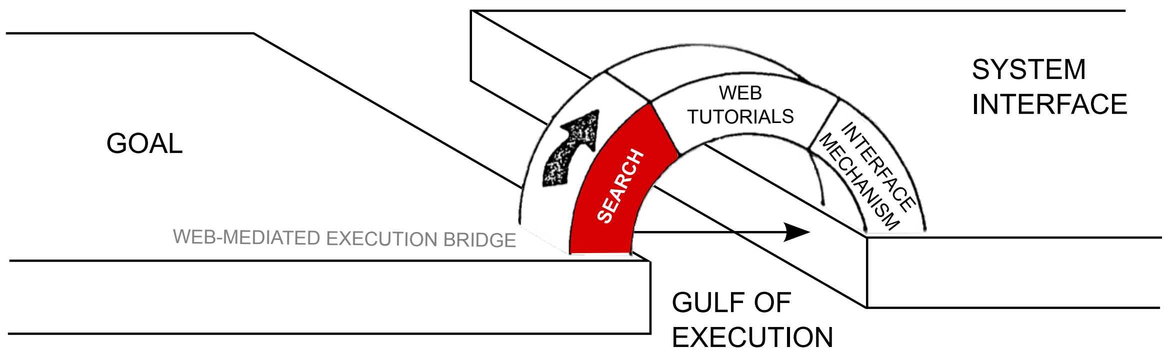


Figure 3.1: In this chapter I investigate the first step across the Gulf of Execution via the web-mediated execution bridge: i.e., Search. I demonstrate that aggregate logs of web search queries can be leveraged to identify common tasks and potential usability problems faced by the users of any publicly available interactive system. To assist with the collection and analyses of these data, I introduce CUTS (Characterizing Usability through Search), an automated system for gathering, labelling and filtering queries for the purpose of identifying common tasks and issues encountered by a system's users. Portions of this chapter were first published in [48].

When users leverage online resources to bridge the Gulf of Execution, their journeys begin with the formation and issuance of web search queries. Given this behavior, search engine query logs serve as centralized repositories cataloguing the day-to-day tasks and needs of the user base of any publicly available interactive system. This chapter describes a method for approximating web search query logs, with the intention of enumerating common tasks, and identifying potential usability problems in these feature-rich applications. Here the goal is to transform such data into forms that usefully complement and

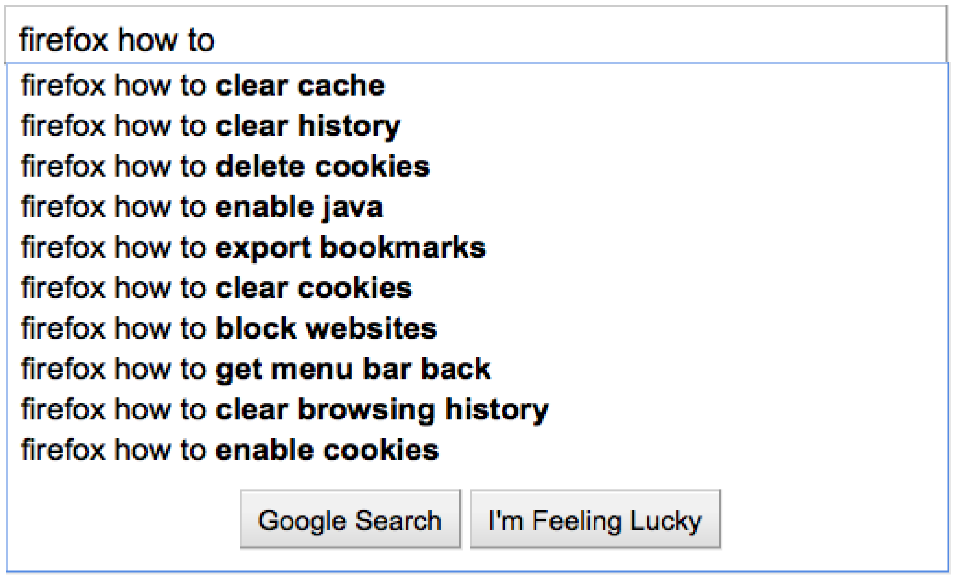

firefox how to
firefox how to **clear cache**
firefox how to **clear history**
firefox how to **delete cookies**
firefox how to **enable java**
firefox how to **export bookmarks**
firefox how to **clear cookies**
firefox how to **block websites**
firefox how to **get menu bar back**
firefox how to **clear browsing history**
firefox how to **enable cookies**

Figure 3.2: The top 10 suggestions provided by Google Suggest for the phrase "firefox how to". (Retrieved September 2010)

augment data collected via traditional usability methods (e.g., cognitive walkthroughs [103], or heuristic evaluations [94]).

An example serves to illustrate the general strategy taken in this chapter. Consider Google Suggest, the query auto-completion service that provides query completion suggestions for a given input. In 2010, given the phrase "firefox how to", Google Suggest produced the list of 10 suggested completions depicted in Figure 3.2. As will be shown later, these suggestions represent an N-best list for predicting the most likely completion of the query. Here, query popularity is a key feature for constructing such lists.

From the list of top 10 Firefox "how to" suggestions (Figure 3.2), it is immediately clear that users have a number of privacy and security concerns, as evidenced by their desire to clear their cache, history, and cookies. However, the eighth item ("firefox how to get menu bar back") is particularly interesting. An inspection of the Firefox user interface (version 3.6 on Windows), reveals that the top-level menu bar is easily hidden by deactivating the "Menu bar" item in Firefox's "View → Toolbars" sub-menu (Figure 3.3, left). However, once this action is taken, it is not easily reversed: The top-level menuing system is now hidden, removing the very means the user would employ to attempt to re-instate this important UI widget (Figure 3.3, right). What is noteworthy about this example is that it quickly moved from data derived from query logs to a testable hypothesis regarding the usability of the software.

The contributions in this chapter lie in expanding this manual process to the automated one shown in Figure 3.4. This automated process is referred to as *CUTS* (characterizing usability through search). CUTS automates the harvesting, ordering, labeling,

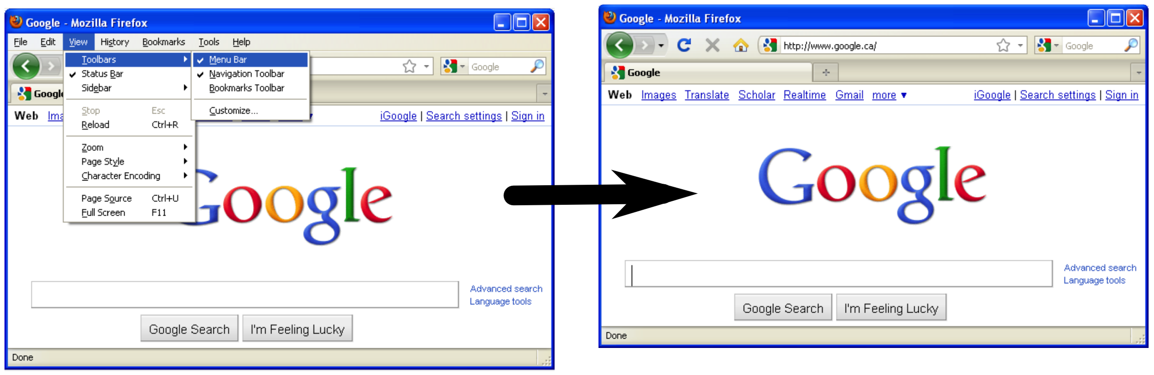


Figure 3.3: In Firefox 3.6 for Windows, the application's main menu can be hidden by disabling the `View -> Toolbars -> Menu` toggle option. Once this action is taken, it is not easily reversed – users must know to use the keyboard shortcut `Alt+V` to reaccess this menu option.

filtering, and grouping of search queries, with the ultimate goal of allowing its operators to better understand the common tasks and needs of a user base. Importantly, like raw query logs, the data produced by CUTS are timely, have a high degree of ecological validity, and are arguably much less prone to self-selection bias than traditional means of collecting data from users.

While seemingly straightforward, automating this process requires overcoming a number of challenges: Raw query logs are not made publicly available; there is a need to automatically determine query intent for the purposes of labeling and filtering queries (for example, to distinguish troubleshooting queries from those seeking to download the application); and differently phrased queries on the same topic should be reduced to a common canonical form. The specific contributions of this chapter, outlined below, address these challenges.

To address the problems of obtaining and ranking search queries, this chapter demonstrates how publicly available query suggestion services (e.g., Google Suggest) and web-based tools for advertisers can be employed to create reasonable approximations of raw query logs.

This chapter also introduces two new query taxonomies to address the need to label and filter queries. The first extends previous search query taxonomies to include categories relevant to interactive systems. For example, this new taxonomy differentiates between queries issued to troubleshoot a problem and those seeking a tutorial. The second taxonomy considers *how* a query is phrased. As will be shown, how a query is phrased closely corresponds to the categories of the specialized taxonomy. CUTS exploits the relationship between these two classification schemes to ascribe query intent from query phrasing.

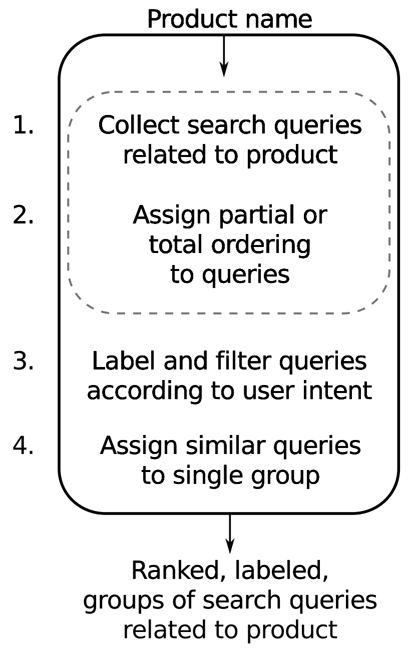

Product name
1.
Collect search queries related to product
2.
Assign partial or total ordering to queries
3.
Label and filter queries according to user intent
4.
Assign similar queries to single group
Ranked, labeled, groups of search queries related to product

Figure 3.4: An overview of CUTS. Steps 1-2 are easily performed with access to raw query logs, but otherwise require approximation techniques. Step 3 utilizes CUTS' domain-specific query taxonomy specialized for interactive systems.

Finally, common questions or issues are often expressed using a number of different query phrasings. To cope with this variability, this chapter introduces a transformation that enables minor differences between queries to be ignored.

The rest of this chapter is structured as follows. I first describe a method for harvesting and ranking search queries using publicly available services. I then introduce two complementary query taxonomies, and present heuristics that can be used to label and filter search queries. The final step of the process, grouping queries, is discussed, and a set of strategies are introduced to assist with this process. I then present a series of examples illustrating the overall utility of this approach. This chapter concludes with a discussion of the limitations of the technique.

3.1 Query harvesting

The core of the CUTS process lies in the analysis of search query logs. Unfortunately, actual search query logs are almost never released publicly by search providers. When access to raw query logs is not possible, search queries can be harvested using publicly accessible interfaces: Modern search engines provide indirect and privacy-preserving access to their logs through their query completion suggestion services [9]. This section describes a process for systematically harvesting queries related to a particular interactive system using these services. This section also provides evidence that the results of this method can be considered a representative sampling of the raw query logs.

3.1.1 Harvesting from auto-completion services

Query completion suggestion services operate as if backed by a prefix tree [9]. When viewed in this way, the characters making up a partially entered query define a path through the tree starting at the root, passing through numerous nodes. Each node contains a listing of popular queries whose prefix matches the path taken thus far (Figure 3.5). Query completion services follow the paths prescribed by partially entered queries, and return the suggestions listed at the ends of these paths.

Given the tree-like structure of these services, a standard depth-first or breadth-first tree traversal can be performed by expanding partial queries one character at a time, starting with the name of the system under investigation (Figure 3.6). A leaf (or external node) is reached when the completion service returns no suggestions for the given prefix.

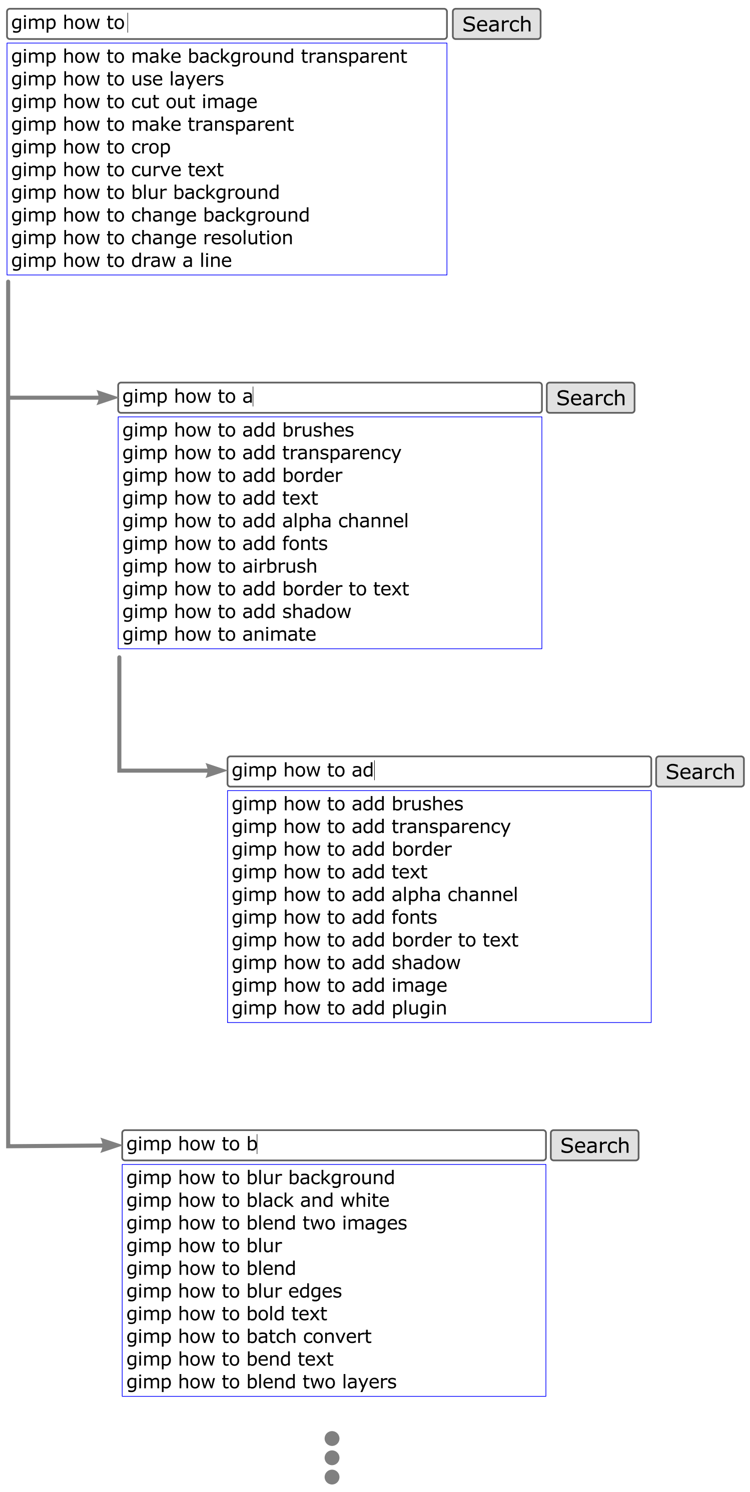
Mining additional queries

Some search providers, such as Google, vary their query suggestions depending on the position of the caret in the search query input box (Figure 3.7). More specifically, Google provides a list of the top 10 completions that either begin or end with the phrases on the left or right side of the cursor. Given this behaviour, the whole tree traversal procedure can be repeated to uncover query suggestions that end with a particular suffix, providing a more complete sampling of the query logs.

By executing a systematic search of the query completion tree, many queries can be collected for a given topic. For example, on June 19th, 2010, a systematic search recorded 74,795 unique queries incorporating the term "Firefox" in Google Suggest's query auto-completion database. Similar results were obtained for other systems for which data was collected (Table 3.1).

3.1.2 Representativeness and timeliness of auto-completions

In harvesting these queries, the working assumptions are that (1) query completion services are derived from the raw query logs, (2) a given query's prevalence in these logs will have some bearing on its ranking in the list of suggestions, and (3), the suggested completions are *timely*. "Timely" query completion services assign more weight to queries performed within a recent window of time. The following subsections briefly provide evidence that these assumptions are sufficiently valid for the intended purposes.


gimp how to

gimp how to make background transparent
gimp how to use layers
gimp how to cut out image
gimp how to make transparent
gimp how to crop
gimp how to curve text
gimp how to blur background
gimp how to change background
gimp how to change resolution
gimp how to draw a line

gimp how to a

gimp how to add brushes
gimp how to add transparency
gimp how to add border
gimp how to add text
gimp how to add alpha channel
gimp how to add fonts
gimp how to airbrush
gimp how to add border to text
gimp how to add shadow
gimp how to animate

gimp how to ad

gimp how to add brushes
gimp how to add transparency
gimp how to add border
gimp how to add text
gimp how to add alpha channel
gimp how to add fonts
gimp how to add border to text
gimp how to add shadow
gimp how to add image
gimp how to add plugin

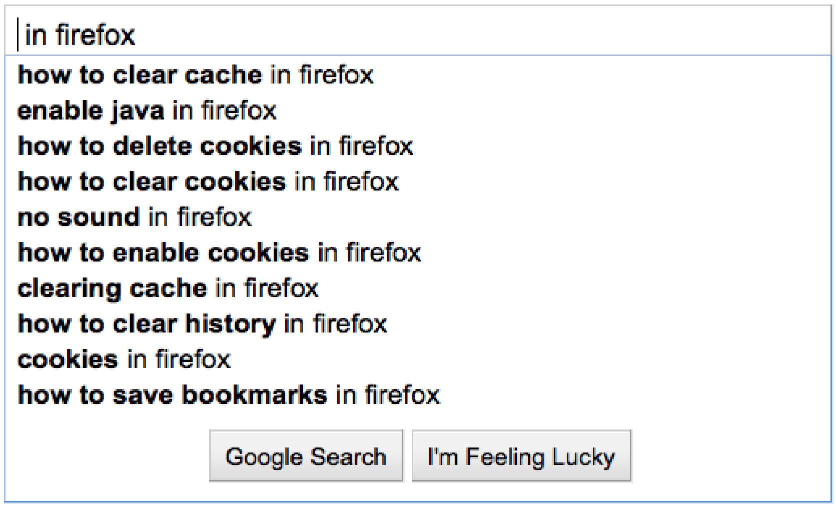
gimp how to b

gimp how to blur background
gimp how to black and white
gimp how to blend two images
gimp how to blur
gimp how to blend
gimp how to blur edges
gimp how to bold text
gimp how to batch convert
gimp how to bend text
gimp how to blend two layers

Figure 3.5: Query completion suggestion services operate as if backed by a prefix tree. A standard depth-first or breadth-first tree traversal can be performed by expanding partial queries one character at a time.

```
firefox,
firefox a, firefox aa, firefox aaa, ...
...
firefox z, firefox za, firefox zaa, ...
```

Figure 3.6: Input sequence representing a depth-first traversal of Google Suggest's prefix tree rooted at "firefox".



|in firefox

how to clear cache in firefox
enable java in firefox
how to delete cookies in firefox
how to clear cookies in firefox
no sound in firefox
how to enable cookies in firefox
clearing cache in firefox
how to clear history in firefox
cookies in firefox
how to save bookmarks in firefox

Figure 3.7: The *Google Suggest* auto-completion service varies its suggestions based on the caret position, enabling additional suggestions to be mined.

Representativeness of query completion suggestions

In the case of Google, some information about their query suggestion service has been published [30]. Specifically, Google's documentation notes that "*All of the queries shown in (Google) Suggest have been typed previously by other Google users*". Google also states:

> *Our algorithms use a wide range of information to predict the queries users are most likely to want to see. For example, Google Suggest uses data about the overall popularity of various searches to help rank the refinements it offers. [32]*

When mining data from Google Suggest, CUTS issues queries from a computer reserved for the purpose of the study (i.e., a system with no prior history of web search), and the system's cookies and browser state are cleared between each request. These steps are designed to bias the auto-completion service towards relying on search query popularity rather than on a user's query history, or on other forms of search personalization.

Application	Description	# of Query Suggestions
iPhone	A smartphone platform	476,462
Ubuntu	A Linux distribution	122,242
Photoshop	An image editor	119,791
Firefox	A web browser	74,795
Chrome	A web browser	45,499
Safari	A web browser	38,207
Kindle	An eBook reader	21,621
Gimp	An image editor	14,569
Nook	An eBook reader	8,985
Audacity	An audio editor	6,517
Kobo	An eBook reader	2,680
Inkscape	A vector graphics editor	2,501

Table 3.1: Number of unique query suggestions provided by Google for various interactive systems.

Timeliness of query suggestions

To identify trends and new issues as they arise, it is desirable that query suggestion services emphasize recent searches over those performed in the more distant past. To study the timeliness of Google's query suggestion service, I monitored the query completion suggestions for a range of products and software applications for a period of approximately three months (June 2010 through August 2010, inclusive). Suggestions were sampled on Monday, Wednesday, and Friday of each week during this timeframe. An analysis of the collected data reveals that Google updates its auto-completion database approximately once every 14 days. These results indicate that Google is actively maintaining its query suggestion database.

Knowing the frequency with which these services are updated is advantageous, but is not sufficient for determining the extent to which current search trends are represented in query suggestions. To investigate this question, one can examine when a noteworthy event begins to appear in query suggestions. A prime candidate for exploring this question is provided by the release of the iPhone 4 on June 24th, 2010. Almost immediately, there were media reports of significant signal degradation when the phone was held in a certain way [55]. The first evidence of this issue was spotted in the query suggestions on July 14th, 2010. On this date, the partial query "iphone d" resulted in Google suggesting [*iphone death grip*], while "iphone a" yielded [*iphone antenna*], and "iphone how to h" yielded [*iphone how to hold*]. None of these queries appear in the suggestions sampled on previous dates. This corresponds to a lag of about 20 days, suggesting that the query completion services place sufficient weight on recent queries.

3.2 Estimating query volumes

After harvesting queries, the next step is to estimate how frequently each is searched (i.e., its *search volume*). When queries are sampled from query suggestion services, detailed query volume information is not made available (current services do not indicate how often each search is performed). Missing data can be substituted in two ways. First, one can complement the data set with data collected from advertising and market research tools, such as the Google AdWords Keyword Tool [31]. Second, one can examine the structure of the synthesized prefix tree to obtain a *partial ordering* of the queries not covered by the advertising and market research tools. I describe each technique in turn.

3.2.1 Using marketing tools to gather ground-truth

Google provides a set of tools that can be directly applied to the problem of ranking queries. For this research, I leveraged the Google AdWords Keyword Tool [31], which is intended to help marketers valuate keywords for advertising purposes. Prior to September 2014, the AdWords Keyword Tool could be configured to report the average monthly search volume for any exact phrase [34], making it possible to directly gather the ground-truth search volumes of many query suggestions. Here, an exact phrase match occurred when the specified input (i.e., the query auto-completion suggestion) exactly matched search queries in Google's query logs, without any additional words before or after the input phrase.

The data collected for this dissertation were gathered in 2010 and adhere to the aforementioned exact match criteria. At the time of this writing (2015), the exact matching option has been replaced with *close variant matching* [34]. Close variant matching resembles exact matching, but allows terms to vary slightly from the input criteria. Such variations include: "*misspellings, singular and plural forms, acronyms, stemmings, abbreviations, and accents.*" [29] Should this research be replicated in today's environment, additional steps would be required to avoid double counting queries.

While many queries can be directly ranked using the Google AdWords tool, not all queries can be ranked in this way; in 2010, Google AdWords provided no data for queries whose monthly search volume was below a threshold of 12 queries per month. This threshold was reached well before the list of query suggestions was exhausted. For example, on June 19th, 2010, I harvested 74,795 unique query suggestions for the Firefox web browser. However, Google AdWords provided search volume data for only 15,057 of those queries. In short, the search volume of about 80% of the Firefox queries fell below the threshold reported by AdWords. Accordingly, one must employ another means of ranking the remaining queries, as described next.

3.2.2 Estimating search volumes for long tail queries

A means of estimating the query volumes of long tail queries (those not described in the AdWords data set) is desired. A considerable amount of prior work has investigated search query logs, and it has often been reported that search query volumes follow a heavy-tailed (Zipf-like) inverse-power law relationship [112, 69, 68, 43, 9] with the most popular search queries occurring exponentially more frequently than less popular queries. This relationship can be described mathematically as:

$$v = \alpha r^{-k} \tag{3.1}$$

where v is the search volume of a query q, r is the rank of the query q when queries are sorted in descending order by popularity, and both α and k are model parameters. This relationship can be expressed as a linear function in the log-log space, as follows:

$$log(v) = log(\alpha) - k * log(r) \tag{3.2}$$

The data returned from the Google AdWords tool is roughly consistent with this trend (Figure 3.8). Using the AdWords data for a given interactive system, one can recover the parameters $log(\alpha)$ and k using linear regression. In practice, query volumes often exhibit the *king effect* [82], where the first 2 or 3 queries appear as outliers to the general trend. As such, I use the Theil-Sen [108] estimator of robust linear regression to deal with these and other outliers.

Once the parameters are recovered, one can extrapolate the query volumes of long-tail queries, for which AdWords provides no data. To do this, one needs to approximately rank long-tail queries by popularity. While query suggestion services do not return the frequency with which each suggested query is performed, I have argued that they operate by returning the most popular queries for a given input. One can use this behaviour to derive a *partial ordering* of the query suggestions. The key insight is this: It is suspected that *the 10 query suggestions returned for a given prefix are more popular than all other queries later harvested that also begin with that same prefix.* An example illustrates this point:

> In 2010, the suggestion “firefox menu bar missing” appeared in Google’s top 10 suggestions for the prefix “firefox m”. Thus, one can infer that the “firefox menu bar missing” query is more popular than the 2362 other suggestions occurring in the data set that also share the prefix of “firefox m”. I write that this query has 2362 *subordinates* in order to convey this relationship, and rank long-tail queries in descending order by these values. This provides only a partial ordering because one can only perform comparisons of a node with its ancestors and descendants in the prefix tree. Nevertheless, a search volume-based ranking will be crudely approximated [9]. Once a ranking is estimated, equation 3.2 can be used to estimate average monthly search volume (Figure 3.8, green dots).

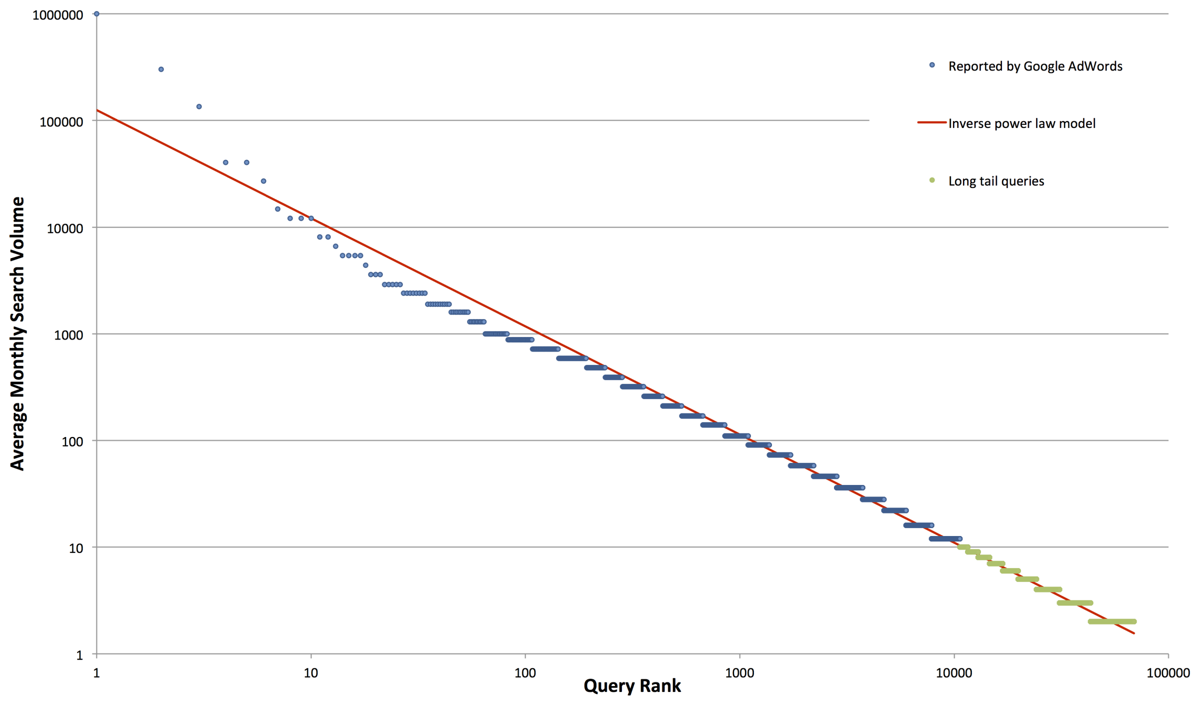


Figure 3.8: Search volumes for individual queries follows a heavy-tailed (Zipf-like) inverse-power law relationship. This relationship manifests as a straight line when both axes are represented on logarithmic scales. This figure presents data pertaining to queries about the Firefox web browser. Advertising data (blue) is used to learn the parameters of a line of best fit (red), which is then leveraged to extrapolate the search volumes of long tail queries (green).

These first two steps of harvesting and ranking queries provide a suitable, privacy-preserving, publicly accessible replacement for raw query logs. In the remainder of the chapter, the technique assumes only that one has access to a ranked list of search queries relating to the interactive system of interest.

3.3 Query classification and filtering

Given a list of queries, the next step is to automatically classify queries according to the likely intent of the individual performing the search (for example, to distinguish troubleshooting queries from those seeking to download the application). Once labeled, query logs can then be filtered to select entries that are potentially related to user tasks and usability issues.

Before queries can be automatically labeled and filtered, one must first understand the range of system-related queries that users submit to search engines. While previous work has developed a number of taxonomies for general classification of search queries (e.g., to distinguish between navigational and information-seeking queries) [18, 71, 107], I found these taxonomies too broad for the intended purposes. Instead, a classification scheme specialized for the domain of interactive systems is needed. Additionally, one needs to understand what features of a query can be used to support automatic labeling.

In this section, I address both of these needs: I introduce a domain-specific taxonomy of *query intent* specialized for interactive systems, and a second classification scheme that describes how a query is *phrased*. As I will show, in this domain, query phrasing is strongly related to query intent. Based on the aforementioned intent-phrasing relationship, I present a set of heuristic templates for automating the process of labeling and selecting queries of interest.

3.3.1 Domain-specific query intent taxonomy

Following the basic methods of grounded theory [117], I developed the query taxonomy by performing open coding on 200 randomly sampled queries regarding the use of software applications. From this initial coding, I identified a set of common, higher-level themes, which led to the taxonomy. The resultant taxonomy includes six separate classes of interactive system queries, synthesized from the perspective of query intent:

- **Operation Instruction**
  Would the query be used to find instructions for performing a specific operation or task?
- **Troubleshooting**
  Would the query be used for troubleshooting a bug or error condition?

- **Reference**
  Would the query be used to find non-task specific reference material? (e.g., a list of keyboard shortcuts)
- **Download**
  Would the query be used to acquire, download, or install something?
- **General Information**
  Would the query be used to find product reviews, comparisons, or other general information?
- **Off-topic**
  Is the query unrelated to the software / product?

3.3.2 Query phrasing classification scheme

In parallel with developing the former taxonomy, I also developed a classification scheme that describes how individual queries are phrased. The motivation for developing this scheme arose during the open coding sessions: For lengthier queries, it appeared that *how* a query was phrased was very much related to the *intent* of the user. As I will show, there is indeed a relationship.

Based on the open coding of the queries, the following high-level categories of query phrasing were identified:

- **Noun phrase** (e.g., gimp brushes)
- **Imperative statement** (e.g., gimp rotate text)
- **Question** (e.g., how to draw a line in gimp)
- **Statement of fact** (e.g., gimp won't start)
- **Present participle** (e.g., rotating text in gimp)
- **Other**

In the next section, I show that raters are able to achieve a high degree of inter-rater agreement when using the intent and phrasing taxonomies to label search queries. This agreement lends support to the overall utility of the taxonomies as instruments for labeling search queries.

3.3.3 Inter-rater reliability of the classification schemes

To establish the inter-rater reliability of these two classification schemes, two researchers applied both schemes to a set of 195 queries sampled from the GIMP and Firefox datasets. The GIMP and Firefox datasets were collected from Google Suggest on May 23rd, 2010 and June 19th, 2010 respectively. Selection of the 195 sample queries proceeded as follows: For each application, the top 50 queries (by search volume) were selected, followed by

Query Source	κ intent	κ phrasing
Firefox, top 50	0.74 (substantial)	0.80 (substantial)
Firefox, random 50	0.86 (near perfect)	0.80 (substantial)
GIMP, top 47	0.66 (substantial)	0.72 (substantial)
GIMP, random 48	0.66 (substantial)	0.81 (near perfect)

Table 3.2: Inter-rater reliability for each of the four sample sets.

an additional 50 randomly selected queries. The resulting set of 200 samples shared 5 queries in common with the set used for the initial open coding and were thus excluded from the validation process.

In labeling this data set, an overall inter-rater reliability rate of $\kappa_{\text{intent}} = 0.76$ was achieved for query intent, and $\kappa_{\text{phrasing}} = 0.79$ was achieved for query phrasing, using the Cohen's kappa measure of rater agreement. Inter-rater reliability across the 4 sources of queries is listed in Table 3.2. The observed agreements are considered to be substantial [83].

Before describing how the query phrasing classification scheme can be used to identify query intent, I first show how queries are distributed across these two classification schemes. These query distributions lend additional arguments for the overall utility of this approach.

3.3.4 Characterizing query data

The classifications of the 195 labeled queries are summarized in Table 3.3. The categories of interest for usability analysis coincide with the first two listed in the table and the taxonomy: "Operating Instruction", and "Troubleshooting". In the sample, about half of all query suggestions fall within categories that are of interest to HCI researchers and practitioners, demonstrating the overall richness of query logs when studying interactive systems.

3.3.5 Relationship between query phrasing and intent

If one compares how a query is labeled in each scheme, one finds that how a query is phrased can be highly indicative of a query's likely intent. These findings are summarized in Table 3.4. For example, in the sample set, if a query is phrased as an imperative statement, there is a 90% likelihood that the query is seeking operating instructions. A similar likelihood (87%) applies if the query is phrased as a question. Finally, if a query is phrased as a statement of fact, then it is almost certainly being used for troubleshooting. These relationships provide a set of strategies for automating the labeling of queries, which I describe next.

	Rater 1		Rater 2	
Query Intent	Freq.	%	Freq.	%
Opr. Instr.	84	43%	80	41%
Troubleshooting	15	8%	17	9%
Reference	21	11%	19	10%
Download	55	28%	62	32%
General	12	6%	12	6%
Off topic	8	4%	5	2%

Table 3.3: Frequencies of query intent labels for the 195 randomly selected GIMP and Firefox queries harvested from Google Suggest.

Query Intent	Noun	Imperative	Question	Fact	P. Participle	Other
Opr. Inst.	0.30 (38)	0.90 (28)	0.87 (13)		1.00 (5)	
Troubleshooting	0.05 (6)			1.00 (9)		
Reference	0.14 (18)	0.03 (1)	0.13 (2)			
Download	0.42 (53)	0.07 (2)				
General:	0.09 (12)					
Off topic						1.00 (8)

Table 3.4: Probability of query intent given its phrasing type based on the labels assigned by rater 1. Raw frequencies are listed in parentheses. Similar values are achieved using the labels assigned by rater 2.

3.3.6 Heuristics for automating query labeling

In the previous section, a relationship between query phrasing and intent was described by examining labels that were *manually* assigned to queries by a pair of human raters. Automating the CUTS process requires mechanization of the query labeling step. Through further inspection of the data, I have found that certain keywords or patterns are highly indicative of each of the different phrasing types. For example, queries containing the phrase "how to" indicate questions. Once a query's phrasing has been established, one can then infer its intent using Table 3.4. Here I focus on queries phrased as questions, imperative statements, and statements of fact because these phrasing types reliably indicate searches for instructions or troubleshooting information.

A partial list of phrasing patterns is presented in Table 3.5. These patterns were

Labels		Pattern	Example Query
Operating Instructions	Question	**how to** __ **in** *SystemName*	how to delete history in firefox
		SystemName **how to** __	firefox how to clear cache
		can *SystemName* __	can firefox block websites
		does *SystemName* __	does firefox have private browsing
	Imperative	**use** *SystemName* __	use firefox for windows update
		make *SystemName* __	make firefox default browser
		SystemName **set** __	firefox set default zoom
		create __ **in** *SystemName*	create a new profile in firefox
		SystemName **create** __	firefox create pdf
Troubleshooting	Fact	*SystemName* **is / isn't** __	firefox is starting slow
		SystemName **can / can't** __	firefox can't add bookmarks
		SystemName **will / won't** __	firefox won't open pdf
		SystemName **does / doesn't** __	firefox doesn't play sound
		SystemName **has / hasn't** __	firefox has no address bar

Table 3.5: Filtering templates for labeling the phrasing and likely intent of queries.

generated through a manual inspection of labeled data, and serve as basic heuristics for labeling different types of queries.

Many queries will not match any pattern, and will thus go unlabeled at this stage of processing. In the next section, I describe a technique for grouping related queries. When queries are grouped, labels for the individual queries are extended to the group, increasing the coverage of the labeling.

3.4 Grouping similar queries

The final step in CUTS is to reduce the variability with which queries are expressed in the data set. In query logs, common questions or issues are expressed using a number of different query phrasings. As an example, GIMP users may search "how to draw a circle in gimp", or they may simply type "gimp draw circle". Given this variability, it is desirable that similar queries be grouped, and their weights or rankings combined, in order to better estimate the prevalence of a given issue.

To group similar queries, one transforms queries to a canonical form where inconsequential differences are ignored (e.g., see Table 3.6). This transformation applies the

“firefox lost toolbar”	
lost my toolbar firefox	firefox toolbars lost
lost firefox toolbar	firefox lost my toolbar
lost all toolbars in firefox	firefox lost all toolbars
lost toolbar in firefox	firefox lost toolbar
lost my toolbar in firefox	lost my firefox toolbar
firefox toolbar lost	

Table 3.6: 11 distinct queries which share the canonical representation “firefox lost toolbar”.

following rules:

- Convert inflected word forms to common word lemmas. Use the WordNet lexical database [91] to perform this transformation (e.g., “deleting cookies” becomes “delete cookie”)
- Remove all instances of stop words (such as “and”, “the”, “to”, “but”, etc.)
- Remove words devoid of alphabetic letters (e.g., “3.6.10”, and other non-English strings)
- Sort the query terms alphabetically.

Using this technique, it is possible to achieve a modest reduction in the size of the data set. As an example, the Firefox data set of 74,795 unique queries is represented by 39,435 canonical query groups (53% of the original size). A group’s cardinality (number of distinct query phrasings) is also related to the popularity of the group’s overall topic or concern; compared to less popular topics, those experiencing high search volume yield logs that contain a more complete sampling of the alternative phrasings with which those queries can be expressed. Consequently, those high-volume queries tend to form groups of higher cardinality. To illustrate this point, Table 3.7 lists the cardinality of the canonical groups associated with the top 10 “firefox how to” queries already mentioned in the introduction. All but the last of these queries fall within the top 99.6th percentile of group sizes, thus reinforcing the popularity of these concerns.

3.5 Final output of CUTS

The output of CUTS is a categorized and ranked list of query groups relating to the system under investigation. A sample of this output, for the Firefox application, is presented in Table 3.8. The final ranking of groups is determined by summing the actual or predicted search volumes of each group’s member queries, and then sorting those groups accordingly.

"Firefox how to" ...	Canonical Form	Cardinality of group
clear cache	cache clear	110
delete cookies	cookie delete	60
clear cookies	clear cookie	44
enable java	enable java	41
export bookmark	bookmark export	40
enable cookies	cookie enable	32
clear history	clear history	30
block websites	block website	29
get menu bar back	back bar get menu	16
clear browsing history	browse clear history	5

Table 3.7: Canonical groups associated with the top 10 "firefox how to" queries.

Groups Containing Opr. Instr. Queries	Groups Containing Troubleshooting Queries
cache clear	not respond
clear cookie	not open pdf
cookie delete	slow
block website	crash
cookie enable	mode safe
proxy	check not spell
delete history	constant crash
speed up	lag
bookmark remove	not password remember
...	...

Table 3.8: Query groups, related to Firefox, output after query harvesting, ranking, labeling, filtering and grouping.

3.6 Examples and Case Studies

In this section, I apply the technique to a number of different interactive systems. The goal here is to demonstrate the wide range of insights that can be gained using this approach. I structure this section by showing how issues related to language, desired functionality, and poor affordances can all be detected using this technique.

3.6.1 The Vocabulary Problem (i.e., "Speak the user's language")

Query logs provide an excellent view of the vocabulary and terminology with which users conceive their use of interactive systems. However, this terminology does not always match that which is used by their systems. When such discrepancies arise, the associated systems can be considered to be in violation of Jakob Nielsen's "Speak the User's Language" usability heuristic [94], or as suffering from the vocabulary problem [53]. I provide two examples of this problem that I identified using the technique.

Black and white, but not grayscale

On May 23rd, 2010, I harvested 14,559 queries relating to the GIMP raster graphics editor. Analysis of the GIMP data set reveals 70 distinct queries inquiring how to convert a color image to black and white (Table 3.9). Together these queries are searched an estimated 560 times per month, or about once every 78 minutes on average.

Inspecting GIMP's interface (version 2.6) reveals that there are at least three alternative methods for converting a color image to "black and white". These methods are labeled as "grayscale", "desaturate", and "channel mixer". Such technical terms may not be familiar to a sizeable portion of GIMP's user base, as evidenced by the vocabulary used in the harvested queries. This issue of vocabulary mismatch is revisited in the next chapter.

Clip, but not crop

Inkscape is an open source vector graphics editor similar to Adobe's Illustrator program. On May 22nd, 2010, I harvested 2,501 queries relating to Inkscape. Interestingly, the 8th highest volume query was [*inkscape crop*], with an average of 480 searches performed each month. However, being a vector graphics application, Inkscape does not have a "cropping" tool; cropping is specific to raster graphics. The equivalent operation for vector graphics is to "clip". This very popular query suggests that new Inkscape users are relying on Google to translate knowledge from one domain (i.e., raster graphics) to another domain (i.e., vector graphics). This behaviour closely resembles similar behaviour exhibited by programmers' use of Google [16]. Recognizing this issue, Inkscape could provide a "crop" command or a help entry that assists users in setting the clipping region of their document.

3.6.2 Desired functionality

In addition to identifying potential usability issues related to terminology, I found query log analysis to be an excellent source for discovering desired functionality.

Query	Monthly Searches	Query	Monthly Searches
gimp convert to black and white	91	gimp image black and white	3
gimp black and white	91	gimp convert color to black and white	3
gimp black and white conversion	58	gimp help black and white	3
black and white gimp	28	gimp convert image to black and white	3
gimp color to black and white	22	make image black and white gimp	3
gimp make black and white	16	how to make a photo black and white in gimp	3
how to make a picture black and white in gimp	12	gimp turn image black and white	3
how to make an image black and white in gimp	12	gimp tutorials black and white	3
gimp make image black and white	9	how to make black and white in gimp	2
convert to black and white gimp	9	gimp channel mixer black and white	2
gimp how to make black and white	8	gimp making image black and white	2
gimp black white	8	gimp converting to black and white	2
gimp black and white background	7	gimp make a photo black and white	2
gimp black white filter	7	gimp focal black and white	2
gimp colour to black and white	7	how to make image black and white in gimp	2
gimp change color to black and white	6	gimp make an image black and white	2
gimp save as black and white	6	gimp black and white image	2
gimp black and white photo	6	gimp and black and white	2
gimp tutorial black and white	6	gimp black and white effect	2
gimp black and white only	5	gimp black white conversion	2
gimp black and white filter	5	gimp black and white tutorial	2
how to black and white gimp	5	gimp picture black and white	2
gimp invert black and white	5	gimp black and white plugin	2
gimp to black and white	5	gimp turn black and white	2
gimp black and white layer	5	gimp black and white with colour	2
gimp image to black and white	5	gimp convert photo to black and white	2
gimp change to black and white	5	make black and white in gimp	2
gimp how to make an image black and white	5	color to black and white gimp	2
gimp convert black and white	4	gimp change white to black	2
gimp make picture black and white	4	gimp only black and white	2
gimp layer black and white	4	convert image to black and white gimp	2
gimp how to black and white	4	gimp change picture to black and white	2
gimp make layer black and white	4	gimp true black and white	2
how to make pictures black and white in gimp	4	make an image black and white in gimp	2
gimp change image to black and white	4	gimp color image to black and white	2

Table 3.9: 70 distinct variations of the query "gimp convert to black and white", together with their estimated average monthly search traffic. In total, it is estimated that this task is queried 560 times a month on average, or about once every 78 minutes.

Blocking unwanted calls

One popular class of queries related to Apple's iPhone product inquires about the possibility of selectively blocking unwanted calls from specific telephone numbers. In 2010, when the data was collected, this feature was not supported by the device, and users searched for information on performing this task at least 5,800 times a month (or once every 7.5 minutes). A workaround popular in the user community was to associate a silent audio clip as the ringtone of unwanted telephone numbers. That this issue was so popular suggests users would have been well-served if provided with a sanctioned means of achieving this same behaviour. Such a sanction means arrived in In September 2013, with the 7th major version of the iPhone operating system (iOS 7).

Changing screen savers

Another example of identifying desired functionality emerged when analyzing the searches specific to Amazon's Kindle eBook reader. Specifically, query log analysis revealed 89 distinct phrasings of the query [*how to change your kindle screensaver*]. In 2010, the Kindle device shipped with a few dozen stock images that were displayed by the device when not in use. However, these images could not be customized by the end user. Again, the popularity of these searches suggested that such a feature would be welcomed. In 2011, the Kindle's manufacturer, Amazon, leveraged the non-customizable screensaver to display advertisements to users.

Drawing shapes in GIMP

Finally, an analysis of the GIMP query data set reveals many queries related to drawing primitive shapes: Roughly 130 unique queries inquire about drawing various types of lines, 80 unique queries inquire about drawing circles, 40 queries inquire about drawing rectangles, 20 queries inquire about drawing squares, and 14 queries inquire about drawing ellipses. Moreover, the suggestions [*gimp how to draw a line*], appears in the top 10 suggestions for the prefix "gimp how to", and the Google AdWords tool reports that the query [*gimp draw circle*] is performed an average of once an hour, each and every day. These queries are noteworthy because GIMP provides no explicit tools for drawing simple shapes. Dedicated tools for these functions would likely find great use by GIMP users.

3.6.3 Ubuntu Linux case study

As a final case study of the types of problems that can be uncovered using query log analyses, I consider how developers of the Ubuntu Linux operating system responded when presented with data and analyses mined from CUTS. In October 2010, I presented the CUTS system at the Ubuntu Developer Summit (UDS 11.04). A number of potential issues relating to Ubuntu were discussed in this venue, including those enumerated in Table 3.10. When presenting these findings, audience members were very enthusiastic, and responded by *filing bug reports, while seated in the audience, as I was delivering the presentation* (e.g., Figure 3.10). I discuss two of the examples I presented in the sections that follow.

Poor discoverability of desktop features

One of the chief advantages of leveraging query data is that, unlike other forms of software instrumentation [4], it can reveal potential usability problems arising from poor discoverability of system features. For example, in examining data mined via CUTS, it was discovered that users asked [*ubuntu where is the trash*] in about 120 different ways,

Example Query	Frequency of Searches
ubuntu sound not working	15 minutes
ubuntu login as root	30 minutes
how to tell what version of ubuntu	60 minutes
ubuntu where is the trash	1.2 hours
ubuntu enable dvd playback	1.4 hours
how to mount ntfs in ubuntu	1.4 hours
ubuntu disable touchpad while typing	4.6 hours
ubuntu volume control missing	4.7 hours
ubuntu 32 or 64 bit how to tell	10 hours

Table 3.10: A listing of the potential usability problems, as enumerated using CUTS, presented to Ubuntu developers at the Ubuntu Developer Summit in 2010.

on average once every 72 minutes. These queries included requests for instructions on placing the a trashcan icon on Ubuntu's desktop (e.g., [*ubuntu add trash to desktop*]). Inspection of the Ubuntu's desktop interface revealed that there was *already* a trashcan icon on the desktop, but its placement and size might cause it to go unnoticed by Ubuntu users (Figure 3.9). To this end, CUTS provides concrete numbers describing just how frequently people fail to find the icon during day-to-day use.

Ubuntu's "authentication failure"

For reasons of security, Ubuntu disables the "root" superuser account by default, requiring users to issue the "sudo" command to gain superuser privileges. The root account has otherwise been present and used in UNIX and UNIX-like systems for decades.

While Ubuntu's policy is arguably a positive change for security, the operating system may not be adequately communicating this policy to new users: Attempts to log in as the root user (in Ubuntu version 10.04) simply result in an "authentication failure" error message. An analysis of the queries related to Ubuntu reveals nearly 130 distinct query phrasings all asking about how to access the root user account. In 2010, the specific query [*ubuntu login as root*] was performed 720 times a month, or about once an hour. Similarly, a search for the error message [*su authentication failure ubuntu*] occurs about once every 7 hours. These findings suggest that users would be well served by a more helpful or detailed error message which could communicate the proper course of action when attempting to login as the root user.

When details of this authentication issue were presented to developers at the Ubuntu Developer Summit, audience members responded by immediately filing the bug report #667509 (Figure 3.10) entitled "*su's "authentication failure" error should help users discover sudo*" [19]. This suggests that CUTS can reveal, or can emphasize the severity of, usability problems that developers may otherwise fail to notice or appreciate. I expand on these possibilities, and on the role of CUTS, in the sections that follow.

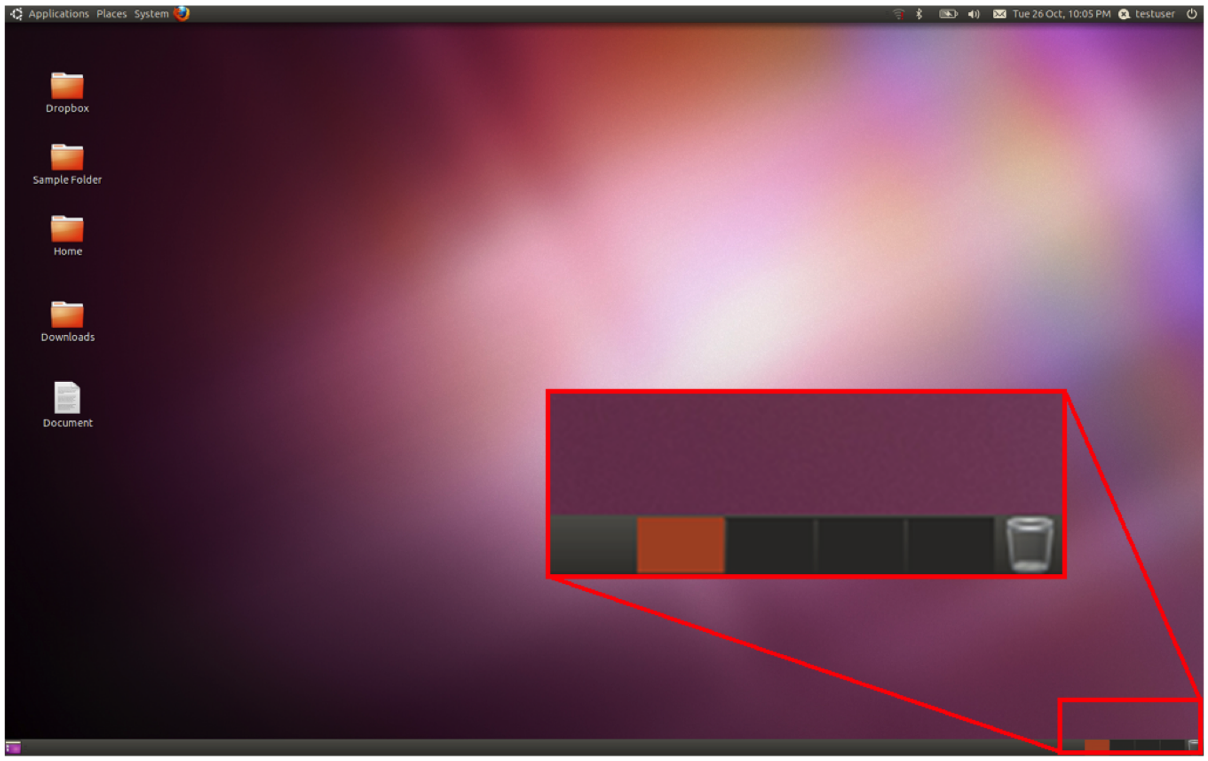


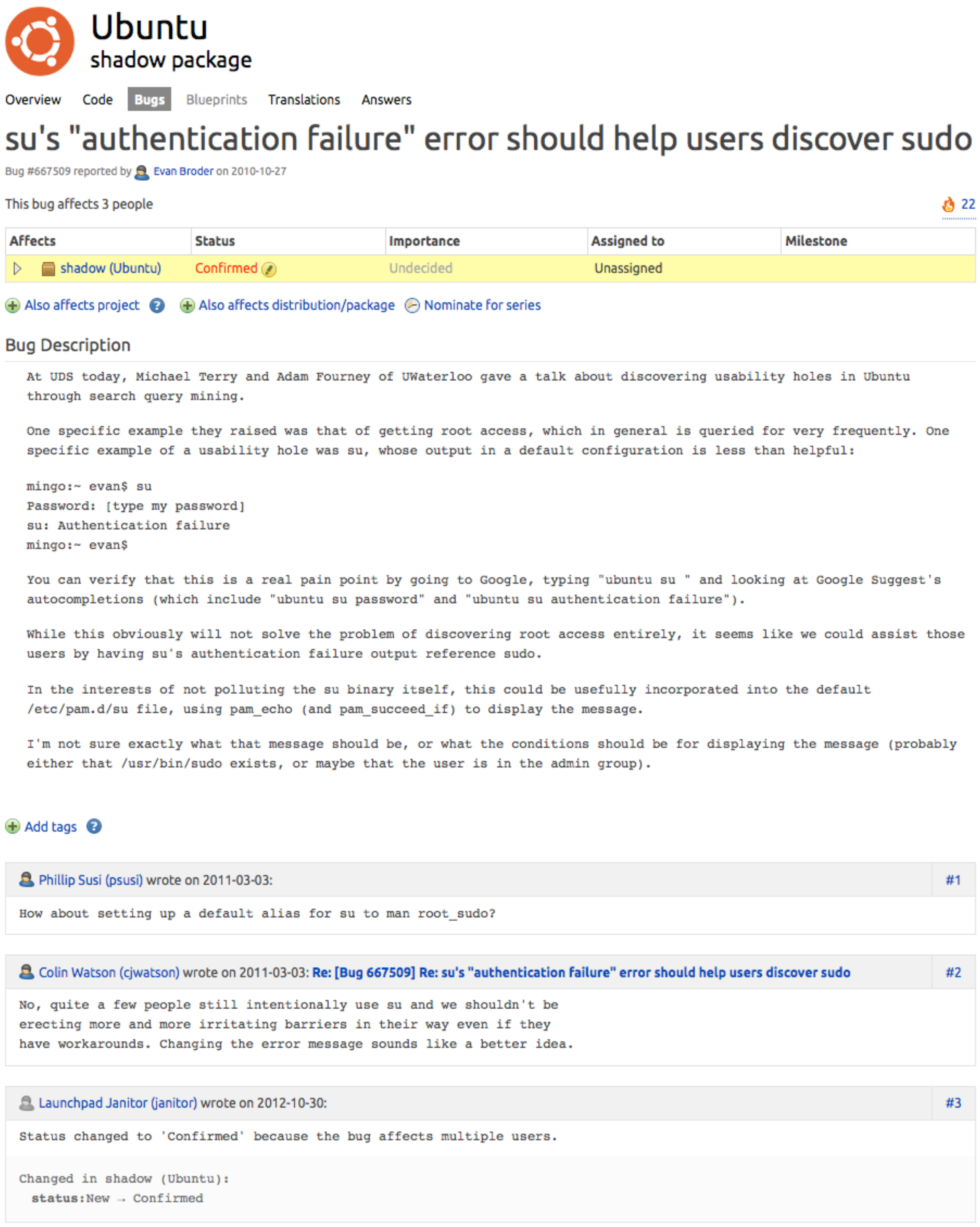
Figure 3.9: The desktop interface of Ubuntu 9.10. Analysis of query data pertaining to Ubuntu reveals that many users fail to notice the trashcan in the bottom right corner of the screen (inset and enlarged in this screenshot).

3.7 Discussion

In this section, I more broadly discuss issues related to using query logs to understand the needs of users of interactive systems. I begin by comparing the output of CUTS to manually curated "frequently asked questions" (i.e., FAQs). I then discuss how query log analysis can factor into existing usability practices, and I enumerate various issues that may affect the rankings produced by this method.

3.7.1 Comparing CUTS' output to published FAQs

CUTS reveals search queries that are frequently performed by a system's users. As such, its output is directly comparable to lists of frequently asked questions (FAQs) commonly provided as documentation for many software applications. However, standard FAQs are curated by individuals, and require continual maintenance and individual judgement regarding inclusion of content. Accordingly, I expect CUTS to more accurately represent the needs of the user base over time. A comparison of CUTS results with the GIMP FAQ lends support to this notion.


Ubuntu
shadow package

Overview Code Bugs Blueprints Translations Answers

su's "authentication failure" error should help users discover sudo

Bug #667509 reported by Evan Broder on 2010-10-27

This bug affects 3 people 22

Affects	Status	Importance	Assigned to	Milestone
shadow (Ubuntu)	Confirmed	Undecided	Unassigned	

Also affects project Also affects distribution/package Nominate for series

Bug Description

```
At UDS today, Michael Terry and Adam Fourney of UWaterloo gave a talk about discovering usability holes in Ubuntu
through search query mining.

One specific example they raised was that of getting root access, which in general is queried for very frequently. One
specific example of a usability hole was su, whose output in a default configuration is less than helpful:

mingo:~ evan$ su
Password: [type my password]
su: Authentication failure
mingo:~ evan$

You can verify that this is a real pain point by going to Google, typing "ubuntu su " and looking at Google Suggest's
autocompletions (which include "ubuntu su password" and "ubuntu su authentication failure").

While this obviously will not solve the problem of discovering root access entirely, it seems like we could assist those
users by having su's authentication failure output reference sudo.

In the interests of not polluting the su binary itself, this could be usefully incorporated into the default
/etc/pam.d/su file, using pam_echo (and pam_succeed_if) to display the message.

I'm not sure exactly what that message should be, or what the conditions should be for displaying the message (probably
either that /usr/bin/sudo exists, or maybe that the user is in the admin group).
```

Add tags

Phillip Susi (psusi) wrote on 2011-03-03: #1

```
How about setting up a default alias for su to man root_sudo?
```

Colin Watson (cjwatson) wrote on 2011-03-03: Re: [Bug 667509] Re: su's "authentication failure" error should help users discover sudo #2

```
No, quite a few people still intentionally use su and we shouldn't be
erecting more and more irritating barriers in their way even if they
have workarounds. Changing the error message sounds like a better idea.
```

Launchpad Janitor (janitor) wrote on 2012-10-30: #3

```
Status changed to 'Confirmed' because the bug affects multiple users.

Changed in shadow (Ubuntu):
  status:New → Confirmed
```

Figure 3.10: Ubuntu bug report filed in response to my presentation of CUTS data to developers attending the Ubuntu Developer Summit in 2011 (UDS '11). This web page was retrieved on May 21st 2015, from [19].

The GIMP FAQ[1] contains 25 questions/answers in the section entitled "Using GIMP." Sixteen of these issues overlap the top issues found using CUTS. The FAQ issues that don't overlap with CUTS results tend to be quite specialized (e.g., "How do I configure X server to do global gamma correction?"). Since GIMP's user base primarily consists of casual users who perform relatively simple tasks [80], very few users will benefit from the answers to these specialized questions. In contrast, CUTS reveals a more representative set of questions related to the simple tasks users have been found to perform (e.g., "gimp how to convert to black and white").

3.7.2 Integrating query log analysis in usability practices

Throughout the chapter, I have been careful to note that query logs can be used to identify *potential* usability problems of interactive systems. While a query may *suggest* that users are experiencing difficulties with a particular aspect of the system, further details and context are required before one can conclude the nature and severity of a potential issue. This additional information can be obtained using standard evaluation techniques involving users or expert evaluators. Since many methods (e.g., cognitive walkthrough) require representative tasks to be identified for evaluative purposes, CUTS can assist by supplying a ranked list of common tasks and needs.

A ranked list of common queries can also be used to assign importance to existing lists of known usability issues. The benefit of using the results of CUTS is that this ranking is derived from the search behaviour of thousands, if not millions, of users. Software producers with limited resources, including volunteer-driven open source products, could thus benefit from this additional means of prioritizing efforts to address known and newly discovered usability problems.

3.7.3 Factors that may impact or compromise query ranking

To effectively use query analysis, it is important to understand and consider the various factors that can impact the weighting and ranking that such analysis produces. In this section, I discuss various effects that influence how often various searches are performed.

User search behaviours and query reformulation

A growing body of research (e.g., [7, 65]) examines user search behaviour. One of the practices observed is that people reformulate their queries when search results do not match their expectations or needs. As an example, a user might start with queries consisting of a few words, and then pose more detailed questions as they fail to find relevant documents in the search results [7]. As a result of this query reformulation strategy, it is conceivable that the analysis proposed in this chapter artificially inflates

[1]http://www.gimp.org/docs/userfaq.html

the importance of issues which are difficult to search for online (e.g., issues where relevant information is scarce). This issue is an artifact of the use of aggregate data, and can be correctly accounted for when session-level query logs are available.

Products with generic names

A number of products have relatively generic names (e.g., Microsoft "Word", etc.), which can cause many irrelevant or off-topic queries to appear in query logs. A similar problem is encountered by products whose names are now synonymous with a class of operations or applications. For example, an altered digital image is often described as being "Photoshopped," regardless of which software application was used for image manipulation.

In these problematic cases, I have found the described filtering techniques (e.g., "how to __ in photoshop") are often enough to filter out the less desirable, off-topic queries.

I also suspect that it is possible to differentiate between the uses of a word by analyzing the results that search engines return for those queries. Search engines are designed to return relevant documents, and often refine their relevance rankings by observing which pages users visit after performing searches [8]. The query-document associations recorded by search engines provide a wealth of information that can further guide analysis of query logs. I revisit this topic in Chapter 4.

3.7.4 Impact of system popularity

In the interest of preserving user privacy, query auto-completion services are unlikely to suggest queries unless those queries have been performed many times, and by many different individuals. As such, the quantity of data available for analysis by CUTS is related to the popularity of the interactive system being studied. In the extreme case, CUTS cannot be used to study systems that are not publicly available (e.g., custom software solutions, prototypes, beta software, etc).

3.8 Conclusion

When faced with difficulties or questions relating to the use of interactive systems, many people routinely turn to Internet search engines as a first line of support. In this chapter, I have introduced CUTS: characterizing usability through search. This process takes the name of an interactive system as input and outputs a ranked and categorized list of common tasks and potential issues that users encounter with that system. These data are assembled by sampling from the query logs of top-tier Internet search engines using public interfaces. Importantly, the results of this process have a high degree of ecological validity, and can directly inform more formal evaluation methods by suggesting particular tasks or issues to study.

In the context of the Gulf of Execution, web search and CUTS represent the first step across the web-mediated execution bridge. In the next chapter, Chapter 4, I examine opportunities that arise when one considers the second step of the journey: i.e., retrieval of relevant online web documents and instructional materials.

Chapter 4

Query-Feature Graphs

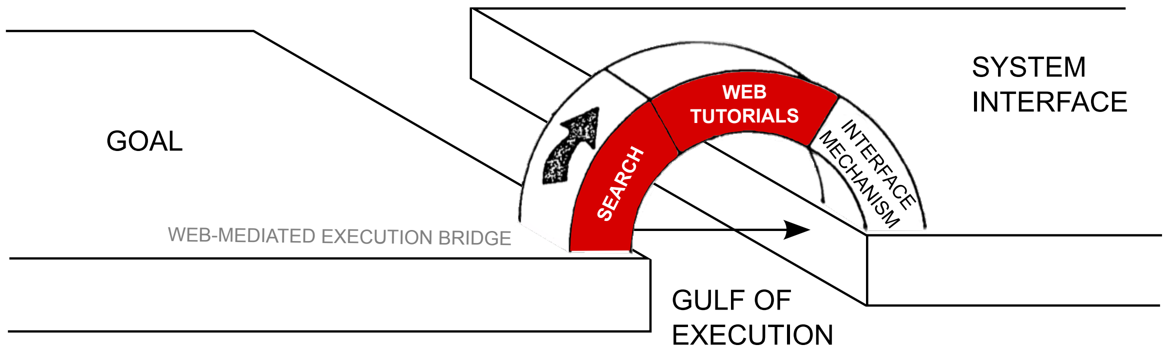


Figure 4.1: This chapter introduces query-feature graphs – structures that address an important instance of the vocabulary problem [53]. Query-feature graphs are constructed by mining the co-ocurrences of commands and query keywords appearing in the texts of online tutorials. In this sense, query-feature graphs connect the first two steps of the web-mediated bridge that crosses the Gulf of Execution. Portions of this chapter were first published in [49].

The techniques and datasets presented in the previous chapter afford a first detailed view of the types of web searches users issue when starting their journeys across the Gulf of Execution. In examining these popular queries, a common theme emerges: Users often employ web search to overcome the *vocabulary problem* [53], an issue that arises when users conceptualize and articulate their needs in ways that do not match the (rather terse) vocabulary of the interactive system. Furnas et al. describe the problem as follows:

People use a surprisingly great variety of words to refer to the same thing. In fact, the data show that no single access word, however well chosen, can be expected to cover more than a small proportion of users attempts. (...) In current computer systems, the vocabulary problem is largely ignored. Designers *decide on the terms to be used, and, as heavy users, grow to find these terms obvious and natural. Other users are simply required to learn the systems words [53].*

As an example of the vocabulary problem arising in a modern feature-rich application[1], consider again the search [*gimp black and white*], a commonly executed query used to learn how to simulate the effect of black and white film using the GIMP raster graphics editor. Since GIMP has no command named "black and white," as discussed in Chapter 3, users who wish to achieve this effect must learn that commands such as "desaturate," "grayscale," or "channel mixer" will yield the desired effect. Indeed, searching for the phrase [*gimp black and white*] returns web pages describing the use of these very commands. In essence, these web pages serve as "Rosetta Stones" between users' conceptualizations of tasks, as expressed by search queries, and the actual tools necessary to accomplish those tasks.

Inspired by this manual process, this chapter presents a system that automatically combines a corpus of common search queries, and a corpus of web tutorials retrieved from those searches, to uncover relationships between users' vocabularies and the relevant system components. These pairings are represented in what I call a *query-feature* graph (Figure 4.2), or *QF-graph* (where "feature" in this context refers to elements in an interactive system). Once formed, a QF-graph can provide the foundation for a range of novel interaction techniques. In this chapter, I illustrate its potential by outlining three possible interaction techniques:

- a search-driven interface in which users type the task they wish to accomplish, and the interface assembles a list of the most relevant commands for the task
- dynamic and ever-evolving tooltips that display tasks which reflect how the user community uses a given command
- app-to-app analogy search, which provides a mapping between the tools necessary to perform a task in one interface and the equivalent tools in a second interface

Collectively, the QF-graph, its mode of construction, the validation of the technique using real-world data, and the example uses of the QF-graph constitute the primary contributions of this chapter. In the grander context of the web-mediated execution bridge, QF-graphs unite the first two steps across the Gulf of Execution.

In the remainder of this chapter, I define the query-feature graph and describe its automated construction in more detail, then presents the results of an evaluation that assesses the quality of the query-feature associations expressed in QF-graphs. I describe three novel interaction techniques enabled by a QF-graph, then conclude with a discussion of the approach's limitations and directions for future research.

[1]Furna et al's characterization of the vocabulary problem was originally published in 1987; however, the data characterizing common web search queries suggest that the interaction challenges arising from the vocabulary problem persist to this day.

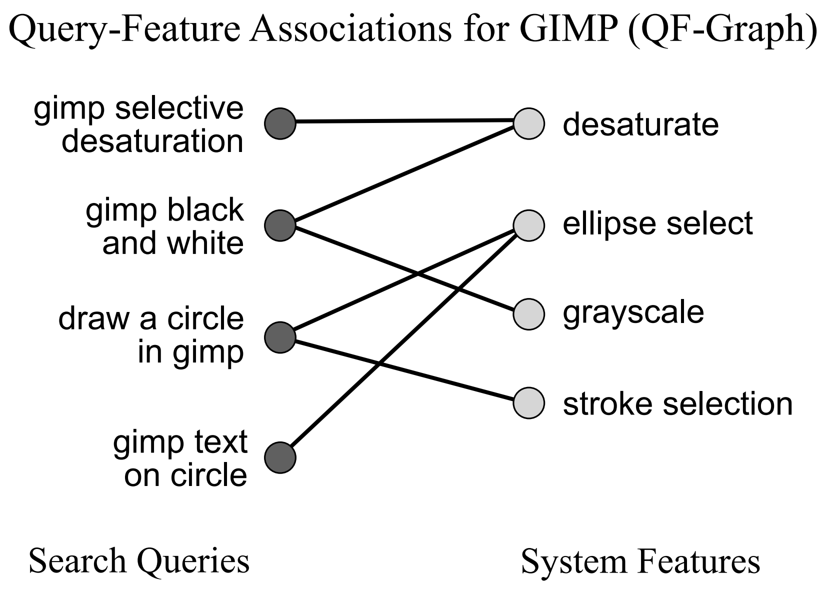


Figure 4.2: Query-feature graphs pair tasks, as naturally expressed in user search queries, with relevant system features.

4.1 The query-feature graph

The query-feature graph directly associates user search queries with relevant system features (commands, menu items, dialogs, preferences, etc.) via an undirected weighted bipartite graph. Formally, the query-feature graph, $G = (\{Q, F\}, E)$, is composed of the following components:

- **Graph Vertices:**

  $Q = \{q_i \,; 1 \leq i \leq N\}$

  Where Q is a set of distinct search queries pertaining to the use of a given interactive system.

  $F = \{f_j \,; 1 \leq j \leq M\}$

  Where F is a set of features (e.g., commands, menu items or other interface components present in the system).

- **Graph Edges:**

  $E = \{(q_i, f_j, w_{ij}) \,; q_i \in Q, f_j \in F, w_{ij} \in \mathbb{R}\}$

Where E is a set of 3-tuples, each representing a weighted edge from a query vertex to a feature vertex. Each weight, w_{ij}, expresses the strength of the association.

QF-graph are created by first enumerating the relevant search queries and system features that populate the vertex sets Q and F, respectively. Associations between queries and features are then established using techniques from question-answering research, which leverage co-occurrences of query terms and mentions of system features in relevant online tutorials. I describe each of these steps in detail below.

4.1.1 Enumerating relevant search queries (Populating Q)

Assembling a meaningful set of queries, Q, is the first challenge in creating a QF-graph. An obvious way to populate this set, then, is to sample the search query logs of web search providers, looking for queries mentioning the interactive system. However, search query logs are not made publicly available. To approximate search query logs, I leverage the *CUTS* procedure described in chapter 3. CUTS leverages query auto-completion services (e.g., "Google Suggest") to sample popular queries from the logs of top-tier search engines. For publicly available software applications, this technique has been demonstrated to reveal tens or hundreds of thousands of queries for a given system.

4.1.2 Enumerating system features (Populating F)

To enumerate the features, F, of a system, I employ a variety of techniques ranging from manual enumeration (e.g., with the Amazon Kindle), to extracting all strings contained within a system's string tables, which are used for language localization (e.g., with Chrome). For feature-rich applications, there are typically hundreds of commands. When strings are extracted from string tables, there are often thousands of strings, many of which represent ancillary text such as error messages and the text of tooltips or other contextual help. Section 4.2.3 discusses some of the challenges posed by these ancillary strings. Table 4.1 lists the five interactive systems used as examples throughout the rest of this chapter together with the mechanism used to enumerate features.

4.1.3 Associating Queries with Features (Populating E)

Associating queries with features is the final step in constructing a QF-graph. In this section, I present the specific challenges inherent in this final step, then describe the use of the question-answering approach, *QAP* [24], to establish query-feature relationships.

System	Description of Feature Vertex Set F
Kindle	**210 commands** discovered through manual exploration of the interface.
GIMP	**830 commands** enumerated by the ingimp [122] project.
Inkscape	**1785 strings** extracted from Inkscape's primary en-US string table.
Firefox	**583 strings** extracted from the interface markup (XUL) files used to layout Firefox's UI.
Chrome (Chromium)	**3088 strings** extracted from Chrome's primary en-US string table.

Table 4.1: The five interactive systems for which QF-graphs were generated, along with a description of the data sources used to populate each graph's feature vertex set.

Challenges

The goal of a QF-graph is to associate the text of queries with the text representing software features. However, both queries [69, 70] and software features typically consist of only a few words, limiting the range of approaches that can be used to establish associations. Specifically, simplistic term-matching approaches such as the vector space model of information retrieval [110] cannot be readily applied: In the vector space model, the similarity of two phrases depends on the set of terms that the two phrases have in common. If the phrases share few words, then their similarity score is low. In this case, the use of term overlap is further confounded by the fact that many queries are task or goal-related and tend to have few words in common with any particular feature of the system.

In order to address the issue of term sparsity, one can simply emulate existing search practices: Each search query can be submitted to a search engine, the relevant web pages can be retrieved, and the commands, actions or tools mentioned within the web pages can be identified. This process enables the creation of associations between search queries and related system functionality.

The strategy of using document retrieval to enrich or expand the set of terms associated with a short phrase is not new. Specifically, Bernstein et al. employed document retrieval to help cluster short Twitter messages [13], and Shen et al. used document retrieval to help classify search queries [113, 114]. However, while this approach has been found to be very effective in these latter contexts, in the context of pairing search queries with specific elements of an interface, this technique breaks down when the resulting documents are multi-topical. For example, a query to learn how to perform a particular task with an application can yield web pages on user forums, frequently asked question (FAQ) pages, or blogs, all of which are inherently multi-topical documents that can reference a wide range of system features in the same document. The challenge, then, is to determine which portion of a document is most relevant to a given search query.

To address this problem of multi-topical documents, one can repeat the basic search process *within* each document, identifying and retrieving short passages that are most relevant to the original query. These passages can then be processed to identify features mentioned therein.

This pipeline of retrieving relevant documents, retrieving relevant passages, then identifying a set of mentioned system commands or features, is essentially the same as that employed by many question answering (QA) services (e.g., see [121]). As such, I employed the QAP (Question Answering Passage) algorithm originally described by Clarke et al. [24] to discover associations between queries and system functionality. I briefly describe QAP in the next section. Interested readers are directed to [24] and [25] for a more thorough treatment of the QAP algorithm.

An Overview of QAP

QAP proceeds in three distinct steps: 1) retrieve short document passages relevant to the user's question or query, 2) identify potential answers in those passages, and 3) rank potential answers across numerous pages and passages so that a final summary can be presented to the user.

Step 1: Passage Retrieval

The first step of QAP is to retrieve passages relevant to the user's query. QAP employs a cover-density ranking approach that treats all document substrings that both begin and end with a query keyword as potential passages. The details of this cover-density ranking are described in [24] and [25]. Notably, the ranking weighs the number of query keywords contained within each substring against the substring's length. Favourable ranks are assigned to short substrings that contain many query keywords. Once substrings are scored, longer fixed-length passages are extracted by expanding each substring about its midpoint. I elected to extract passages consisting of 300 words after early experiments suggested that this value was effective for the types of web pages and documents available for analysis (forums, blogs, etc.). The original QAP paper, [24], utilized passages consisting of 200 words.

The original QAP papers employed cover-density ranking over all documents in the corpus. In the QA literature, it is more common to first limit the search space by creating a short list of documents that are potentially relevant to the original question or search query [121]. Passage retrieval is then employed over this smaller set of documents. This first step is accomplished by simply using Google's public search API [33] to retrieve the top 8 documents for each query. Whether using the original QAP formulation, or this modification, each document contributes only its highest scoring passage to the next step of the process.

Step 2: Answer Extraction

The second step of QAP is to identify potential answers mentioned within the top k scoring passages. In this chapter $k = 8$. In the original QAP work, each query was analyzed to determine the form of its expected answer. Depending on the question, answers might take the form of dates, proper names, cities, numbers etc. Such answers can be detected using a named entity recognizer, or by matching a passage's phrases against lists of potential answers. In the context of QF-graphs, document phrases are matched verbatim against the features enumerated in the QF-graph's vertex set F. Since developing query-feature graphs in 2011, more sophisticated methods of detecting commands mentioned in documents have been developed. Specifically, named entity recognition methods have been adapted to address this need [84, 47, 101]. See Appendix A for more details.

Step 3: Ranking Answers

The final step of QAP is to rank the potential answers that were identified in passages. To do this, QAP exploits the built-in redundancy of the web: The web is a large corpus, and the answer to any question is likely to be found in many documents. As such, QAP ranks answers using the following measure:

$$\text{score}(q, f) = n_{q,f} \times log\left(\frac{|D|}{d_f}\right) \tag{4.1}$$

Here, $n_{q,f}$ represents the number of passages returned for query q in which the feature f is mentioned. Similarly, d_f is the number of distinct documents in the corpus in which the feature's corresponding phrase f occurs, and $|D|$ is the number of documents in the corpus overall. The term $|D|/d_f$ is just the familiar inverse-document frequency (*idf*) of the system feature f. The *idf* statistic is computed from the corpus of web pages relating to the interactive system under investigation. Depending on the interactive system, this corpus consists of tens or hundreds of thousands of web documents collected using standard web crawling practices.

Using QAP to generate edges

At the heart of the QAP question answering algorithm is the function $score(q, f)$, given by equation 4.1 above. This function assigns a numeric score to the tuple (q, f), expressing the strength of the association between the query q and system feature f. This score provides a means for weighting the edges in the QF-graph. Specifically, G's edges are defined as follows:

$$\begin{aligned} E = \{ \quad & (q_i, f_j, w_{ij}); \quad q_i \in Q, \quad f_j \in F, \\ & w_{ij} = \text{score}(q_i, f_j) \quad \} \end{aligned} \tag{4.2}$$

Additionally, notice that QAP accepts any sequence of words as input. As such, the QF-graph can be actively updated as new searches are performed. This is accomplished by simply appending new queries to the query vertex set Q, and executing QAP to establish each new query's associations with the fixed set of system features F.

In theory, this method should establish reasonable connections between user queries and specific elements in the user interface. The next section presents results of my analysis examining the quality of these connections.

4.2 Evaluation

To evaluate the quality of a QF-graph, I employed a variety of metrics. Some of these metrics are standardized and are used throughout information retrieval literature. Another, I developed specifically for this particular problem domain. Since results are often difficult to interpret on their own, I compare QAP results to those achieved when using a more basic term-matching approach for associating queries with features (specifically, cosine similarity ranking using the standard vector space model [110] with tf-idf weighting).

4.2.1 Experiment Setup

The QF-graphs produced by the techniques described in this chapter were evaluated using the following high-level steps. First, a set of test queries is chosen. Second, for each test query, QAP's results are recorded. Finally, the relevance of each result is judged by an expert. Since I have already covered how QF-graphs are generated, I describe the first and last steps of this process.

Selecting test queries

For this experiment, 20 queries were randomly selected from the query-feature graphs pertaining to each of the five interactive systems listed in Table 4.1. Because queries were selected from the query-feature graphs (which itself was built using queries harvested via CUTS), the queries represent real-world searches performed by users.

Judging relevance

Many queries describe a goal or a task that the user would like to perform. One would like to know which system features are relevant to the query, but one would also like to know which sets of features are sufficient for completing the task implied by the search. To accommodate both needs, *solutions* for each test query were manually crafted. A solution is a collection of relevant commands or system features that "solves" or accomplishes the goal implied by the query. As an example, there are two solution sets for the query [*firefox how to clear cookies*]:

1. Clear Recent History, Cookies, Clear Now
2. Preferences, Privacy, Show Cookies,
   Remove All Cookies

A system feature is said to be relevant to a query if the feature appears in any of the query's solutions. Similarly, one says that a set of features, S, *covers a solution* when S contains references to all system features required to implement that solution. By this definition, the full set of features F covers all solutions, and the empty set $\emptyset$ covers no solutions.

Given a selection of test queries and their associated solutions, it is possible to employ a number of metrics to measure the quality of the QAP query-feature associations. I describe these metrics next.

4.2.2 Performance Metrics

Four search-quality metrics were employed to assess the QF-graphs: Mean precision at 1, percent correct at 10, mean average precision, and mean precision at *. Each of these metrics is described below.

- **Mean precision at 1**
  Mean precision at 1, denoted $\overline{P_{@1}}$, is a standard information retrieval metric [124] that measures the proportion of test queries whose top-ranking QAP result is judged to be relevant.

- **Percent correct at 10**
  The *percent correct at 10* measure (%C@10) is another simple metric that measures the proportion of test queries for which at least one correct solution is covered by the query's top-10 QAP results.

- **Mean average precision**
  Mean average precision (MAP) is a widely used information retrieval metric that averages the precision of a set of search results, measured at various levels of recall (see [124] for more details). It is generally described as a measure of the area underneath the precision-recall (or, receiver operating characteristic) curve. MAP scores range from 0 to 1, with higher scores indicating better results.

- **Mean precision at ***
  In question answering literature, one standard measure of performance is *mean reciprocal rank* (MRR) [121]. A query's reciprocal rank is simply the multiplicative inverse of the rank corresponding to the first correct answer in the queries' list of

results. The mean reciprocal rank metric is the average of the reciprocal ranks of all test queries.

In the domain of feature-rich applications, many queries cannot be “answered” by a single result, but rather are “covered” by a set of results that combine to form a correct solution. As such, I developed a metric similar in spirit to MRR, described below, which I refer to as *mean precision at **.

For each query q, let r^* be the smallest integer such that q's top r^* ranking QAP results cover one complete solution for the query. Precision at r^*, denoted $Prec_*(q)$, is the proportion of the top r^* results that are judged to be relevant to the query. $\overline{P_*}$ is the arithmetic mean of these $Prec_*(q)$ scores across all test queries.

Importantly, the $\overline{P_*}$ measure is equivalent to MRR in cases where the query's solution is covered by a single command or system feature.

4.2.3 Experiment Results

Results from applying these four metrics to the experimental QF-graphs are listed in Table 4.2. On average, 77% percent of the test queries were “answered” (or covered) by the top-10 results returned by QAP. Moreover, the first QAP result was relevant to the query in 63% of test cases.

Regarding individual applications, results for GIMP are quite good in part because it was possible to obtain a very accurate list of GIMP's commands, but also because GIMP command names are sufficiently technical to not conflict with everyday language, making it easier to identify commands mentioned in web pages. Similarly, results for Firefox are superior to those of Chrome because Firefox's interface markup language (XUL) allowed the easy identification of strings associated with menu items, buttons and other command-related UI components. Conversely, neither Chrome nor Inkspace's localization databases provide this information. As a result, the Chrome and Inkscape QF feature-sets are much larger and contain a broader set of strings (which explains why table 4.1 lists nearly six times as many strings from Chrome as it does for Firefox). While most of these spurious strings rarely occur in search queries or web documents, some strings (e.g., Chrome's “And then click”) are sufficiently common to introduce errors.

To provide further context for interpreting these results, and to measure the impact of QAP, I repeated the experiment using a typical implementation of the vector space model to match query phrases to system features. This approach is similar to those employed in existing interface search tools (e.g., Mac OS X's help menu search), and ranks query-feature associations by averaging the importance weights of the words that both the query and system feature have in common. A word's importance weight is

System	$\overline{P_{@1}}$	$\overline{P_*}$	MAP	%C@10
GIMP	0.800	0.725	0.467	90%
Firefox	0.750	0.601	0.496	75%
Chrome	0.500	0.598	0.382	75%
Inkscape	0.500	0.536	0.264	70%
Kindle	0.600	0.633	0.458	75%
Overall	0.630	0.619	0.413	77%

Table 4.2: Performance when using QAP for discovering query-feature associations for five different interactive systems.

System	$\overline{P_{@1}}$	$\overline{P_*}$	MAP	%C@10
GIMP	0.450	0.302	0.132	30%
Firefox	0.500	0.501	0.264	70%
Chrome	0.050	0.197	0.074	45%
Inkscape	0.150	0.160	0.081	35%
Kindle	0.400	0.384	0.124	50%
Overall	0.310	0.309	0.135	46%

Table 4.3: Performance when using the vector space model for discovering query-feature associations.

simply its inverse document frequency, described previously. Results from these trials are listed in Table 4.3. Again, the results for Chrome and Inkscape are worse than for the other three applications, and all are substantially worse than those obtained using query-feature graphs. As noted earlier, the Chrome and Inkscape string tables contain spurious strings (e.g., the text of error dialogs, etc.). These much longer strings are favoured by cosine similarity ranking, and the vector space model, because their longer lengths increase the chances of the string and the query having terms in common.

In all cases, with all metrics, QAP's results are superior to those obtained when using simple term matching (the standard vector space model).

From these experiments I conclude that, for a given query, the top-10 query-feature associations discovered by QAP are reasonable, and, in every case, the results obtained using QAP are better than those produced by more typical implementations of interface search. Finally, these experiments establish a baseline with which future research can be compared, and improvements measured.

4.3 Applications

QF-graphs can serve as the computational back-end for a number of novel interface mechanisms. In this section, I describe how the QF-graph can improve search-driven interaction, support dynamic tooltips, and enable application-to-application analogy search. I use example data from actual QF-graphs to illustrate the utility of QF-graphs in these hypothetical interface mechanisms.

4.3.1 Search-driven Interaction

The concept of search-driven interaction is simple: The user types in a few keywords and the system returns a ranked list of relevant system commands and interface components. This style of interaction can be useful when an application's features number in the hundreds or thousands [95].

QF-graphs provide a direct means of supporting search-driven interaction. When the user enters a query, the QF-graph is consulted to retrieve relevant system features. If the QF-graph does not contain an entry for the query, QAP can be used to provide this information on demand.

To illustrate the potential of this approach, the following examples demonstrate search results obtained using QF-graphs for three different interactive systems. For each example, I provide the top five results returned from querying the QF-graph. The queries themselves were drawn from the corpora of queries produced by CUTS, and thus represent frequently issued queries by users.

Query: *"gimp convert to black and white"*
As noted in Chapter 3, this query is issued by users who would like to convert color digital images to images that consists only of shades of gray. This effect is most easily achieved using GIMP's, *grayscale* or *desaturate* commands, but can also be accomplished by manipulating options in the *channel mixer* tool, or by *decomposing* the image in order to extract its Luminosity or Value channels (in HSL or HSV color spaces, respectively).

QF-graph search results:
- Channel mixer
- Grayscale
- Desaturate
- Channels
- Decompose

Query: *"how to get the kindle to read to you"*
In this case, the user would like to enable the text-to-speech feature of the Amazon Kindle. In 2011, at the time of data collection, this task was accomplished by pressing the *Aa* hardware button, then navigating to the *Text-to-speech* section of the dialog, and finally selecting the *turn on* command.

QF-graph search results:

- Aa
- Text to speech
- Turn on
- Web browser
- Down (directional keypad)

Query: *"change download location firefox"*
Here, a user of the Firefox web browser would like to change the location to which Firefox saves downloaded files. This can be achieved by opening Firefox's general preferences and entering a different value in a text field titled "Save files to." Alternatively, the user can check the radio button titled "Always ask me where to save files."

QF-graph search results:

- Save files to
- Always ask me where to save files
- Always ask
- Location
- Save

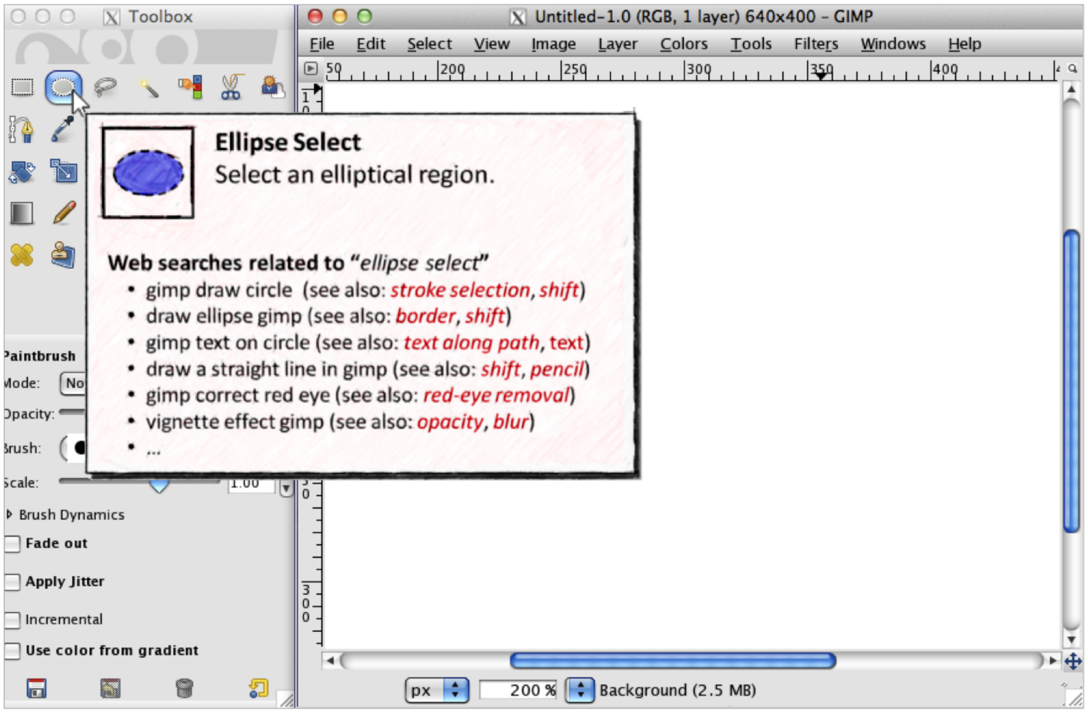
Importantly, each of these three queries represent tasks or goals, as commonly expressed by users, and do not mention any system commands by name. Nevertheless, the QF-graph returns results directly relevant to the goals implied by the queries, demonstrating the utility and robustness of the approach.

4.3.2 Dynamic tooltips

The features represented in a QF-graph can also be "queried" to determine the set of search queries associated with a given command, menu item, or option. This capability motivates *dynamic tooltips.*

Dynamic tooltips extend standard tooltips or balloon help by proactively describing the range of tasks that utilize the command or interface component currently in focus. These task descriptions are derived from QF-graphs, which are themselves derived from real-world user search queries and from web content, such as FAQs, forums, tutorials, and blog posts. As a result, these tooltips dynamically track and reflect current use of the software by the community. Moreover, these queries serve a similar role as anchor text in web-based information retrieval, providing a large set of concise descriptions of how a command is used in practice. Figure 4.3 provides an example of the contents such a dynamic tooltip could display for GIMP's "ellipse select" command (where the contents are derived from GIMP's actual QF-graph).

To generate the contents of a dynamic tooltip, the system first determines which queries are associated with a given system feature f. The set of related queries Q_f is



Ellipse Select
Select an elliptical region.

Web searches related to "*ellipse select*"

- gimp draw circle (see also: *stroke selection*, *shift*)
- draw ellipse gimp (see also: *border*, *shift*)
- gimp text on circle (see also: *text along path*, text)
- draw a straight line in gimp (see also: *shift*, *pencil*)
- gimp correct red eye (see also: *red-eye removal*)
- vignette effect gimp (see also: *opacity*, *blur*)
- ...

Figure 4.3: A mockup of a dynamic tooltip for GIMP's "*Ellipse Select*" command. The list of related searches is derived from GIMP's QF-graph, as are the pairs of commands associated with each search query.

simply the set of vertices neighbouring f in the QF-graph. As an example, Table 4.4 provides a partial list of the queries neighbouring the "ellipse select" command in GIMP's query-feature graph.

An exhaustive examination of these queries (in this case, numbering over 750), reveals references to a variety of tasks other than drawing circles or ellipses (e.g., writing text along a circle, or correcting red eye). However, while there may exist numerous queries, many of these queries refer to the same topic, as can be seen in the partial list of queries associated with "ellipse select" shown in Table 4.4. In other words, there is considerable redundancy in Q_f.

By removing redundancy in the set Q_f, one can more concisely express the variety of tasks in which the command f is involved. In order to remove redundancy in Q_f, one can repeatedly remove queries q whose query-feature edge weight is less than that of some *equivalent query*, $p \in Q_f$. Two queries, q and p, are considered equivalent if they share 4 of their top-5 search results. Applying this procedure to the set of queries related to GIMP's "ellipse select" tool yields the queries listed previously in Figure 4.3.

The queries listed in a dynamic tooltip can also be augmented with an additional list of related commands. Specifically, the top two features associated with each query (excluding the command for which the tooltip is generated) can be displayed to provide more context about the tools necessary to complete the task represented by the query.

Rank	Query
1	gimp draw circle
2	draw a circle in gimp
3	drawing a circle in gimp
4	gimp drawing circle
5	gimp tutorial circle
6	draw ellipse gimp
7	gimp ellipse draw
...	

Table 4.4: A partial list of queries neighbouring GIMP's "ellipse select" command.

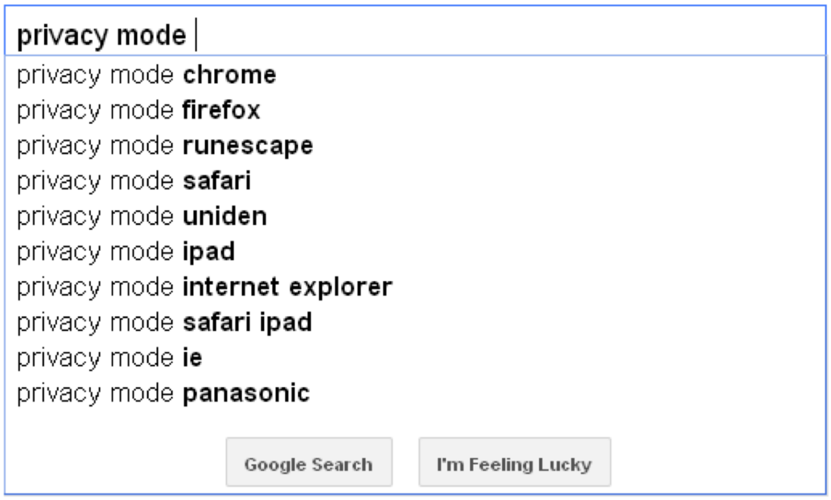
As an example, the dynamic tooltip for the "ellipse select" tool in Figure 4.3 lists an example query, "gimp text on circle", along with two additional commands: "text along path" and the "text tool". Both of these latter commands can be used in conjunction with "ellipse select" to print text along the circumference of a circle. This specific use of QF-graphs is similar in spirit to the user and item-based command recommendations discussed by Matejka et al. in [90]. However, the recommendations generated from QF-graphs are likely to be more task-specific and task-focused since they derive from web pages and tutorials describing specific tasks.

4.3.3 App-to-App Analogy Search

Within a given domain, competing applications often provide similar functionality, but use different naming conventions or vocabularies for those features. As an example, in the domain of web browsers, Firefox's "private browsing" feature is equivalent to Chrome's "Incognito" mode (both modes limit the amount of information tracked and exchanged when browsing the web). Despite these different branding of this common feature, users issue similar queries when searching for these capabilities (since queries typically express a high-level goal). As an example, Figure 4.4 depicts a set of Google auto-complete suggestions for the prefix "privacy mode". Note that the suggestions list many of the most popular browsers (Firefox, Chrome, Internet Explorer / IE, and Safari) despite none of these browsers offering a command, menu item or option named "privacy mode". These similarities in user queries make it possible to associate queries in one application to queries in a second, comparable application.

Linking queries from two different applications serves to connect the QF-graphs of the two applications (Figure 4.5). Once connected, the paired graphs enable *analogy* search, or the ability to directly relate the commands of one application to similar commands in the second application.

As a demonstration of this concept, the results of applying analogy search to the aforementioned private browsing example are listed below. As can be seen, analogy search


privacy mode

privacy mode **chrome**
privacy mode **firefox**
privacy mode **runescape**
privacy mode **safari**
privacy mode **uniden**
privacy mode **ipad**
privacy mode **internet explorer**
privacy mode **safari ipad**
privacy mode **ie**
privacy mode **panasonic**

Figure 4.4: Google auto-complete suggestions for the prefix "privacy mode". The suggestions list many of the most popular browsers despite none of these browsers offering a command named "privacy mode". This example serves to highlight that people often use similar terminology to express concepts, making it possible to link functionality across disparate applications.

is able to correctly associate Firefox's "start private browsing" command to Chrome's "new incognito window" command.

Analogy: *Chrome commands similar to Firefox's "Start Private Browsing" command:*

Results:
- New incognito window
- Incognito
- Session
- And then click

Analogy search begins by identifying the top 10 queries in Firefox's QF-graph related to the "Start Private Browsing" command. For each query, a term frequency inverse document frequency (tf-idf) vector is created, as is standard in the vector space model of information retrieval [110]. A weighted sum of the 10 query vectors is then computed in order to synthesize a single feature vector for the "Start Private Browsing" command. Weights in the summation correspond to the query-feature edge weights in the QF graph. This amalgamation of queries into a communal feature vector helps to mitigate the term sparsity problem, as previously discussed in section 4.1.3. Once the feature vector is computed, an identical process is performed for each of Chrome's commands. The similarity between pairs of commands is then computed as the dot product of the two vectors, and the highest scoring Chrome commands are listed in the search results.

As a further example of the effectiveness of app-to-app analogy search, consider two

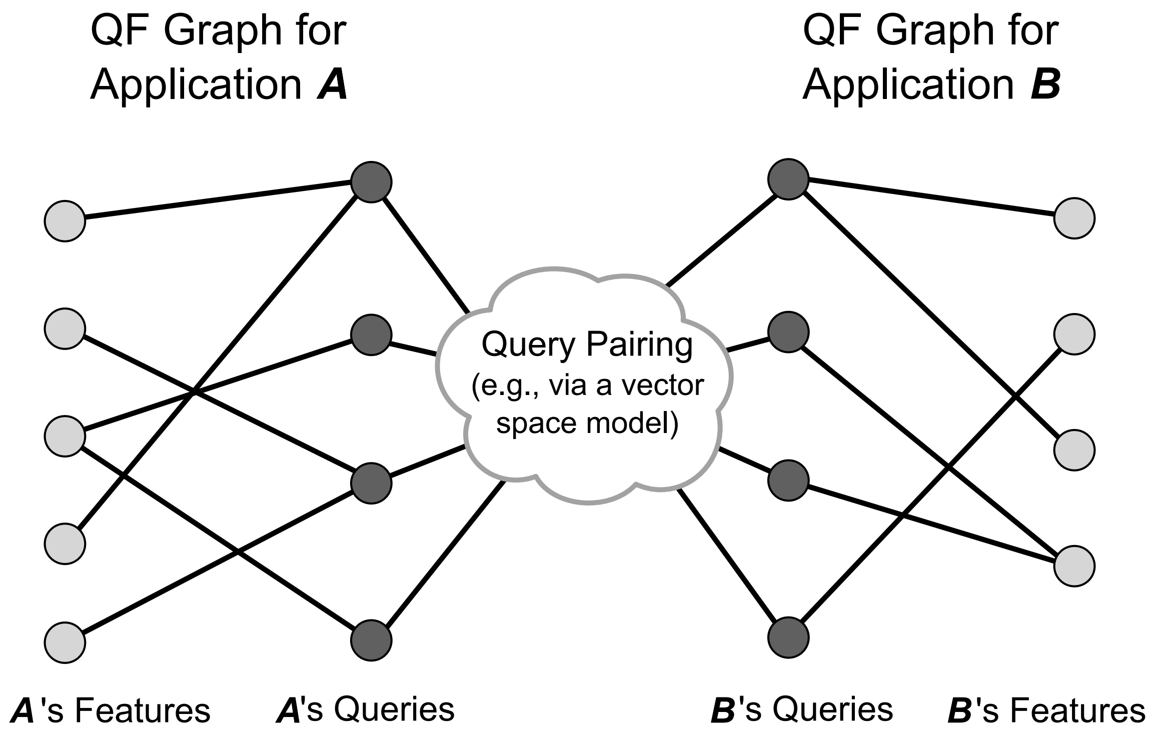

QF Graph for Application *A*
QF Graph for Application *B*
Query Pairing (e.g., via a vector space model)
A's Features
A's Queries
B's Queries
B's Features

Figure 4.5: App-to-app analogy search relates features found in one application to similar features found in other applications. To accomplish this, the QF-graphs of two applications are joined together by using the standard vector space model to identify similar queries.

somewhat different applications: GIMP and Inkscape. GIMP is a raster graphics editor, while Inkscape is a vector graphics editor. Importantly, GIMP and Inkscape both edit images, but do so using vastly different metaphors and data (namely, pixels vs. vectors).

As an example of how these applications differ, GIMP allows users to crop an image using a "crop" tool. To achieve a similar effect in Inkscape, users must first select objects of interest, then either set the "clipping region," or "fit the page to the selection." Despite these marked differences, app-to-app analogy search is able to correctly associate GIMP's crop tool with the appropriate Inkscape commands.

Analogy: *Inkscape commands similar to GIMP's "crop" command:*

Results:
- Crop marks
- Select all in all layers
- Select
- Fit page to selection

Finally, it is also possible to use analogy search to identify related commands *within the same* application. The following example illustrates this point:

Analogy: *GIMP commands similar to GIMP's "stretch contrast" command:*

Results:
- White Balance
- Auto (Levels)
- Stretch Contrast
- Colors

In the above example, analogy search correctly inferred that GIMP's "stretch contrast", "white balance" and "auto levels" commands are related in that they are often used in conjunction (or in place of) one another, in order to enhance a digital photograph. (In this case, each of these commands is used to manipulate an image's histogram.)

4.4 Discussion

I conclude the discussion of QF-graphs by considering some of their limitations, challenges in automatically creating query-feature associations, and directions for future work.

4.4.1 "Feature" ambiguity in web documents

In this chapter, I identified system features referenced within web pages by simply searching those web pages for instances of phrases matching the names of commands, menus, and other interface components. This approach works rather well for commands with technical names (e.g., "unsharp mask"), or for longer phrases ("Always check to see if Firefox is the default browser on startup"). Such phrases are unlikely to appear accidentally in documents. However, for short command names, or for commands with common names (e.g., "Delete", "Save"), there is considerable ambiguity, and it is difficult to decide if the document is referencing a command, or if the phrase is simply part of the document's prose. In this chapter, the QAP scoring function (equation 4.1) addresses the problem by exploiting the redundancy afforded by multiple relevant passages, all the while using inverse document frequency to reduce the impact of common phrases. In other words, the scoring function requires that a feature with a common name appear in many relevant passages in order to achieve a high score.

Appendix A explores more sophisticated means of identifying references to system features in tutorials and forum postings. Authors often specify the full paths of commands in their text. As an example, rather than simply writing "grayscale", many authors write "Image→Mode→Grayscale" when referring to GIMP's grayscale command. In the latter case, it is clear that the terms “Image”, “Mode” and “Grayscale” are references to interface components. Identifying these types of patterns increases the confidence that the use of a word actually refers to an element in the system. Likewise, tutorial authors often capitalize command names, and may style commands differently than surrounding text. Each of these informal conventions can be used as a machine learning feature for detecting commands mentioned in instructional material.

4.4.2 Exploiting temporal associations

Additionally, just as search queries are often task-related, so too are the documents they retrieve. As such, many documents specify sequences of commands that must be executed in a particular order to achieve a desired outcome. These command sequences are not reflected in the current QF-graph, nor in the search results returned by QAP. This can be disconcerting when the order of commands returned does not match the order in which those commands should be executed. As an example, consider the query “how to draw a circle in gimp”. Depending on the strengths of the query-feature associations, the system may return a ranked list of commands where “stroke selection” appears before “ellipse select” (where both commands must be used to draw a circle in GIMP since it provides no tools for drawing geometric primitives).

In the future, I would like to extract command sequences from documents, and use this sequencing information to improve the range of possible applications of QF-graphs. As an example, it would be beneficial if search-driven interaction could return sequences of actions rather than individual commands. Similarly, the availability of sequencing information could allow the recommendations made by dynamic tooltips to automatically update as the user progresses through a given task.

4.5 Conclusion

In this chapter, I presented QF-graphs, and demonstrated how they can be constructed automatically from logs of search queries, web pages, and localization data. While it is conceivable that QF-graphs can be constructed using alternative means or data sources, the approach outlined in this chapter confers a number of advantages. Specifically, by drawing from query logs and web pages, one ensures that QF-graphs graph can be continuously updated as system usage patterns change. Moreover, a completely automated approach ensures that data from thousands of users can be considered when associating queries with system features.

This chapter also outlined how QF-graphs can be used to advance search-driven interaction, while paving the way for new interaction techniques such as dynamic tooltips and application-to-application analogy search. Collectively, these mechanisms help to bridge the Gulf of Execution in cases where users are able to articulate their goals as search queries, but are unsure of how to accomplish those goals in an interactive system.

4.6 Addendum

In the time between the first publication of Query-feature graphs [49] in 2011, and the preparation of this dissertation in 2015, Query-feature graphs have been leveraged and extended by other researchers in a number of significant ways [72, 1]. First, Khan leveraged query-feature graphs as the foundation of a command recommendation system [72]. In developing their recommender system, Khan et al. undertook additional steps to: (1) cull queries that did not clearly specify a task (e.g., "gimp 2.6 fonts"), and (2) to resolve minor mismatches in the feature names used in the original research (e.g., differences between GIMP 2.6 and GIMP 2.8.6). This suggests that there is a need to consider how to address issues that arise when online documentation (and models trained thereon) diverge from the terminology expressed in a user interface, as a result of iteration and evolution of the software over time, or between localities. This issue is detailed and discussed in Chapter 6.

Additionally, in 2014, Eytan Adar et al. presented the CommandSpace system which is similar in spirt to Query-feature graphs, but leverages deep learning to jointly model system features and user vocabulary [1]. As with query-feature graphs, CommandSpace is trained on a corpus of web tutorials, and allows users to map from keyword searches to system commands and vice versa. Notably, Adar et al. compared the performance of CommandSpace to that of Query-feature graphs, and found that their approach leveraging deep learning significantly outperformed those generated using QAP – the method I employed to construct Query-feature graphs for this dissertation. To this end, I consider Adar et al.'s work to be the new state-of-the art in this space, and should serve as the starting-point going forward.

Chapter 5

InterTwine

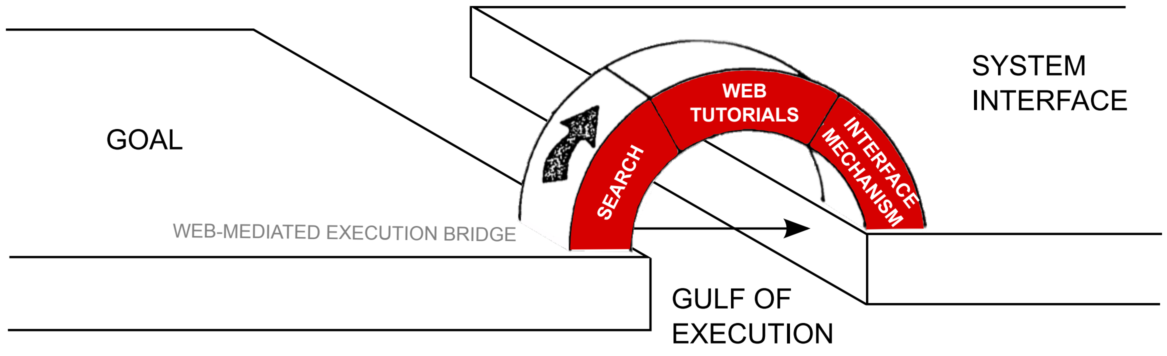
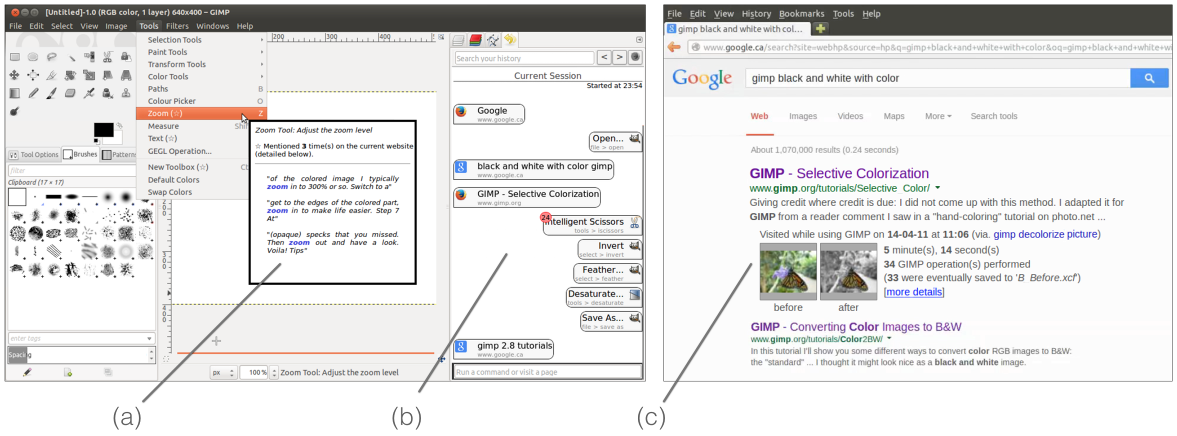


Figure 5.1: This chapter introduces InterTwine, a system that integrates all three steps across the web-mediated execution bridge. With InterTwine, actions in the web browser directly impact how information is presented in a software application, and vice versa. Portions of this chapter were first published in [46].

A central tenant of this dissertation is that people rely on web search and web documents to help bridge the Gulf of Execution when using feature-rich software. Users access web materials both as a first line of technical support, and as means for coping with the software's complexity [16, 77, 116]. This tightly coupled use of web-based information with desktop software suggests that it is worthwhile to consider these separate systems—the search engine, online documents, and the desktop application—as parts of a *single system* for performing work.

This chapter integrates all three steps across the web-mediated execution bridge to address the processes of finding and re-finding task-specific information when using desktop software. In this context, information foraging theory provides a useful framework for considering these practices [102]. Information foraging posits that people make use of *information scent* to guide their selection and use of information resources within



Zoom Tool: Adjust the zoom level

☆ Mentioned **3** time(s) on the current website (detailed below).

*"of the colored image I typically **zoom** in to 300% or so. Switch to a"*

*"get to the edges of the colored part, **zoom** in to make life easier. Step 7 At"*

*"(opaque) specks that you missed. Then **zoom** out and have a look. Voila! Tips"*

About 1,070,000 results (0.24 seconds)

GIMP - Selective Colorization
www.**gimp**.org/tutorials/Selective_Color/
Giving credit where credit is due: I did not come up with this method. I adapted it for **GIMP** from a reader comment I saw in a "hand-coloring" tutorial on photo.net ...

Visited while using GIMP on **14-04-11** at **11:06** (via. gimp decolorize picture)
5 minute(s), **14** second(s)
34 GIMP operation(s) performed
(**33** were eventually saved to '*B_Before.xcf*')
[more details]

GIMP - Converting **Color** Images to B&W
www.**gimp**.org/tutorials/**Color**2BW/
In this tutorial I'll show you some different ways to convert **color** RGB images to B&W: the "standard" ... I thought it might look nice as a **black and white** image.

Figure 5.2: InterTwine links web browsers with feature-rich software applications to create *interapplication information scent*. Specifically, InterTwine modifies application tooltips to describe how they are mentioned on the web page currently open in the browser (a). Likewise, InterTwine extends web search snippets with details of how work documents evolved when that web page was last accessed (c). Finally, InterTwine maintains an interactive interapplication event history to facilitate re-finding past actions and relevant pages (b).

"patches," or collections, of information [21]. When using the web, information scent is provided by elements such as a search engine's autocomplete service, the short page snippets shown in search results, or previously visited links rendered in a different color. In desktop software, menu hierarchies, command names, tool icons, and tooltips all provide affordances that can be considered forms of information scent that assist users in finding relevant functionality.

While these separate systems each provide useful forms of information scent to guide the pursuit of desired information, they largely function independently of one another: the activities in one application have no effect on the presentation of information in the other, forcing the user to manually link information patches between the two applications. For example, desktop software generally has no awareness of when the user turns to the web to learn how to complete a task with the software. Thus, when the user finds relevant information on the web, they must manually connect that information with the affordances and cues provided by the desktop software. This motivates the development of mechanisms that more effectively link these distinct systems, to ease the processes of finding and re-finding information within and across applications.

This chapter describes *InterTwine*, a system that introduces the concept of *interapplication information scent*. Interapplication information scent links the separate information patches of the web browser and desktop software, injecting novel forms of information scent into both applications to facilitate the finding and re-finding of information. For example, InterTwine embellishes a desktop application's menus with markers (what I call "beacons") for menu items described in the currently open, front-most browser window or tab (Figure 1a). The current implementation explores these concepts in the context of the Firefox web browser and the GNU Image Manipulation Program (GIMP).

InterTwine is composed of three conceptual entities: 1) a shared, interapplication history produced by recording actions in the web browser and desktop application, 2) mechanisms to identify information likely to be of relevance from this shared history, and 3) novel interface mechanisms that introduce context-aware, interapplication information scent synthesized from this shared history.

This system demonstrates three different classes of interapplication information scent: application bridges, history snippets, and history digests.

Application bridges are information scent cues that link information in the currently open web page with the relevant features of the desktop application. The previously mentioned menu beacons are an example of application bridges. InterTwine also bridges applications by injecting relevant snippets from the open web document into the tooltips of the desktop application. Together, these bridges help users establish and maintain a shared information context *across* application spaces, easing the process of locating information referenced in one application in the other application.

History snippets are a form of information scent that communicate the context surrounding the past use of a command or web page. InterTwine provides history snippets by embellishing tooltips with a paged display that lists the commands and web pages leading up to, and following, previous invocations of the command. This context helps people re-trace past steps at a glance, and re-access relevant web content.

History digests provide context-dependent summaries of how the desktop application was used with respect to a given web page. InterTwine presents these history digests by augmenting Google search results with summaries that include before and after screenshots of the related application document, and the commands invoked in the desktop application when that web page was open. Warnings are presented to the user for pages in which operations were performed but later abandoned (suggesting the page may not have been that useful for the task). This shared information scent helps users more effectively locate web pages previously found to be useful.

Collectively, this chapter makes the following contributions:

1. I present results from a formative study that informed the need for, and design of, mechanisms that provide interapplication information scent

2. I introduce the concepts of a shared, interapplication history and interapplication information scent, and demonstrate three types of interapplication information scent: application bridges, history snippets, and history digests

3. I demonstrate these concepts within InterTwine, a system that ties together the separate information spaces of Firefox and GIMP

4. I validate these concepts via the results obtained from the formative study

In the remainder of this chapter I present the details of the formative study, then describe the InterTwine prototype, and the types and instances of interapplication information scents it provides. I then describe a strategy for constructing interapplication history. Finally, I report how participants of the formative study and iterative design process responded to the final version of the software, and discuss future directions for research.

5.1 Formative Study

To guide development of InterTwine, I conducted a formative study to: 1) understand the breakdowns that occur when online resources are used to support work in feature-rich applications, and 2) collect feedback on early designs of InterTwine. Eleven individuals (six male, five female, mean age of 26), with varying levels of experience with image editing software, participated in this study. Participants received a $10 Amazon gift card as remuneration for participating in the formative study.

Each session was divided into two parts. In the first part of the session, participants were asked to perform a pair of tasks (described below) using unmodified versions of GIMP 2.8 and Firefox 26. This afforded an opportunity to observe, firsthand, how people find, follow, and re-find online materials when performing tasks in a feature-rich application.

In the second half of each session, participants performed the exact same tasks again, but using the experimental interface designs. These designs ranged from sketches to fully implemented prototypes. For non-functional prototypes, I explained how they functioned and asked users for feedback on their utility to complete the tasks they just performed. For functional prototypes, participants were asked to think aloud as they used the prototypes to complete the same tasks again. Whenever possible, the prototypes were populated with data generated in the first half of each session, thus giving participants a chance to evaluate designs with "live" data. Designs were iterated after each interview, to continually evolve the prototype.

Two tasks, whose solutions are non-trivial in GIMP, were chosen for study. One task was to place a thick black border around a sample of large text (i.e., outlining the text). The other task was to modify a color photo so that the background was black and white (i.e., selective desaturation). In all cases, tasks were presented as pairs of before and after pictures without any descriptive text. The order of presentation of tasks was counterbalanced across participants. Prior to performing the first task, participants were strongly advised to seek online materials for assistance.

Participants' actions were captured by event logging software, screen capture software, and an audio recording device.

5.1.1 Summary of Study Results and Implications for Design

The study found that participants initially experienced trouble locating commands mentioned in the web-based tutorials when switching back to GIMP. Participants also had difficulty recalling their previous actions, both online (e.g., search queries, as in [119]), as well as in the application (e.g., commands and procedures).

While I expected participants would encounter some of these issues, they occurred more frequently and with greater severity than expected. For example, I observed participants identify promising commands mentioned in tutorials, and then almost immediately forget the identity, location, and details of those commands when switching into the desktop application. This difficulty was observed even when tutorials explicitly and unambiguously mentioned the locations of commands in the interface. In all of these situations, participants typically adopted a strategy of systematically exploring an application's menus and tooltips in the hope of recognizing their target.

These results suggest there is value in linking the separate information spaces of the web browser and desktop application, to make finding information presented in one application easier to locate in the other application. The difficulty in recalling past actions also suggests the value of mechanisms that assist in re-finding task-specific information at a later time. These results directly led to the development of application bridges and history snippets in InterTwine.

The iterative design process also revealed that participants were enthusiastic about seeing previously edited GIMP work documents associated with relevant web pages in web search result snippets. However, they expressed disinterest when presented with fine-grained details of errors, dead-ends, or unsuccessful sequences. Accordingly, history digests summarizing past activities were developed. Digests included indications of which web pages appeared to have had no effect on advancing the solution.

Participants were also asked how they would feel if non-relevant web pages were wrongly associated with GIMP work documents (e.g., linking a work document to a news article that was coincidentally open in the web browser). Participants commented that such errors were easy to spot from the snippets provided, and could be safely ignored, but posited that a high frequency of errors would quickly eliminate any such system's advantages over the status quo. As such, InterTwine takes several measures to ensure the relevance of any associations made between web pages and GIMP documents.

Finally, when presented with a shared history depicting the use of both applications, the participants expressed a desire to see the full history, without omissions. Early designs revealed only the most important commands or web pages, and were generally disliked, as were designs that presented browsing and procedural details in separate locations. Accordingly, InterTwine includes a shared history that shows *all* activity in both applications.

I now describe InterTwine's design in detail.

5.2 InterTwine

InterTwine is composed of three conceptual parts: a shared interapplication history, mechanisms to identify potentially relevant information from that history, and interapplication information scent that helps users link the separate information spaces of the web browser and desktop application.

Individual parts were implemented via a plugin-in for the Firefox web browser, a modified version of the GIMP image manipulation system, and a shared datastore and associated shared history service that mediates communication between Firefox and GIMP.

While the shared history service serves a key role in the system, it operates automatically in the background, and is not directly accessed by the user; users access its capabilities through the various interface components described below.

InterTwine modifies GIMP by adding an *interapplication history transcript*, by embellishing its menu items with *beacons*, and by adding *history snippets* to tooltips. In Firefox, InterTwine augments Google search result pages with *history digests*. I describe each of these components in turn.

5.2.1 Interapplication History Transcript

The foundation of the InterTwine system is an interapplication interaction history. While these data are used to derive many of InterTwine's other features, users can also directly view and interact with this history.

The current implementation depicts this shared history in a pane that adopts the metaphor of a chat program. When an action is performed in an application, a "speech bubble" is produced representing that action (Figure 5.3). Entries contributed by GIMP appear on the right. Entries contributed by Firefox appear on the left. This metaphor was chosen to make it easier to visually parse the histories, and to establish a common visual design that can be used in other parts of the interface.

All items displayed in the transcript are interactive. Clicking on a GIMP command bubble causes the command to be invoked, and clicking on a web page bubble causes the web page to open in Firefox and the web browser to come to the foreground.

The transcript can also be searched. InterTwine's transcripts are indexed by command names, command tooltips, file names, web page titles, internet search queries, web page body text, dates, and times. This extensive coverage is designed to allow users to index into the transcript using almost any detail recalled about a previous session, and addresses the goal of helping users re-find information.

Finally, embracing the notion of a chat system, users can directly add to this transcript by typing at the chat prompt (Figure 5.4). When users begin typing at the prompt, their input is automatically completed with the names of GIMP commands, as well as the titles of web pages in the browsing history. Selecting an item from the list of suggestions causes

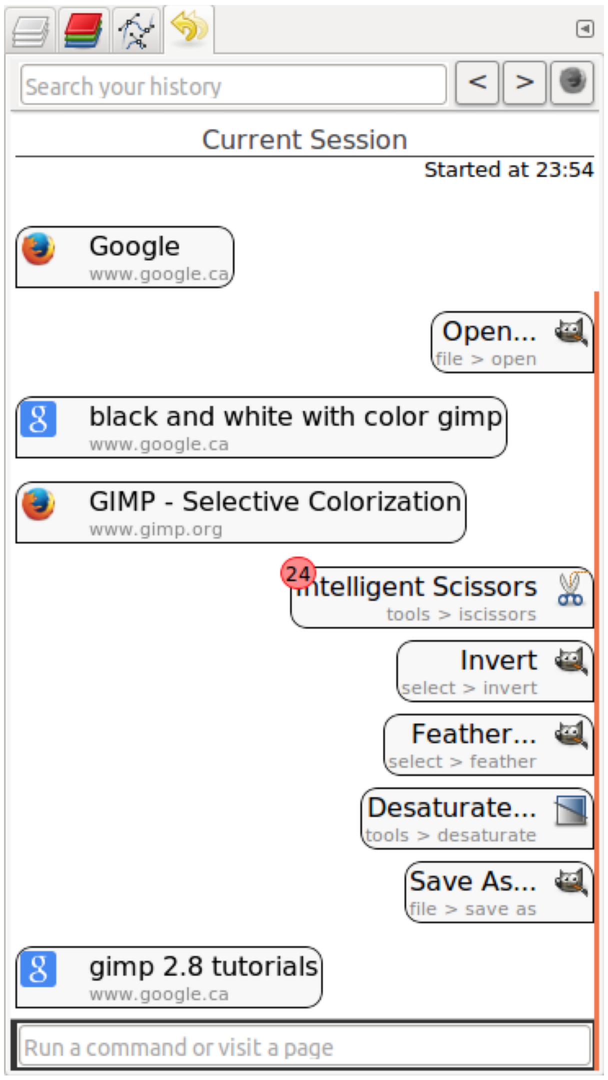
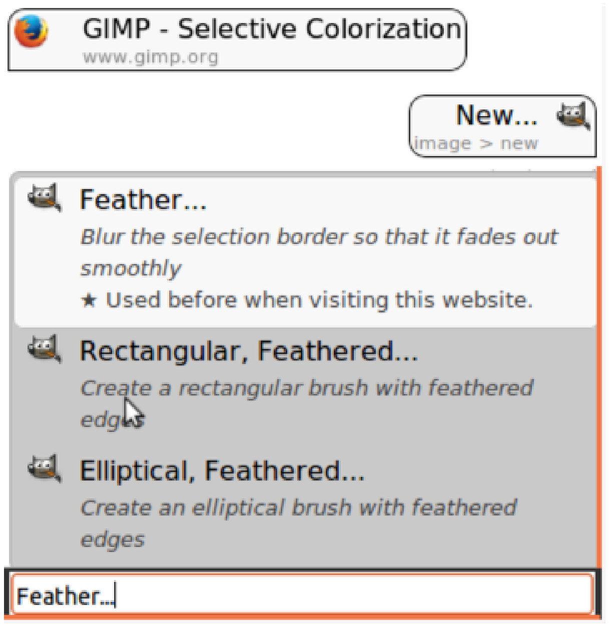


Figure 5.3: InterTwine adopts the metaphor of a chat program to communicate interapplication event history. When users issue commands in GIMP, the commands appear as speech bubbles on the right-hand side. When users visit pages in Firefox, the pages appear as speech bubbles on the left-hand size.

the command to be performed, or the website to be opened and brought to the foreground. This capability draws some inspiration from Hendy et al.'s graphical enhanced keyboard accelerators [62].

As with all InterTwine features, autocompletion makes extensive use of the inter-application history and the current state of both applications when determining what suggestions to provide, as well as their order. As an example, if a web page is open in the user's browser, then the suggested commands are embellished with one style of beacon (i.e., marker) if they are mentioned in that web page, and another style of beacon if they were previously used the last time that web page was open (as seen in Figure 5.4).


GIMP - Selective Colorization
www.gimp.org

New...
image > new

Feather...
Blur the selection border so that it fades out smoothly
★ Used before when visiting this website.

Rectangular, Feathered...
Create a rectangular brush with feathered edges

Elliptical, Feathered...
Create an elliptical brush with feathered edges

Feather...

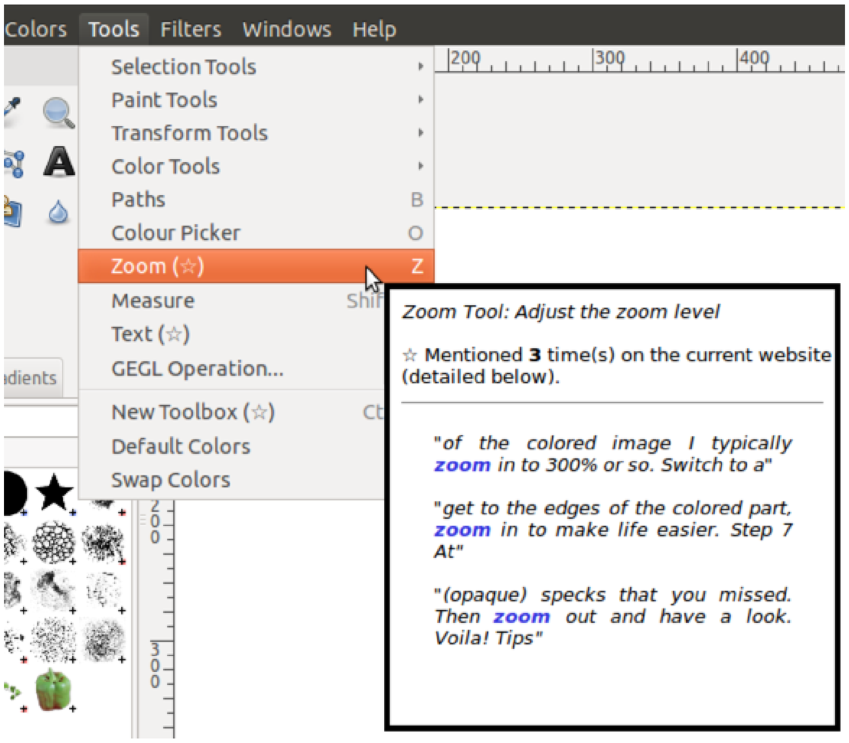
Figure 5.4: A "chat prompt" is located directly below InterTwine's events transcript. From this prompt, users can execute GIMP operations or visit web pages by typing commands. The prompt autocompletes the input, allowing users to issue commands with only a few key presses. Auto-completion suggestions are prioritized and contextualized based on the current context. Context cues include: the page the user is visiting, the document the user is editing, and the set of commands the user has recently issued.

5.2.2 Interapplication Information Scent in Menus and Tooltips

InterTwine modifies the presentation of GIMP menus and tooltips to provide interapplication information scent. This information scent is informed by the shared history and the currently visible web page.

InterTwine embellishes menu items with a hollow star icon (a beacon), to communicate that a given menu item is mentioned on the web page the user currently has open in the browser (Figure 5.5). These menu items' tooltips are also augmented to present web page excerpts that mention the given menu item. These beacons and excerpts are designed to increase the information scent of relevant menu items, and help bridge the separate information spaces of the web page and the desktop software.

InterTwine's menu items are further enhanced based on their history of use. Specifically, if users have previously visited a web page and issued a command with that page open, then InterTwine displays a filled star icon next to the menu item to indicate past relevance. Furthermore, the item's tooltip contains excerpts of the interapplication history transcript detailing its context of use (Figure 5.6). In cases where the menu item has been used in multiple contexts, users can page through a slideshow of these excerpts using the left and right arrow keys. Finally, at any time, users can press the *F1* key to



Zoom Tool: Adjust the zoom level

☆ Mentioned **3** time(s) on the current website (detailed below).

"of the colored image I typically **zoom** in to 300% or so. Switch to a"

"get to the edges of the colored part, **zoom** in to make life easier. Step 7 At"

"(opaque) specks that you missed. Then **zoom** out and have a look. Voila! Tips"

Figure 5.5: InterTwine embellishes menu items with beacons (star icons) when those items are mentioned by name in the currently visible web page. Likewise, menu tooltips gain snippets describing the context in which each menu item is mentioned in the web page.

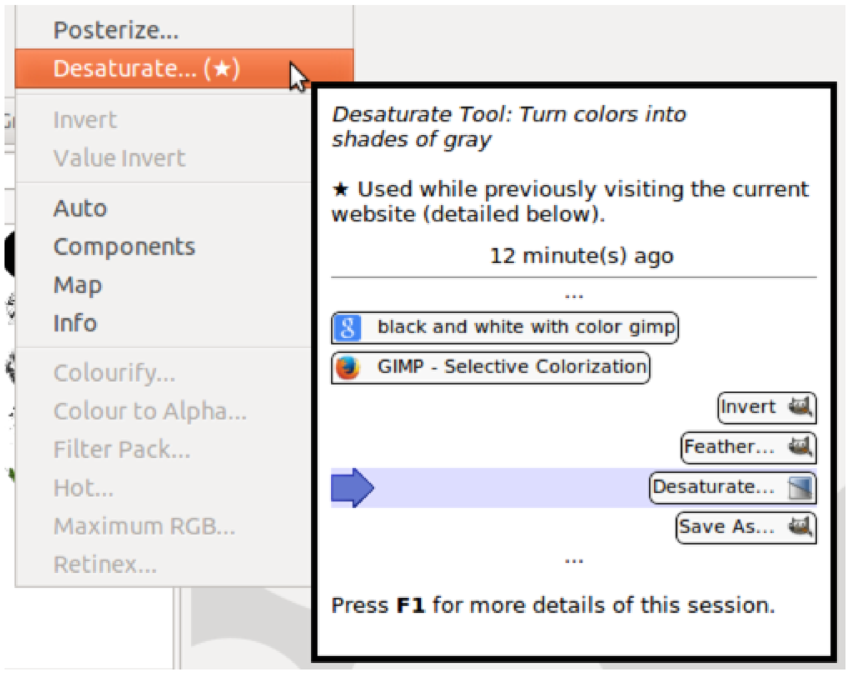
scroll the full interapplication history (described above) to the corresponding time when the command was used.

5.2.3 Interapplication Information Scent in Web Search Results

In Firefox, InterTwine modifies the Google search results page by enriching the standard search result snippets with details extracted from the interapplication history.

In situations where a search result item has been previously visited, InterTwine retrieves details of the past visit, and generates a summary (or digest) of this information for review (Figure 5.7). These digests include two screenshots of GIMP's canvas: one taken when the user first accessed the search result, the other taken when leaving the target web page. These images serve as a visual summary of the work that was done when previously visiting the page.

The snippet also details how long the page was previously open, as well as the number of GIMP operations performed while the page was focused in the browser. Two different counts are presented: the total number of operations performed, and the number of operations ultimately saved to the work document. These counts differ when commands are undone, or when the user closes a document without saving or exporting a result. In



Desaturate Tool: Turn colors into shades of gray

★ Used while previously visiting the current website (detailed below).

12 minute(s) ago

...

...

Press **F1** for more details of this session.

Figure 5.6: When revisiting a web page, InterTwine places additional beacons (filled-stars) next to commands that were used before in the context of the web document. In these cases, the tooltips gain excerpts from the interapplication history, describing their earlier context of use.

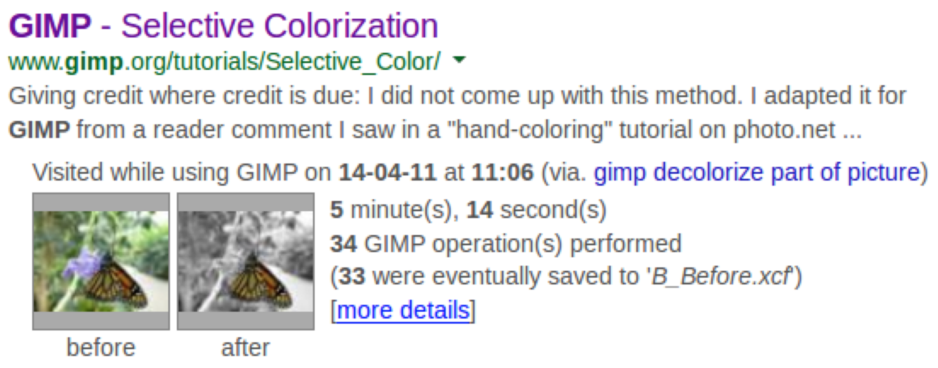
extreme cases where all commands are abandoned, the snippets instead present a warning (Figure 5.8).

Finally, users can optionally click a hyperlink in the snippet to reveal the context in which the page was previously accessed. As with tooltips, these details are presented as excerpts from the interapplication history.

5.3 Implementation

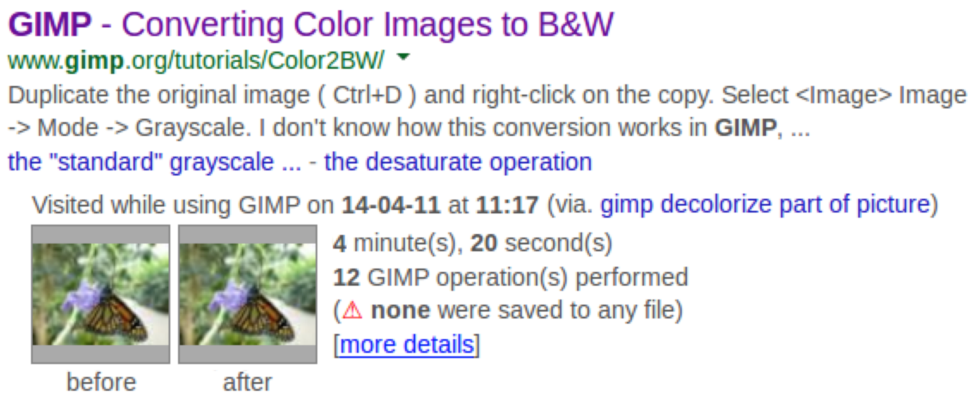
InterTwine's implementation consists of three components that interoperate on a user's local machine: a modified version of the GIMP image manipulation system, a plugin-in for the Firefox web browser, and a local coordination service that mediates communication between Firefox and GIMP. I describe each of these in turn.

InterTwine requires two distinct sets of modifications to GIMP. First, GIMP must be instrumented to record a user's low-level interactions with the software, as well as to record screenshots of the user's work document as it evolves over time. GIMP must also be modified to display the interapplication history transcript, as well as custom menu items and tooltips. InterTwine implements the transcript and the custom tooltips by embedding a Webkit browser directly into the GIMP application. As such, InterTwine's



GIMP - Selective Colorization
www.gimp.org/tutorials/Selective_Color/
Giving credit where credit is due: I did not come up with this method. I adapted it for GIMP from a reader comment I saw in a "hand-coloring" tutorial on photo.net ...
Visited while using GIMP on **14-04-11** at **11:06** (via. gimp decolorize part of picture)
5 minute(s), **14** second(s)
34 GIMP operation(s) performed
(**33** were eventually saved to '*B_Before.xcf*')
[more details]

Figure 5.7: InterTwine modifies Google search result snippets when web pages have been previously visited. Here, InterTwine presents history summaries, which include screenshots of how the user's document evolved when previously reading the page. Additionally, the summaries present statistics describing the time spent, and the number of commands issued.



GIMP - Converting Color Images to B&W
www.gimp.org/tutorials/Color2BW/
Duplicate the original image (Ctrl+D) and right-click on the copy. Select <Image> Image -> Mode -> Grayscale. I don't know how this conversion works in GIMP, ...
the "standard" grayscale ... - the desaturate operation
Visited while using GIMP on **14-04-11** at **11:17** (via. gimp decolorize part of picture)
4 minute(s), **20** second(s)
12 GIMP operation(s) performed
(⚠ **none** were saved to any file)
[more details]

Figure 5.8: InterTwine's history summaries present warnings to users in cases where earlier page visits failed to result in commands being invoked, or in cases where command invocations were ultimately abandoned by the user (e.g., commands that were undone or unsaved).

tooltips and transcripts are themselves implemented in HTML and JavaScript. This architecture was especially useful during the formative study, allowing the quick iteration of InterTwine's designs.

As with the modifications to GIMP, InterTwine's Firefox plug-in serves two roles. First, it instruments the browser to track a user's actions online. Second, it modifies the presentation of the interface (i.e., Google search results). In both cases, these actions are achieved by injecting custom JavaScript code into the pages a user visits online. Instrumentation is achieved by coercing visited web pages to signal interface events (e.g., page loads, scrolling, etc.) by making an asynchronous request to InterTwine's local service.

Finally, InterTwine's local coordination service consists of a lightweight web server running in the background of the user's machine. This service both collects instrumentation data from GIMP and Firefox, and generates content for the GIMP transcript and tooltips, as well as the Google search result snippets. Additionally, this service processes the instrumentation data to generate the interapplication history. Next, I describe a method of generating and refining the interapplication history.

5.3.1 Creating and Refining Shared Histories

In the formative study, users made extensive use of tabbed browsing, and followed hyperlinks by opening each in a new tab. Once opened, tabs were rarely closed before the end of each session. Instead, participants returned to earlier pages by switching tabs rather than by relying on the browser's backward and forward buttons.

Two users were also observed adopting a strategy of "pre-fetching" search results by opening promising links in new tabs prior to visiting any individual result. In these cases, it was common to retrieve pages that were never actually consulted.

Finally, when participants leveraged online resources to perform unfamiliar tasks, they often continued to explore and experiment with the application's interface, sampling the application's capabilities and undoing many commands.

All of these practices complicate the process of producing a meaningful, shared interapplication history: By the time users complete a task, the histories and interactions logs are extremely noisy, with relevant commands and web pages hidden in a sea of dead-ends and failed experiments. Accordingly, further processing and filtering of the shared interapplication history was found to be necessary.

In an effort to improve upon the naïve approach to creating a shared history, InterTwine gathers additional interaction details not present in standard browsing histories. In particular, InterTwine records which browser tabs are visible at any given moment, providing an indication of relevance for each tab. Likewise, InterTwine records interaction events that occur on web pages (e.g., page scroll events), and uses these events to estimate the degree to which a page is being utilized. Specifically, application commands are attributed to web pages only if the web pages mention the name of the application ("GIMP") in their body text, and if the application commands occur within 5 minutes of a web page interaction event.

Additionally, InterTwine tracks the outcome of each command, noting if the command is ever undone, and if not, whether the application document is ever saved (indicating that the work in that document was deemed useful). These data are then reflected in the history digests shown with Google search results: digests display whether a visited web page previously contributed any commands that were still intact when the work document was saved.

5.4 Feedback on Final Design

To evaluate the final design of InterTwine, participants from the initial formative study were invited to return 15 days after their initial session. In addition to collecting feedback on how the design had improved (or regressed), this final session afforded an opportunity to observe breakdowns that occur when people repeat tasks several weeks apart, providing a way to validate the system's core purpose (i.e., to assist in re-finding information). Five participants responded to the invitation, and returning participants received an additional $10 Amazon gift card as remuneration.

In this return session, participants were asked to perform one of the original tasks. Since each participant experienced a different prototype in the formative study, and because early prototypes were not always functional, it was not possible to carry forward a participant's earlier interapplication history. Instead, participants were greeted with a fresh installation of GIMP, Firefox, and InterTwine. As with the formative study, minimal instructions were provided. Tasks were presented as pairs of before and after pictures without descriptive text, and participants were advised to seek online materials. InterTwine's features were discussed as they were discovered, and participants were asked for their interpretations before correct use was demonstrated.

5.4.1 Results

At the onset of this final session, all five participants felt they remembered enough about the task to complete it without consulting the web. However, when the time came to actually perform the task, none were able to complete the task without consulting web resources. When asked about this discrepancy, one participant explained:

> *"I think my dependence on the Internet is pretty high. I'm sure I could have [completed the task], but sometimes I doubt myself and think I'll do this faster with Google"* **P2**

After participants visited a few web pages, I called attention to InterTwine's interapplication history, and provided a brief overview of InterTwine's other features. Participants were then asked to continue the task, and to provide feedback as they worked.

Participants were generally positive about InterTwine's individual features. For example, P1, P3 and P5 were very enthusiastic about InterTwine's menu and tooltip enhancements. P3 had the following to say about these features:

> *"You don't have to guess anymore. When you read something and you think yeah, OK, and [then]... I mean, I was powering through this pretty quick, because I thought I knew how to do it. But if you are on a web page and you*

go up there and... [in] the two seconds you actually read something versus going to do it, you forget what's going on, and then boom, it's right there [referring to InterTwine's beacons] " **P3**

Likewise, participants P2, P3 and P4 explicitly mentioned finding the shared history to be a useful tool. P2 reported this feature as the single most significant improvement from the prototypes discussed in the formative design process. P3 was especially enthusiastic about the shared history's command prompt, stating:

"I'm just thinking like for an advanced user, someone who has been using it for a while, just having the quick keys [command prompt] down here rather than going through all that, that's kind of nice... Yeah, so once you get handy with that I can see it being really, really powerful." **P3**

All five participants felt that the historical digests, added to Google search results, were especially useful. This was not unexpected, as 10 of the 11 original formative design participants responded enthusiastically to prototypes with this feature. On this topic, P5 noted:

"I like where it says how long you've been on the web page and what [you] did. It will tell me I saw this web page quickly and I didn't like it so I X'ed out of there." **P5**

Finally, P2 noted that InterTwine's features were compelling enough to switch image editors:

"My image editor of choice is Paint because it's really simple... I do have Paint.NET on my computer, but I don't use it. But with this sort of input, I would be more likely to use GIMP, because it would help me do some more advanced things that would be difficult to figure out otherwise." **P2**

5.4.2 Areas for Improvement

While participant feedback was very positive, participants offered suggestions for improvements. For example, despite being very enthusiastic about the menu beacons, P3 initially failed to notice them at all. When the beacons were pointed out by the researcher, the participant explained that the beacons looked like menu text, and he was not sufficiently familiar with GIMP to know if they were something new. This suggests the need to refine the presentation of these markers, so that their designs better communicate their purpose.

Likewise, P5 noted that she found the features of InterTwine's shared history (e.g., search and auto-complete) to be complex, and worried that she would not "use it to its full potential." However, P5 felt that the excerpts of the history, presented in the tooltips and search results page, were acceptable.

Finally, both P1 and P2 requested a mechanism for manually managing the history. In particular, they expressed interest in the ability to rate pages and command sequences, as well as the ability to hide or delete items with poor ratings. I feel that these are excellent feature requests, and that manual management of interapplication history is a compelling design dimension to explore in the future.

A final encouraging sign that InterTwine is offering features that are generalizable, and of value, is that participants offered numerous suggestions for other applications that could benefit from the same interactions. For instance, P5 suggested InterTwine could be applied to SPSS, Minitab, and other statistics packages to help users recall how to perform various statistical tests. Likewise, P1 commented that an InterTwine-like system would be useful when using geographic information systems. Finally, P2 stated:

> *"I'm just thinking like, even writing an essay, like if you had your documents up like in Mendeley or whatever, you can know you wrote this with this document open, and you know what to cite. I'm just thinking of other applications. It seems... I'm just astounded, this is really cool."* **P2**

5.5 Discussion & Future Work

InterTwine represents one targeted exploration of the notion of interapplication information scent. The types of interaction mechanisms chosen in this research were largely determined by following the most salient leads uncovered in the formative study. In this section, I discuss other promising research trajectories, and other possible application domains.

5.5.1 Community Aggregation

InterTwine operates entirely on one's own personal computer, leveraging only personal browsing history and interaction logs. One noteworthy finding from the formative study was that participants were almost entirely unconcerned with sharing their interaction details with a central service like Google (10 of 11 participants had few or no concerns). Given this, one could imagine aggregating a community's interapplication histories, enabling a number of additional applications.

One compelling use of this aggregated data would be to improve indexing of web-based tutorials. For example, a search engine could use this information to influence the ranking of search results, steering users away from tutorials that frequently result in abandoned commands. Likewise, since interapplication history does not depend on page text, these data could be used to index non-textual tutorials such as screencasts and video demonstrations. For example, a search engine could index tutorial videos using the names of the application commands and tools that users typically invoke while visiting each video. Users could then search for video demonstrations by naming commands of interest. With additional instrumentation of the web browser, video timestamps could be extracted and synchronized with application command invocations, allowing search engines to index *into* videos, and enabling capabilities similar to those described in [73, 81, 77]. Beyond indexing web documents and videos, aggregate data could also be applied to command recommendation systems (e.g., [90]), and related projects (e.g.,[48, 89]).

5.5.2 Interapplication Information Scent in Other Domains

InterTwine creates interapplication information scent between a web browser and a feature-rich raster graphics application. Other application pairings are possible—including ones that do not involve a web browser. As an example, a video editing application might notice coordinated use of audio editing software, and could adapt by highlighting the menu commands necessary to insert a new audio track into the video. Likewise, an operating system's file browser might visually modify folder icons in cases where the user has a terminal open in those directories, presenting a persistent visual trail as the user navigates the command shell through the file system. Finally, after using graphical user interface design software, such as QT Creator, programming environments could highlight lines of code related to objects recently worked with in the interface designer.

5.5.3 Fading Information Scent

In the current implementation, interapplication information scent is generated from the past history and current context of both applications. These information scents persist as long as the relevant context is present. However, it is possible that this context (especially from the shared history) could result in too much information scent accumulating, reducing the effectiveness of the concept.

To deal with this problem, interapplication information scent could fade out over time, similar in spirit to Baudisch et. al's [11] phosphor effects in the interface. For example, menu beacons could begin to fade after a web page has been open for 20 minutes. In general, I have not considered the possibility of incorporating hysteresis into the relevance models, but this capability may be especially useful for some of the mechanisms proposed above (such as the trails left through the file system as the user browses directories).

5.6 Conclusion

Online resources play an integral role in people's strategies for dealing with the complexities of feature-rich applications. However, applications and web browsers currently function as separate isolated entities, unaware of how activities in one application may relate to activities in the other.

In this chapter, I introduced InterTwine, a system that bridges these two application domains through the constructs of *interapplication history* and *interapplication information scent*. I demonstrated three classes of tools to provide interapplication information scent: application bridges, history snippets, and history digests. Together, these mechanisms help users find and re-find task-relevant commands and resources.

A formative study spanning two sessions reveals that InterTwine's features resonate with users and suggest their overall utility. Though InterTwine's current implementation is tied to GIMP and the Firefox web browser, I believe the ideas presented in this chapter can be generalized to other feature-rich applications. This sentiment is shared by many of those who took part in the participatory design process, as evidenced by their many enthusiastic suggestions for which applications to work on next.

Chapter 6

Challenges and Limitations

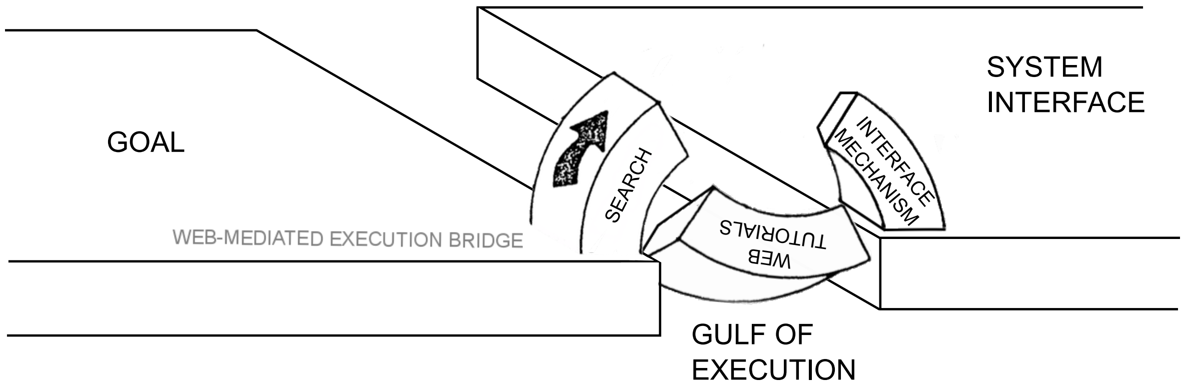


Figure 6.1: In this chapter I explore some of the meta-issues and challenges that arise when conducting the types of research championed in this dissertation.

The preceding chapters presented details of CUTS, Query-feature graphs, and InterTwine – three projects aimed at studying and supporting users as they leverage online resources to bridge the Gulf of Execution. These projects and approaches are subject to their own unique considerations and limitations, many of which have been discussed in their respective chapters. In this chapter I discuss some of the meta-issues which impact this style of research in general.

6.1 Model Cannibalization

An ideal outcome for this dissertation would be for the ideas presented therein to influence or to be incorporated into the software systems that people engage with on a daily basis. However, widespread adoption (e.g., commercialization) of these ideas introduces new complications. In particular, services and interactions described in this dissertation have the potential to change how users employ web search, or how people otherwise engage with online materials. For example, Chapter 4 describes how query data, together with a corpus of tutorials, can be used to enhance the tooltips within a feature-rich application. In the ideal case, these tooltips would address many common questions posed by users, thus diminishing the user's need to search online for this information. Unfortunately, query-feature graphs, the models on which these tooltips are based, depend on users issuing such queries in the first place. Addressing this model cannibalization problem, where use of a model alters the behaviours on which the model is based (thus diminishing its value), remains a topic for future work. Paths forward likely require investigating methods for adapting existing models in response to changing behaviours, and this likely requires relying more heavily on the types of behavioural data produced by systems like InterTwine.

6.2 Impact of Media Coverage

In addition to model cannibalization, external factors can also diminish the long-term representativeness of models derived in part, or in whole, from query data. In particular, issues arising from media coverage are well-known to researchers who leverage query data to model real-world phenomena [26, 20]. As an example, the model used by Google Flu Trends over-predicted, by 200%, the incidence of influenza during the 2012-2013 North American flu season. This error was ultimately attributed to increased media coverage of a notably bad flu season [20].

In the context of feature-rich interactive systems, media coverage of a system's features, bugs, or vulnerabilities, can create challenges to the systems and techniques proposed in this dissertation. As an example, Apple's iPhone 4 suffered from a design defect in which holding the phone in a particular manner caused a loss of reception (as reported in Chapter 3). This defect was widely reported in the media, increasing public awareness to the problem. This is illustrated by the Google Trends tool [35], which can be used to visualize changes in interest over time for such queries such as [*iphone antenna*] (Figure 6.2). As is illustrated in Figure 6.2, this increased level of interest in the iPhone's antenna is represented by a large spike in query volume at a time corresponding to the iPhone 4's release in June 2010. However, a similar spike in interest in antennas is observed for other smartphones in circulation at that time. For example, Figure 6.3 characterizes interest in the query [*droid antenna*], in reference to the Motorola Droid line of smartphones. Since the Motorola phones did not suffer from the same antenna issue as the iPhone, and because Motorola user are unlikely to have experienced the antenna problems first-hand, this increase in search traffic is likely a spillover effect from media coverage of the iPhone's defect. Likewise, Figure 6.2 reveals that the level of interest in the iPhone's antenna has remained elevated, compared to pre-2010 levels, despite significant improvements to antenna design that have occurred since the iPhone 4's release. This example serves to illustrate that media coverage directly impacts the search behaviours of a population. Adapting models and techniques to be robust against such media-induced spikes in interest is an active area of research [27].

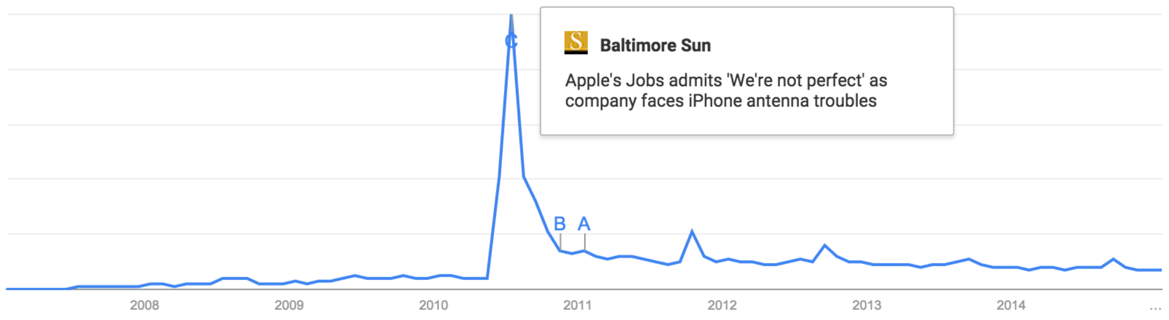


Figure 6.2: Query volume over time for the phrase "iphone antenna". A clear spike is visible midway between 2010 and 2011. This corresponds to media coverage detailing the antenna problems encountered by users of the iPhone 4. Since 2010, interest in iPhone antennas has remained elevated compared to pre-2010 levels.

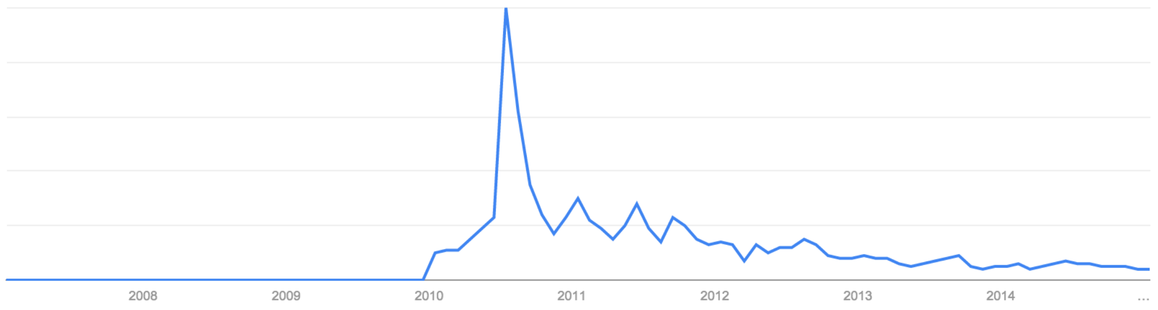


Figure 6.3: Query volume over time for the phrase "droid antenna" (in reference to Motorola's Droid line of smartphones). As with the iPhone, interest in the Droid's antenna spikes in June 2010. However, unlike the iPhone, Motorola Droid devices did not suffer from systematic failures of their antennas. As such, this increased level of search activity is likely a side effect of the media coverage discussing the iPhone's antenna problems.

6.3 Video & Pictorial Tutorials

The challenges discussed above impact the availability and the representativeness of query data. There also exist challenges in the continued use of web tutorial content to contextualize user behavioural data. Namely, the increasing prevalence and popularity of online video tutorials poses additional challenges to the techniques described in this dissertation, which depend on the ability to parse and process tutorials as *written* text.

Video tutorials, or tutorials relying on images to communicate important processes, are effectively opaque to text-based analyses. One path forward is to more strongly rely on behavioural data (e.g., via application instrumentation). For example, one may not need to parse a video tutorial if it is known what commands users issue while watching such videos. I elaborate on this possibility in the next chapter, which discusses future work.

Beyond leveraging behavioural data, existing research has investigated the feasibility of extracting task-relevant information directly from video content. For example, Juho Kim et al. demonstrated the possibility of using crowdsourcing platforms such as Mechanical Turk to extract step-by-step instructions detailed in a broad range of video tutorials [73]. Likewise, Pongnumkul et al.'s Pause-and-Play system leverages template matching to recognize tool activations in tutorial videos depicting use of the Photoshop raster graphics editor [104]. Finally, Dixon and Fogarty's Prefab system [41] could conceivably be leveraged for a similar purpose, or to detect other forms of interaction with user interface widgets. Adapting the work described in this dissertation to leverage these approaches remains a topic for future work.

6.4 Issues of privacy

Finally, any research involving the use of query logs or other behavioural data is subject to concerns about data privacy. With the exception of Chapter 5, which involved a laboratory study and informed consent, the work presented in this dissertation leverages data available on the public Internet. Namely, Chapter 3 leverages query auto-completion suggestions, and Chapter 4 pairs these data with the texts of online tutorials. I do not anticipate any threats to privacy arising from the work presented in this document.

That being said, many of the proposals for future work, here and in the next chapter, involve the bulk collection of user behaviour data. If one moves in this direction, one must take additional steps to ensure that they are acting as good stewards of this potentially sensitive information. I believe InterTwine serves as a useful model for how to proceed. In InterTwine, the recording of online behaviour is supported through a value proposition, and by adhering to certain guaranteed limitations on the data collection. Regarding the value proposition, InterTwine provides useful tools and services (e.g., menu beacons, and expanded search snippets) in exchange for the collection of certain types of behavioural data. I believe such exchanges of consideration are an important aspect of these classes of systems. Likewise, InterTwine is careful to gather behavioral data only when it is relevant to the services being provided. For example, browsing events are logged to InterTwine's shared history only when the visited websites mention the word "gimp" in their document text. With these safeguards in place, and with full disclosure of the types and uses of data being collected, I believe it is acceptable to move forward with larger scale deployments of the ideas proposed in this dissertation.

6.5 Conclusion

In this chapter I discussed some of the limitations and considerations that help to contextualize the work presented in this dissertation, and I discussed some of the challenges one might face when moving this research forward. However, I believe that meeting these challenges is a worthy venture: This dissertation has already enumerated many of the advantages of integrating the steps across the web-mediated execution bridge, and I believe there remain a large number of compelling opportunities to be explored. I discuss these opportunities in the next chapter.

Chapter 7

Opportunities and Future Directions

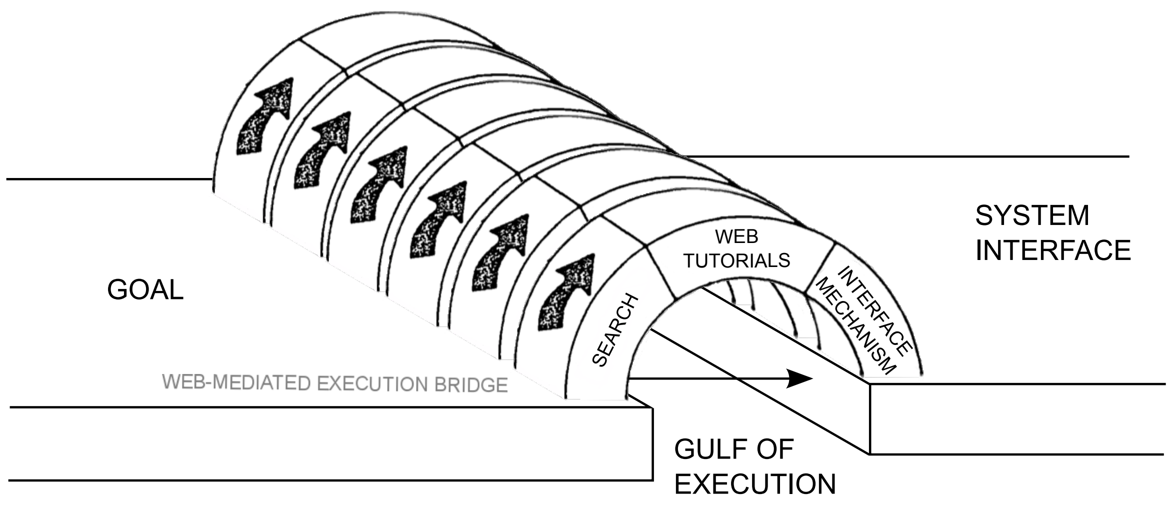


Figure 7.1: This chapter explores potential avenues of future work, with a particular focus on the opportunities afforded from the aggregation and integration, over many users, of fine-grained behavioural data across all three steps of the web-mediated execution bridge.

The previous chapter explored the challenges and limitations of the techniques proposed in this dissertation. However, advancing this research affords profound opportunities that cannot be overlooked. The web-mediated execution bridge establishes an environment where users express their tasks and goals directly to a computer system, in their own words, but in ways that can be reasoned about in software. In this chapter, I discuss some of the opportunities that arise from this direct line of communication between users and computer systems. I begin by exploring the possibilities that emerge when these types of integrations are deployed at larger scales. I then contemplate a world where web search serves an even larger role in our interactions with feature-rich technology. Finally, I discuss how the techniques proposed in this dissertation might be applied in broader contexts beyond the use of feature-rich software.

7.1 Large-scale Integrations and Deployments

Ultimately, the original research presented in this dissertation concluded with InterTwine, a system which is privileged in that it can observe users' complete journeys across the Gulf of Execution. However, a chief limitation of InterTwine is that, unlike the work presented in Chapters 3 and 4, the data available to InterTwine is local to the user's machine, and is limited to considering the actions previously performed by a *single* user. In this section I explore possibilities that emerge when these types of integrations are deployed at larger scales. I begin by discussing the feasibility of these types of large-scale deployments and data collection efforts.

7.1.1 Feasibility of Large-Scale Deployments

In this section I argue that the time of this writing is an ideal moment in history for pursuing the ideas presented in this dissertation on a broad scale. Integrations between web search, web documents and feature-rich applications are now more straightforward than ever before. Consider, for example, that Microsoft maintains a search engine (i.e., Bing.com), produces a web browser (i.e., Internet Explore), hosts instructional content (e.g., [38, 39]), produces a range of feature-rich software applications (e.g., Microsoft Office), and thus is well-positioned to integrate all steps across the web-mediated execution bridge on a massive scale. Similar arguments can be made about Google, or about Adobe Systems[1]. As noted in Chapter 2, Microsoft has already begun to move in this direction, having directly integrated Bing.com web search into their operating system and their suite of online office products (Figure 7.2). Now is the time to consider how best to leverage the opportunities afforded by these large-scale integrations.

7.1.2 Improving InterTwine

One of the chief limitations of InterTwine is that the system's interaction histories are locally maintained by a user's computer, and thus are limited to describing the actions of an individual. A direct consequence of this limitation is that several InterTwine features can operate only when the user revisits a document. InterTwine's enhanced search snippets, for example, are subject to this limitation. Deploying InterTwine-like integrations at web scale would allow search engines to bootstrap the creation of these snippets, deriving the necessary details from records of how the documents were previously used by other members of the community in the past.

[1]While Adobe does not maintain a search engine or a web browser, it does produce a set of extremely popular web browser plugins (e.g, Flash) that, like InterTwine, could enable such integrations.

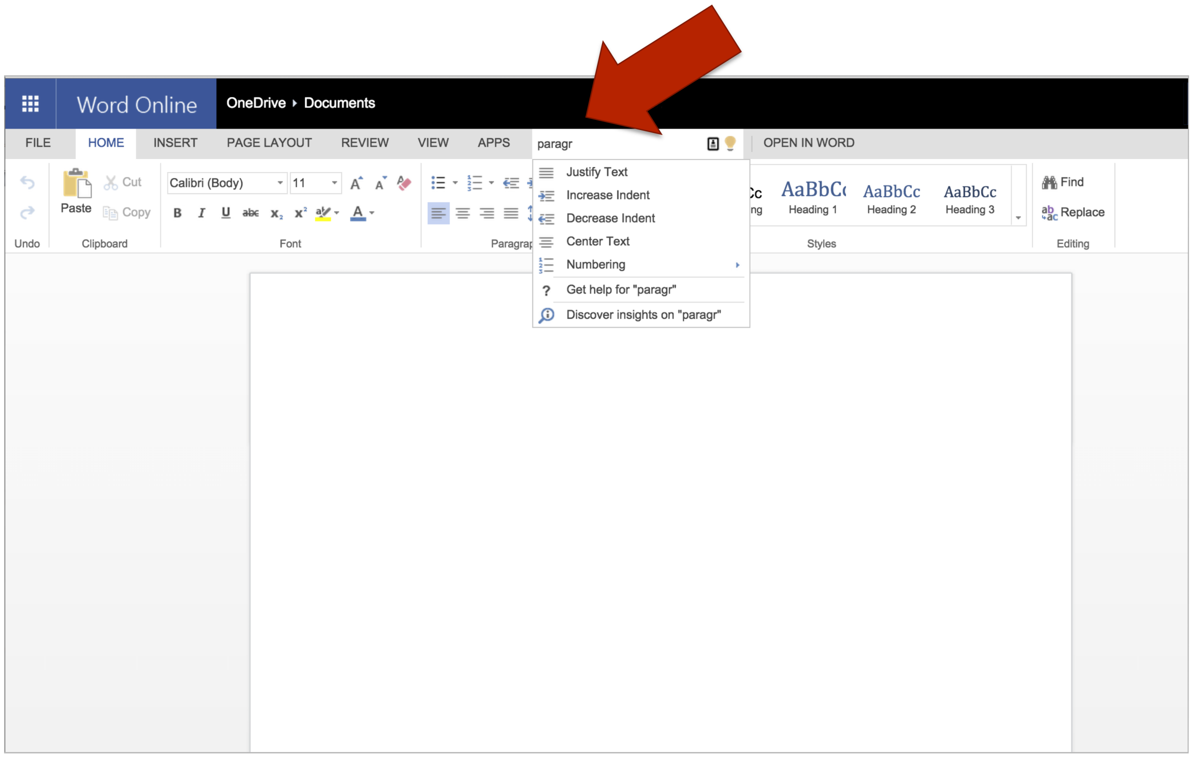


Figure 7.2: Screenshot of the Microsoft Word Online user interface [37] – a web-based implementation of Microsoft's popular word processing software. In late 2014, Microsoft integrated Bing.com web search directly into the application's menuing system. This integration is both well-aligned, and entirely compatible, with the work described in this dissertation (much of which was originally published in 2011-2012).

7.1.3 Implicit Feedback for Ranking Tutorials

In a similar vein to bootstrapping the creation of the enhanced search snippets, web-scale deployment of InterTwine-like systems could enable new forms of implicit feedback [3] for improving the ranking of search results. For example, if a particular tutorial is frequently associated with many "*Undo*" operations, or instances where work is abandoned without saving, then a search provider might consider penalizing this document in future search results. Here, it is important to contextualize user behaviour with the search queries users are issuing. For example, a tutorial describing how to convert an image's color mode to grayscale is likely to be of value to those searching [*how to make a picture black and white*], but would provide misleading [2] information to those seeking instructions on achieving selective desaturation – a popular post-processing effect often described as [*black and white with one color*]. Thus, the observation of frequent "*Undo*" operations associated with the latter search query should not adversely effect the document's ranking for the former query.

[2]Grayscale images lack the color channels necessary to produce the desired effect.

Timestamp	Menu Label	Command Tooltip
01:13:49	Open...	Open an image file
01:13:56	Other (75%)	Set a custom zoom factor
01:14:26	Intelligent Scissors	Select shapes using intelligent edge-fitting
01:14:50	Invert	Invert the selection
01:14:56	Feather...	Blur the selection border so that it fades out smoothly
01:15:01	Desaturate...	Turn colors into shades of gray
01:15:08	Save As...	Save this image with a different name
01:15:48	Quit	Quit the GNU Image Manipulation Program

Table 7.1: A partial listing of data collected by InterTwine during a session in which a user was tasked with selectively desaturating the background of an image. In addition to timestamps and menu labels, this table lists the tooltips associated with operations performed during the session. I argue that the collection of tooltips can be read as a transcript, and can be used to index video tutorials in cases where it is known that a user is watching such content while performing tasks in the application.

7.1.4 Indexing Video and Pictorial Tutorials

Continuing the theme of leveraging application context and activity to impact the generation and presentation of web search results, large-scale deployments of InterTwine-like systems enable compelling possibilities for the indexing, analysis, and interpretation of video tutorials. Specifically, a search engine could index video tutorials by treating the list of commands and actions invoked by the user as a form of transcript. For example, Table 7.1 lists the menu labels and tooltips, as captured by InterTwine, for a sequence of commands that achieve the previously-mentioned selective desaturation effect in the GIMP software. As can be seen from this example, the list of tooltips reads almost as if it were a descriptive transcript of actions performed in the video tutorial. Since video tutorials often are consumed through web browsers, these types of command sequences and tooltip transcripts are easily associated with online video content using a tool such as InterTwine.

7.1.5 Interpreting and Contextualizing Instrumentation Data

Large-scale deployments of InterTwine-like systems could also enrich the types of usability data that can be extracted from software instrumentation efforts. HCI research has a long history of studying logs of low-level interface events, with the goal of gaining insights into a system's usability [63, 67]. While past work was successful in characterizing application usage in aggregate (e.g., average number of commands known by users), early techniques were largely unsuccessful in yielding fine-grained insights about a system's usability. Recent successes in this space (e.g., [5, 4]) result from the incorporation of additional context to facilitate the interpretation of low-level events. Notably, Akers et al. demonstrated that low-level backtracking events (e.g., occurrences the "*Undo*" command) can be used as indicators of potential usability problems in feature-rich software. However, correct interpretation of these events depends on access to short video clips detailing the contexts in which these commands were invoked. I hypothesize that web search and web browsing logs could similarly be used to provide the context needed to interpret these indicative low-level actions.

Likewise, Hurst et al. demonstrated that mousing events (e.g., dwelling, or the failure to issue a command after opening a menu) can be leveraged to determine when users are struggling to find commands in the menus of a GUI [66]. Pairing these data with records of user search queries might provide a means of determining the functionality users are seeking but failing to find. For example, if one observes a user of the GIMP software opening the "*Color*" menu, then failing to issue a command, and finally searching online for [*how to make a photo black and white in gimp*], then one might surmise that the user expected to locate the desired functionality in the aforementioned menu[3].

7.1.6 Longitudinal Studies of Software Learning

I am also very excited about the possibilities that arise when systems such as InterTwine are deployed and studied over longer timeframes. As suggested by Matthew Richardson in [106], longitudinal behaviour data (query data in particular), enables researchers to answer questions that would be difficult or impossible to otherwise address. In the context of this dissertation, one compelling use of longitudinal data could be to explore temporal patterns in the information needs of a system's user population. Specifically, one might ask: Can query log data characterize a user's transition from novice to expert?

In this vein, Ryen White, Eric Horvitz and I recently developed techniques to explore temporal patterns in the information seeking habits of new and expectant parents (Figure 7.3). We found that information needs evolve over time, sometimes very rapidly, but that the sequence and timing of concerns is nonetheless predictable, unfolding almost as if following a script or a schema. One wonders if the techniques developed for that work can be applied or adapted to produce a similar set of models and visualizations for queries pertaining to use of an interactive system. For example, one might investigate the types of queries users issue in the first, fifth, or fifteenth week following the release of a major update to a software product.

7.2 Search-driven Interaction

It is also worthwhile to consider one extreme point in the design space over the types of integrations proposed in this dissertation. In this section I consider a future in which

[3]GIMP's "*Color*" menu does contain the "*Desaturate*" command, but users frequently fail to recognize this command's relevance to the task of converting an image to black an white, as detailed in Chapters 3 and 4).

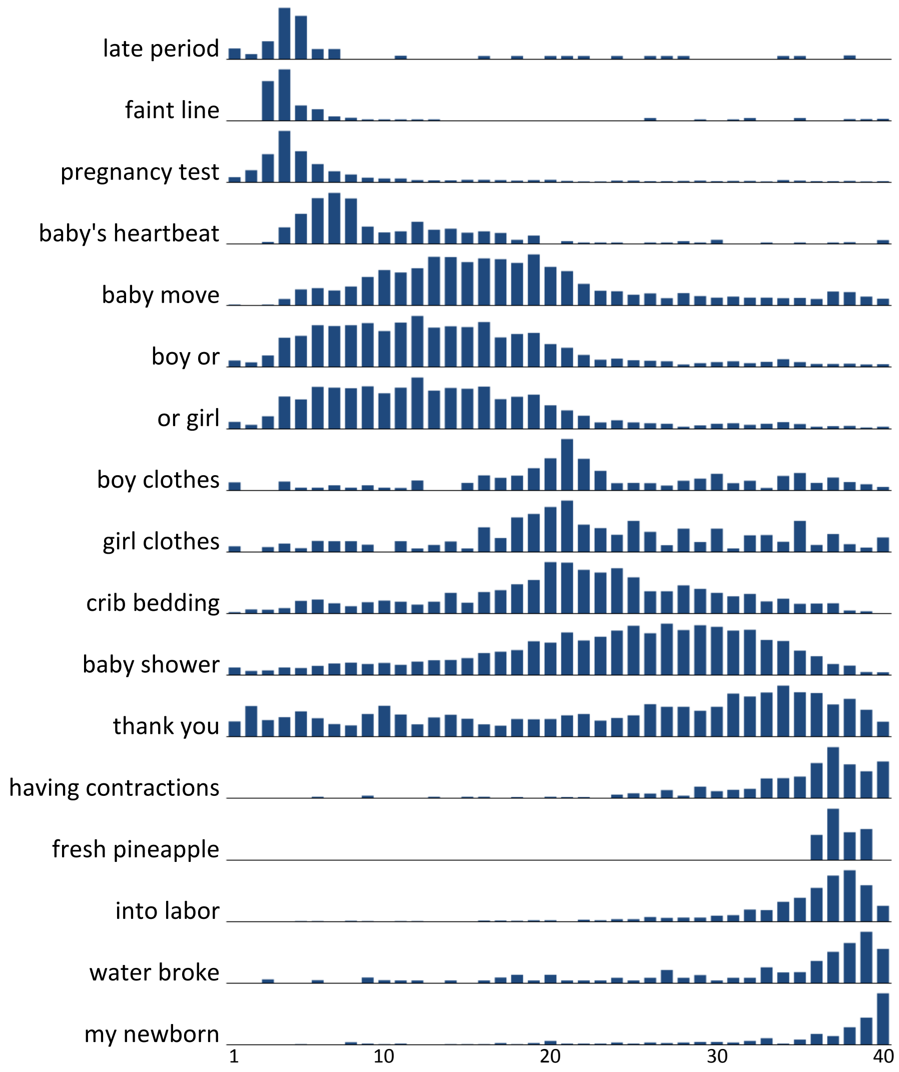
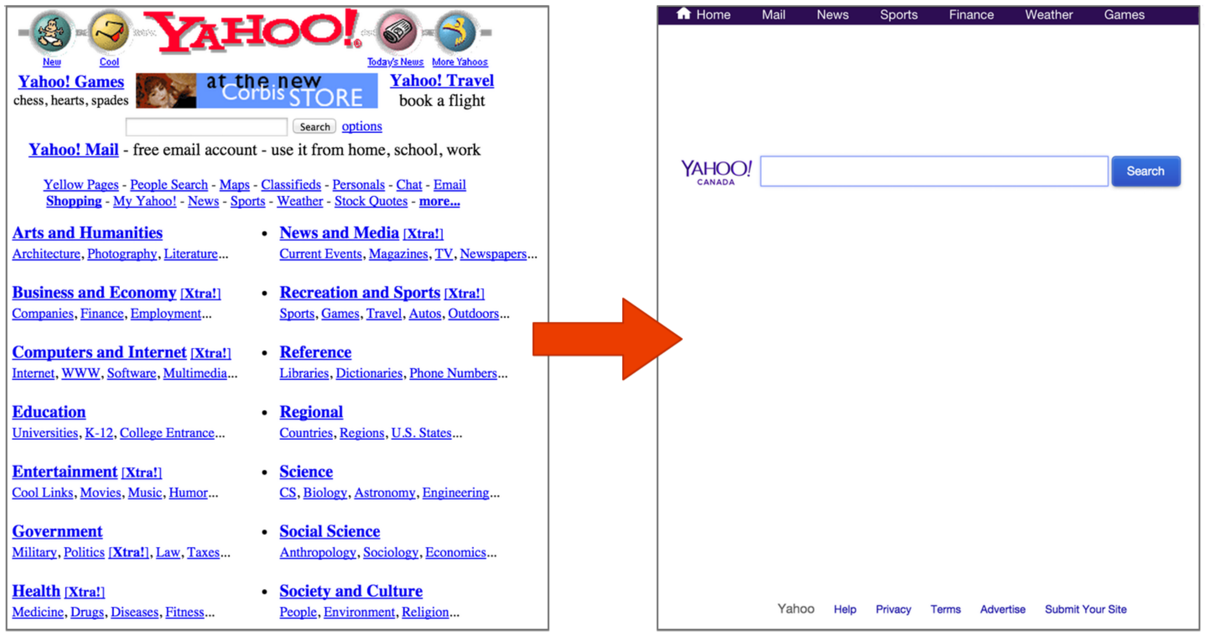


Figure 7.3: Histograms of query bigrams over 40 weeks of gestation. Time-dependent query volumes of each bigram (left) are displayed by gestational week for searchers who self-identified as pregnant (n=13,030 searchers). In this figure, reproduced from [52], each bar represents the proportion of searchers who searched at least once for the bigram of interest in the corresponding gestational week. In the context of this dissertation, I ask: Can similar visualizations be generated to characterize temporal patterns in how people learn software? For example, what queries do people ask in the first week following the adoption of a new piece of technology?

web search becomes a primary means of interacting with feature-rich software. This exploration is motivated by the following observation: In the late nineteen-nineties, when the web was still nascent, search was often paired with portals or directories organizing popular web content into a series of topic categories (Figure 7.4, left). These directo-


New Cool Today's News More Yahoos

Yahoo! Games
chess, hearts, spades

at the new Corbis STORE

Yahoo! Travel
book a flight

Search options

Yahoo! Mail - free email account - use it from home, school, work

Yellow Pages - People Search - Maps - Classifieds - Personals - Chat - Email
Shopping - My Yahoo! - News - Sports - Weather - Stock Quotes - more...

Arts and Humanities
Architecture, Photography, Literature...

Business and Economy [Xtra!]
Companies, Finance, Employment...

Computers and Internet [Xtra!]
Internet, WWW, Software, Multimedia...

Education
Universities, K-12, College Entrance...

Entertainment [Xtra!]
Cool Links, Movies, Music, Humor...

Government
Military, Politics [Xtra!], Law, Taxes...

Health [Xtra!]
Medicine, Drugs, Diseases, Fitness...

- News and Media [Xtra!]
  Current Events, Magazines, TV, Newspapers...
- Recreation and Sports [Xtra!]
  Sports, Games, Travel, Autos, Outdoors...
- Reference
  Libraries, Dictionaries, Phone Numbers...
- Regional
  Countries, Regions, U.S. States...
- Science
  CS, Biology, Astronomy, Engineering...
- Social Science
  Anthropology, Sociology, Economics...
- Society and Culture
  People, Environment, Religion...

Home Mail News Sports Finance Weather Games

YAHOO! CANADA Search

Yahoo Help Privacy Terms Advertise Submit Your Site

Figure 7.4: The evolution of the Yahoo! search page between December 1998 (left), and May 2015 (right). Early versions of the Yahoo! search page featured a directory of popular web content categorized by topic. As users became more comfortable with web search, and as the web increased in scale, this directory listing has not survived to modern instantiations of the Yahoo! search web page.

ries can be thought of as a taxonomy of web pages, and represent one particular way of organizing this information space. As the web became more expansive, and as users became more familiar with web search, the portals and directories were outmoded by search (Figure 7.4, right). This scenario is directly comparable to the present-day organization of feature-rich software (Figure 7.5), where features are carefully organized into the tree-like structures of a system's menuing system. Now that search is being incorporated into feature-rich software, I ask if the same evolution will occur in user interfaces (Figure 7.6), as was seen with web portals? And, if so, what interaction challenges may arise?

Likely, one of the first challenges that one would encounter, should search replace the standard GUI, would be one of discoverability. One of the tenets of good usability is to favour recognition over recall by making "objects, actions, and options visible" [93]. One advantage of standard GUIs is that users can navigate menus, windows and tabs to discover the functionality afforded by a feature-rich system. An interactive system where functionality is accessed only through queries would be at a disadvantage in this regard. However, query autocompletion services, and command recommendation systems similar to [90], could alleviate much of this concern.

As one ventures further into this region of the design space, it is likely that other, more significant, challenges will emerge. But, if such an evolution is fraught with challenges

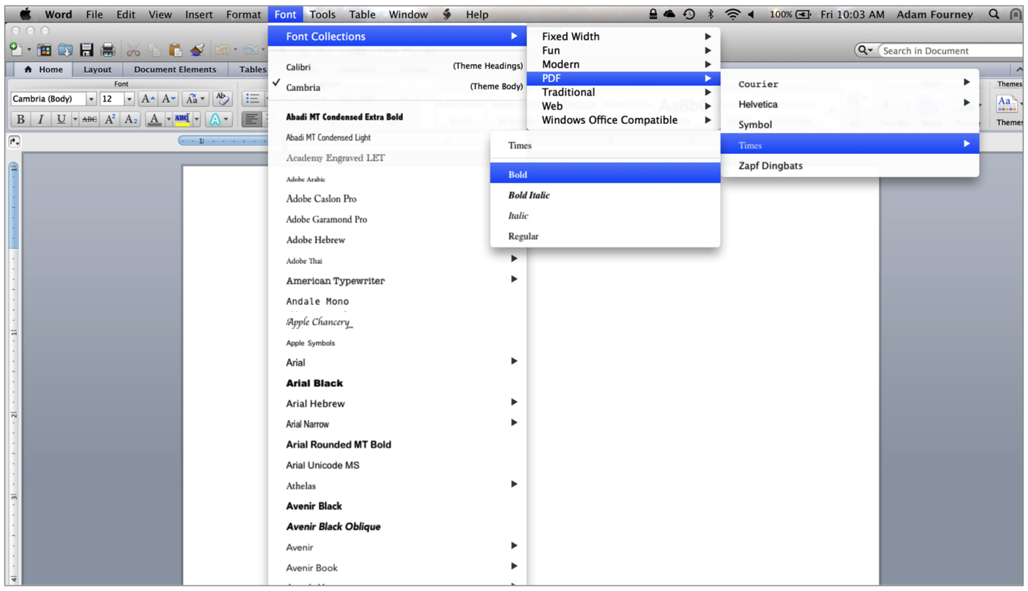


Figure 7.5: A screenshot of Microsoft Word, version 2011, for the Macintosh operating system. In this interface, functionality is neatly organized into hierarchical menus, representing a categorization or taxonomy of system functionality.

and pitfalls, why pursue this direction in the first place? One of the primary motivating factors, for pursing such an extreme, is that computation is increasingly moving onto mobile and wearable devices such as smart phones and smart watches. Screen real-estate for displaying standard GUI widgets is at a premium, and query or natural language-based interactions are becoming a primary means of interacting with these devices (e.g., users of Apple's smart watch can issue spoken commands to access much of the device's [28] functionality). As computation increasingly moves off of our desktop computers and onto our mobile devices, query-based interactions will likely play a more significant role in how people get work done. This sentiment is well supported by Donald Norman, who has written about the role of search as a means of interacting with complex software:

> *GUIs work well when the number of alternative items or actions is small. When the number of items reaches the level typical of today's complex operating systems, applications and the information spaces of the Internet, the GUI does not scale well. (...) What is to replace the GUI? Ah yes, journalists are constantly asking me that question. Well the answer is simple, and its already here: search [95]*

To this end, I believe the ideas presented in this dissertation will only increase in relevance over time, and I look forward to a future where our interactions with feature-rich software benefit from the same types of advances that have propelled web search to its present status as an indispensable tool for navigating the complexities of our modern information-rich lifestyles.

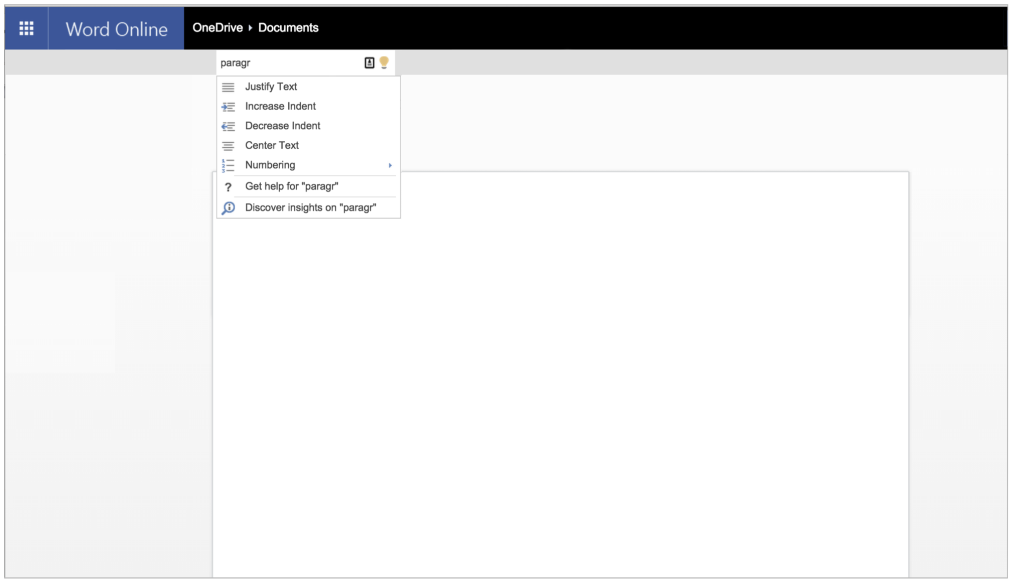


Figure 7.6: A hypothetical future version of the Microsoft Word interface, where standard GUI components are replaced with a search bar. This evolution mirrors that of the Yahoo! search result page, depicted in Figure 7.4.

7.3 Integrations in other domains

In the introduction to this dissertation, the Gulf of Execution was described as a situation where a user has a goal in mind, can express that goal in words, but struggles to determine the correct sequence of steps needed to reach his or her objective. The web-mediated execution bridge then established a framework in which both the user's goal and the user's subsequent actions can be observed and reasoned about by a computer system. While this dissertation has focused entirely on goals and actions related to the use of feature-rich software, the web-mediated execution bridge need not be limited to this domain. People routinely express their personal goals as search queries, while leveraging technologies and sensors to track their day-today activities in the real world. As such, it is becoming increasingly feasible to contemplate the web-mediated execution bridge, and the types of analyses and integrations described in this dissertation, in contexts much broader than outlined thus far.

One example of this broader context is afforded by the personal informatics, or quantified self, movement. Li et al. define personal informatics systems as "*those that help people collect personally relevant information for the purpose of self-reflection and gaining self-knowledge*" [86]. For instance, one may wear an on-body sensor such as the Fitbit [45] with the goal of tracking one's sleep patterns and optimizing one's chances for a restful sleep. However, it can be difficult for people to interpret the low-level data that is output from such devices [86, 22]. I hypothesize that pairing these data with a synchronized record of web search could provide quantified selfers with the context needed to better interpret their data, perhaps relating their sleep patterns to activities or stressors encountered throughout the day. Web query logs are ideal data sources in these contexts because, as noted by Matthew Richardson, they often resemble surveys in which users are asked "*to, every day, write down what they were interested in, thinking about, planning, and doing.*" [106].

Likewise, when personal successes (or failures) are achieved, it may be possible to relate these positive (or negative) outcomes to strategies employed by users – especially if users have accessed online how-to content, or have sought advice from an online forum or social network. To this end, existing work has already demonstrated how analyses of Twitter data can be used to both detect those who have attempted to quit smoking, and their eventual successes or failures [92]. Likewise, in the space of web search, work has demonstrated that query logs can be used to both detect one's interest in dieting, and to monitor one's subsequent caloric intake over the ensuing weeks or months [127]. In each of these cases, I believe that the web-mediated execution bridge provides a common framework for characterizing the uses of web search, and for contemplating the development of tools that could help users achieve their personal goals.

7.4 Conclusion

In this chapter I characterized some potential directions for future research, including: exploring opportunities that arise should systems such as InterTwine ever be deployed at web-scale, contemplating future interactive systems where search is a primary means of getting work done, and discussing the applicability of the web-mediated execution bridge in contexts beyond the use of feature-rich software. In all cases, I believe that search will become an even more essential component in our interactions with technology, and with the world in general. I hope that this dissertation can serve as a foundation for research moving forward in this space.

Chapter 8

Summary and Conclusion

Web search, online tutorials, and other online resources, serve an integral role in how people learn and use feature-rich software systems on a daily basis. Users depend on web resources for initial instruction, as a first line of technical support, and as a strategy for coping with the complexities of feature-rich systems (e.g., by leveraging web resources as an external memory store that can be accessed at will [116]). When users rely on web resources to support their use of complex software, the act of querying a search engine signals that the user has a goal in mind, that they are able to express their goal in their own words, but that they are unsure (or cannot recall) which concrete low-level operations will accomplish their goal in the software application. These low-level plans and action sequences are enumerated in the online resources retrieved from search (e.g., web tutorials), and are translated by the user into actions performed within the software itself. In other words, users are leveraging online resources to bridge Donald Norman's Gulf of Execution [96].

When leveraging online resources to bridge the Gulf of Execution, each step of the journey is mediated by a user interface (UI), namely the UIs of: the web search engine, the retrieved document, and the application itself. Actions in these three environments constitute steps across what I have termed the *web-mediated execution bridge*. Contextualizing user interactions with respect to the web-mediated bridge, and developing methods to form or enhance connections between each of these environments, affords new opportunities to learn about users, and enables the development of new end-user tools and services. In order of appearance, the opportunities explored in this dissertation include:

Characterizing Usability through Search (Chapter 3)
In Chapter 3, I argued that query logs of web search engines serve as centralized repositories cataloguing the day-to-day needs of a system's user base, offering insights that are unprecedented in HCI research both in scale and in ecological validity. To accomplish this, I described how aggregates of query logs can be approximated using data made available by query-autocompletion services as well as advertising tools.

Using these data, I developed an automated system for gathering, labelling and filtering queries for the purpose of identifying common tasks and issues encountered by a system's user base on a day-to-day basis. I presented several examples and case studies demonstrating how these data could be used to detect potential usability defects, and to contextualize these defects with real-world incidence rates. In this sense, CUTS indirectly helps to close the Gulf of Execution by directing developer attention towards problem areas, ideally resulting in improvements going forward.

Query-Feature Graphs (Chapter 4)
In Chapter 4, I presented an automated system that leverages search queries and relevant web tutorials to construct query-feature graphs, structures that map from technical terminology in an application's interface (e.g., GIMP's "desaturate" tool) to words, phrases, and concepts familiar to the user (e.g., "make black and white"). Query-feature graphs effectively link the first two steps across the web-mediated execution bridge, and enable a number of compelling applications including: interface search, intelligent tooltips, and application-to-application analogy search. In this same chapter, I assessed the quality of Query-feature graph associations, which were discovered automatically using methods from question-answering literature. To this end, I reported that the question-answering approach significantly outperformed a baseline cosine similarity-based approach – the latter of which is in common use among existing implementations of interface search.

InterTwine (Chapter 5)
Finally, Chapter 5 presents a system which integrates all three steps across the web-mediated execution bridge, enabling a feature rich application to respond to actions performed on the web, and enabling web resources (e.g., search engine result pages) to respond to actions performed in the feature-rich software. In this chapter, I explored one concept afforded by this bidirectional communication: Namely, inter-application information scent. To this end, InterTwine augments web search results by listing software commands used when previously viewing a webpage, and highlights menu items in desktop applications when those items are mentioned in online tutorials retrieved by the user. Feedback from an initial set of users revealed that InterTwine's features were both welcomed and appreciated, and users expressed an interest in achieving similar integrations with other feature-rich software technologies they use on a daily basis.

As human-computer interaction researchers, system designers, and user experience practitioners, it is time that we begin considering software applications, web search, and online support materials as individual parts of a larger holistic system to be studied, designed and evaluated in concert. As this dissertation has shown, these integrations result in novel insights about users, and in tools that greatly expand and enhance user capabilities.

Appendix A

Named command recognition

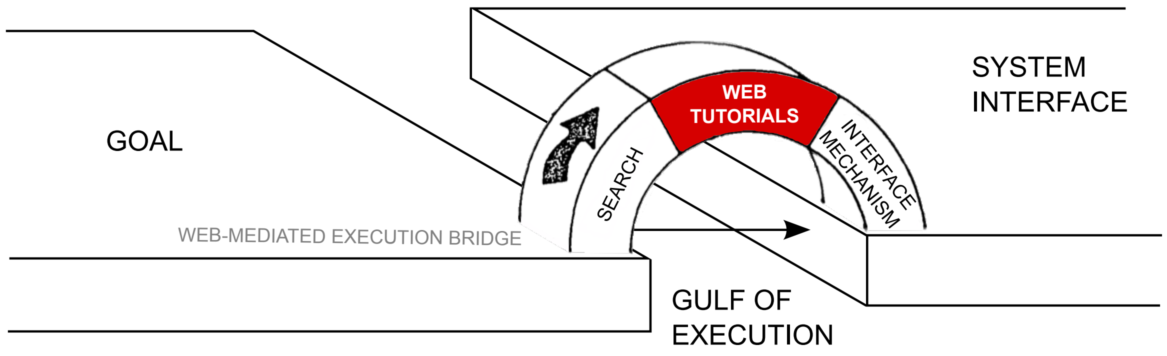


Figure A.1: This appendix describes an approach to named command recognition – a domain-specific instance of the more general named-entity recognition problem. The goal of a named command recognizer is to detect commands or system features that are mentioned in tutorials downloaded from the Internet. Portions of this chapter were first published in [47].

The Internet contains a wealth of reference material, tutorials, and other documentation related to the use of interactive systems. Written for the benefit of users, this documentation is expressed in natural language. However, from the perspective of a software system, this documentation is opaque and unactionable.

Recently, the CHI community has demonstrated interest in the problem of automatic identification of references to user interface components within online documentation. For example, Chapter 4 describes query-feature graphs, structures which leverage web tutorials to pair high-level search terms with the corresponding features of a user interfcae. Likewise, work by Ekstrand *et al.* demonstrates how a custom search engine can extract commands mentioned in software tutorials, so that those commands can be listed in the snippets presented as part of enhanced search result pages [42]. Similarly, InterTwine, presented in Chapter 5, extracts commands mentioned in online tutorials to enable a form on interapplication information scent. Also closely related is work by Lau *et al.* [85], which has demonstrated the possibility of extracting *action-target-value* triples from how-to instructions, with the goal of advancing machine-guided help systems.

The problem of identifying user interface elements mentioned within instructional documentation is a domain-specific instance of *named entity recognition*. In the context of technical documentation, we are interested in the problem of extracting named entities that refer to interface components, such as commands, menu items, dialogs, settings, and tools within the interface.

In this chapter, we introduce a named-entity recognizer for detecting user interface elements mentioned within HTML documents. We call this specific problem *named command recognition*. In the following sections, we enumerate the specific challenges for this problem, then discuss how certain informal conventions in tutorial writing can be leveraged to better detect named entities in this context. From these conventions, we derive a set of features and a general classification strategy that leads to accurate recognition of user interface elements referenced within text.

A.1 Challenges and Recognition Strategies

Web-based tutorials and documentation use natural language to describe how to perform specific tasks with interactive applications. To be clear and unambiguous, authors typically refer to user interface elements by their given, visible names (i.e., captions or labels) [85, 48]. For example, an author may write that the user should invoke the "Undo" command from the "Edit" menu.

Given this practice, an obvious strategy to named widget recognition is to simply search the documentation for strings matching known widget labels. This tact requires a complete list of widget captions, but numerous, reliable approaches exist for automatically enumerating and extracting the captions of widgets in both web [87] and desktop [48, 105] applications. We refer to this overall strategy as the *baseline* approach to named widget recognition. This baseline approach has been employed in the past by the enhanced search results work by Ekstrand *et al.* in [42].

While simple and straightforward, the baseline strategy is prone to error. Consider, for example, a tutorial describing various options for converting a colour image to black and white with the GIMP raster graphs software[1]. In the full tutorial, approximately 170 distinct phrases match the labels or captions of components found in the GIMP user interface. However, upon inspection of the tutorial, only approximately 50 phrases are actual references to GIMP operations, making the false positive rate of this approach greater than 70%. An excerpt from this tutorial is listed in Figure A.2, with all matches outlined with rectangles and false positives crossed out.

[1]http://www.gimp.org/tutorials/Color2BW/

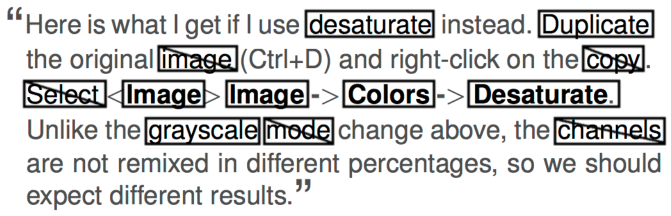

“Here is what I get if I use desaturate instead. Duplicate the original ~~image~~ (Ctrl+D) and right-click on the ~~copy~~. ~~Select~~ <**Image**> **Image**-> **Colors**-> **Desaturate**. Unlike the grayscale ~~mode~~ change above, the ~~channels~~ are not remixed in different percentages, so we should expect different results.”

Figure A.2: An excerpt from a tutorial describing how to convert a colour image to black and white using GIMP [1]. Substrings matching the names of GIMP commands are outlined in rectangles. False positive matches are crossed out.

Online documentation also often includes its own set of menus, menu items, and controls for navigating and interacting with the website. These documentation widgets can also generate false matches when employing the baseline approach.

In the following sections, we identify a number of cues that can be employed to improve upon this baseline accuracy.

A.1.1 Leveraging Prior Beliefs

In the previous example, the term “desaturate” can be found in the first sentence (Figure A.2). This term is highly technical, and rarely occurs in more general writing. Without additional evidence, we would expect that the use of this term refers to GIMP’s “Desaturate” command. Conversely, in the second sentence, we encounter the term “image”. This term is very generic, and occurs in many contexts beyond GIMP tutorials. Without additional evidence, we assume that it represents a false detection. Thus, by estimating how often a phrase refers to a widget in a given corpus, it is possible to correctly label many named widget references without any further consideration of the context in which the matches occur.

A.1.2 Leveraging Informal Conventions

In sentence 3 of Figure A.2, the term “Image” twice refers to actual, named widgets. In such situations, further evidence is needed to overcome the prior expectation that generic phrases do not represent commands. This additional evidence is provided by informal conventions that are often employed in tutorial writing. For example, command names are often capitalized, and are occasionally styled to appear differently than the surrounding text (in this case, with a bold weighting). Special punctuation is also often employed to specify menu hierarchies (e.g., “->” in “File->Open”). The use (or lack of use) of these conventions can also factor into the classification of terms.

A.1.3 Leveraging Page Context

Page context can provide additional evidence to help correctly identify named widgets in cases where the online documentation contains links and website navigation with names matching those of the software it is documenting (e.g., "Help", "About", etc.). Specifically, matches that occur in regions of the page that resemble site navigation reduce the confidence that the matching term directly references a user interface widget.

Given this foundation, we now describe a feature set and a general classification framework that enables accurate recognition of named widgets.

A.2 Classification Framework

Identifying UI components referenced in web documentation is a three step process. First, the document is examined for substrings matching the names of known interface components or commands (the baseline approach described above) to yield a set of *candidate matches*. In the second step of the process, a set of features is extracted from the context of every candidate match (where the features are derived from the cues and informal conventions discussed above). In the final step, a classifier determines which substring matches are indeed references to widgets.

A.2.1 Feature Extraction

To more correctly classify candidate named widgets, we employ the following set of features.

Capitalization

Recognizing the tendency for authors to capitalize references to commands, our classifier employs two related capitalization features: 1) Whether a candidate match's first token is capitalized, and 2) the total number of capitalized tokens within the match sequence. For example, "Save Selection to File" has three of its four tokens capitalized, including the first.

Next and Previous Tokens

The *next and previous tokens* features record the tokens immediately preceding and following the candidate named widget. These features are designed to model common phrases in which named widgets appear (e.g., "the File menu"). They also model conventions for describing menu hierarchies, such as the use of ">" or "->".

Next and Previous Candidates

Given a candidate match, the *next candidate* and *previous candidate* features record the candidate matches found before and after the current match. All tokens occurring between candidate matches are ignored. These features are designed to recognize and leverage common command sequences (e.g., "Copy and Paste") as well as parent-child relationships expressed in menus (e.g., "Edit > Paste As > New Layer").

Element Occupancy Ratio and Element Type

In web-based tutorials, references to interface elements are often expressed with some styling that makes the text visually distinct from the surrounding text. The *element occupancy ratio* feature models these markup differences as follows. First, we identify the HTML element directly enclosing the candidate match. We then compute the ratio between the number of tokens making up the candidate and the total number of tokens enclosed by the HTML element. For example, the word "pencil" exhibits a 1: 4 ratio in the phrase "`<p>Select the pencil tool</p>`", but a 1: 1 ratio in the phrase "`Select the <b>pencil</b> tool`". In addition to the element occupancy ratio, we extract a feature recording the *type* of the enclosing element.

Text-to-Tag Ratio and Location (wrt. the Start of the Page)

Tutorial content, and hence named widgets, often make up the "main content" of the enclosing HTML document (as opposed to secondary content such as navigation, headers, and advertisements). In the information extraction literature, the *text-to-tag ratio* has been found to be a good feature for discriminating between main and secondary content [126]. Specifically, main content is typified by regions of the HTML document containing many text tokens, but few HTML tags. These regions are said to have a high text-to-tag ratio. Similarly, main article content is often found in the central region of an HTML file, somewhere between header content and footer content. Thus, we use the *location* of the match within the HTML document to help rule out candidates that are likely part of a tutorial's site navigation.

Notably absent from this list of 10 features is any representation of our prior beliefs regarding the likelihood that a candidate match represents a named entity. Prior beliefs are those held before considering evidence (i.e., features), and are modelled by the classifier directly. We describe the classifier next.

A.2.2 A Naive Bayes Classifier with Witten-Bell Smoothing

The features outlined above are compatible with many modern text classification and information extraction techniques. In this work, we elected to construct a recognizer that employs naive Bayes classification. While potentially less effective than more complex techniques (e.g., [44]), naive Bayes classifiers have nonetheless proven to be effective general purpose classifiers, and are certainly sufficient for demonstrating the feasibility of named-widget recognition. Moreover, naive Bayes classifiers confer a number of unique advantages. First, the number of parameters in a naive Bayes model scales linearly with the number of features and classes [109]. As a result, comparatively less training data is required to produce an effective classifier. Additionally, the "naive" independence assumption affords the ability to independently learn the feature distributions, thus enabling efficient training in the types of distributed systems typical of existing web indexing platforms.

In order to classify commands, we employ a separate binary naive Bayes classifier for each UI component we would like to recognize. When classifying a candidate match, we invoke only the classifier corresponding to that named component. For example, substrings matching the text "File" invoke the classifier corresponding to the system's *File* menu.

By treating commands separately rather than collectively via a single class or classifier, we ensure that the framework is able to directly model the subtle differences in context in which named entities are expected to occur (e.g., enabling the classifier to learn menu structures). Unfortunately, the strategy aggravates the problem of sparse data, and the training data may not include mentions of every available widget. To overcome the sparse data problem, we use Witten-Bell smoothing [130]. Witten-Bell smoothing uses the training data to estimate the likelihood of novel events. It then uses this estimate, together with a more general "backoff" model, to redistribute probability mass, thus filling in missing information. In our case, the backoff model is constructed by pooling the training data for all named entities into a single "generic widget" class. As a concrete example, suppose that the training data does not include any examples of the "Cut" command. With Witten-Bell smoothing, we can use what we know about commands in general to predict how references to the "Cut" command might appear in text.

A.3 Evaluation

To evaluate the accuracy of the classification framework, we trained classifiers for each of the software applications listed in Table A.1. In the following sections, we describe how the necessary training data was collected, how the classifiers were evaluated, and report the results of the evaluation.

A.3.1 Generating Training Data

In the course of conducting previous research with query-feature graphs (Chapter 4), we had previously amassed a corpus of thousands of web documents pertaining to each of the

	GIMP	Inkscape	Thunderbird	Total
Classifier:				
True Positives	598	405	445	1448
True Negatives	2040	2668	4215	8923
False Positives	53	91	71	215
False Negatives	133	181	164	478
Precision	0.92	0.82	0.86	0.87
Recall	0.82	0.69	0.73	0.75
F1 score	0.87	0.75	0.79	0.81
Accuracy	93.4%	91.9%	95.2%	93.7%
Baseline:				
F1 score	0.41	0.30	0.22	0.30
Accuracy	25.9%	17.5%	12.4%	17.4%

Table A.1: Evaluation results characterizing the performance of the proposed classification framework. For comparison, the final two rows of the table present performance characteristics of the baseline method.

software applications listed in Table A.1. These corpora were collected using standard web crawling procedures. To generate training data for a particular application, we randomly sampled 35 documents from the associated document collection. We then identified candidate matches within each document, and manually labeled each as either referring to a command or not. Since terminology varies in generality from application to application (e.g., the names of commands in GIMP tend to be technical), and because we sampled an equal number of pages for each application, the number of labeled examples in each training set differ.

A.3.2 Evaluation and Results

In order to measure the accuracy of the classifiers, we employed leave-one-out validation on a per-page basis: in each round, the classifier is trained with items found in all but one of the web documents, and is evaluated using the withheld document. We report the results of this experiment in Table A.1. To provide a point of comparison, the table also presents the accuracy achieved when using only the baseline approach.

The results suggest that the classification framework performs well. The tested classifiers are able to accurately label an average of 93.7% of all candidate named entities, surpassing the baseline accuracy by a wide margin. Viewed as a retrieval problem, the classifier returns an average of 75% of all actual named widgets (i.e., recall), with an average true-positive rate of 87% (i.e., precision). This yields an overall F1 score of 0.81.

To verify the classifiers were modelling the phenomena as expected, we manually inspected the conditional probability tables making up the various naive Bayes classifiers.

Feature Set	F1 score
Next and Previous Candidate Matches	0.78
Next and Previous Tokens	0.63
Capitalization	0.63
Text-to-Tag Ratio / Location	0.59
Element Occupancy Ratio / Element Type	0.50
No features (i.e., using the prior distribution directly)	0.50

Table A.2: Classification performance when using only the features named in the leftmost column.

Through this inspection, we found that the classifiers were indeed modelling various structural aspects of the target interface. For example, our approach automatically learns that the most likely token to follow a reference to GIMP's "Fuzzy Select" command is the word "tool", which is expected since "Fuzzy Select" is a tool that appears in GIMP's toolbox. Similarly, the words "window" or "dialog" are the most likely tokens to follow references to GIMP's "Channels" window.

Finally, to determine the relative importance of the various features employed by the classifier, we re-ran the experiment with classifiers employing various subsets of the available features. For example, we found that classifiers employing *only* the "Next and Previous Candidates" features achieve an overall F1 score of 0.78 across the three applications. Similarly, the classifiers utilizing only the "Capitalization" features achieve an overall F1 score of 0.63. The difference between these scores reflects the relative importance of the associated features. Of all features, "Element Occupancy Ratio" and "Element Type" were the least effective (F1 score of 0.50), providing less inferential leverage than we would have liked. These features can likely be left out of future classifiers without much detrimental effect. A summary of the F1 scores for the various feature sets is listed in Table A.2.

A.4 Discussion and Future Work

In this chapter, we have demonstrated a system that draws upon informal practices in tutorial writing to detect references to named widgets in online documentation. The resulting named widget recognizer provides the foundation for a number of new interaction possibilities, including: (1) new means of indexing and searching tutorials, (2) the ability for users to invoke commands directly from within tutorial text, and (3) the creation of summaries that highlight important steps in long tutorials (i.e., generating "Quick Start" guides).

As with any complex system, there are a number of limitations of our recognizer worthy of further discussion.

First, our recognizer compares online documentation against a list of known widgets, and disambiguates between meaningful references and spurious/coincidental string matches. Our recognizer is both trained and evaluated using automatically generated lists of all strings in a user interface (e.g., captions, labels, tooltips, etc.). A limitation of this approach is that it does not consider cases where online documentation refers to a widget using alternative text (e.g. "click the 'No' button" when the button is actually labeled "Cancel"). Consideration of such novel synonyms can be expected to lower the recall scores as compared to the results reported in Table A.1. This situation can be partially ameliorated by adding common synonyms to the list of widget names, but a more general solution for detecting novel synonyms remains a topic of future work.

Additionally, our system cannot disambiguate between distinct widgets that share a common name. For instance, in the GIMP 2.6 interface, both a menu and a panel share the caption "Layers". Fortunately, this does not significantly impact the applicability of the proposed technique to applications such as improved tutorial indexing or tutorial summarization. Moreover, we can often rely on the user to differentiate between multiple possible interpretations. In the future, we hope to leverage additional context to automatically disambiguate between identically named widgets.

Despite the aforementioned considerations, the proposed recognizer can be immediately applied to existing research, including: query-feature graphs, presented in Chapter 4; InterTwine, presented in Chapter 5; the work of Lau *et al.* [85]; and, the work of Ekstrand *et al.* [42].

A.5 Addendum

In the time between the first publication [47] of this work in 2012, and the preparation of this dissertation in 2015, Laput et al., improved upon our results by leveraging Conditional Random Fields (CRFs) rather than Naive Bayes classifiers. In their work [84], the authors trained on 400 tutorials, rather than on 35, and achieved F1 scores of 0.98 and 0.97 for detecting Photoshop tools and menu items respectively. This can be compared to the results presented in this Chapter, which achieve an F1 score of 0.87 for detecting mentions of GIMP commands in general. While we cannot rule out the impact of training on an order of magnitude more data, we consider CRFs to be the state-of-the-art in named command recognition going forward.

References

[1] Eytan Adar, Mira Dontcheva, and Gierad Laput. CommandSpace: modeling the relationships between tasks, descriptions and features. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 167–176, New York, NY, USA, 2014. ACM.

[2] Eytan Adar, Jaime Teevan, and Susan T. Dumais. Large scale analysis of web revisitation patterns. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, page 1197–1206, New York, NY, USA, 2008. ACM.

[3] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 19–26, New York, NY, USA, 2006. ACM.

[4] David Akers, Robin Jeffries, Matthew Simpson, and Terry Winograd. Backtracking events as indicators of usability problems in Creation-Oriented applications. *ACM Trans. Comput.-Hum. Interact.*, 19(2):16:1–16:40, July 2012.

[5] David Akers, Matthew Simpson, Robin Jeffries, and Terry Winograd. Undo and erase events as indicators of usability problems. In *Proc CHI '09*, page 659–668, New York, NY, USA, 2009. ACM.

[6] Anne Aula, Natalie Jhaveri, and Mika Käki. Information search and re-access strategies of experienced web users. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, New York, NY, USA, 2005. ACM.

[7] Anne Aula, Rehan M. Khan, and Zhiwei Guan. How does search behavior change as search becomes more difficult? In *Proc CHI '10*, page 35–44, New York, NY, USA, 2010. ACM.

[8] Ricardo Baeza-Yates and Alessandro Tiberi. Extracting semantic relations from query logs. In *Proc KDD '07*, page 76–85, New York, NY, USA, 2007. ACM.

[9] Ziv Bar-Yossef and Maxim Gurevich. Mining search engine query logs via suggestion sampling. *Proc. VLDB Endow.*, 1(1):54–65, 2008.

[10] Rob Barrett, Eser Kandogan, Paul P. Maglio, Eben M. Haber, Leila A. Takayama, and Madhu Prabaker. Field studies of computer system administrators: analysis of system management tools and practices. In *Proc CSCW '04*, page 388–395, New York, NY, USA, 2004. ACM.

[11] Patrick Baudisch, Desney Tan, Maxime Collomb, Dan Robbins, Ken Hinckley, Maneesh Agrawala, Shengdong Zhao, and Gonzalo Ramos. Phosphor: Explaining transitions in the user interface using afterglow effects. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, page 169–178, New York, NY, USA, 2006. ACM.

[12] Paul N. Bennett, Ryen W. White, Wei Chu, Susan T. Dumais, Peter Bailey, Fedor Borisyuk, and Xiaoyuan Cui. Modeling the impact of short- and long-term behavior on search personalization. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, page 185–194, New York, NY, USA, 2012. ACM.

[13] Michael S. Bernstein, Bongwon Suh, Lichan Hong, Jilin Chen, Sanjay Kairam, and Ed H. Chi. Eddi: interactive topic-based browsing of social status streams. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, page 303–312, New York, NY, USA, 2010. ACM.

[14] S. R. K. Branavan, Luke S. Zettlemoyer, and Regina Barzilay. Reading between the lines: learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, page 1268–1277, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[15] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. Example-centric programming: integrating web search into the development environment. In *Proceedings of the 28th international conference on Human factors in computing systems*, page 513–522, New York, NY, USA, 2010. ACM.

[16] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the 27th international conference on Human factors in computing systems*, page 1589–1598, New York, NY, USA, 2009. ACM.

[17] Michael Brasser and Keith Vander Linden. Automatically eliciting task models from written task narratives. In Christophe Kolski and Jean Vanderdonckt, editors, *Computer-Aided Design of User Interfaces III*, pages 83–90. Springer Netherlands, January 2002.

[18] Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 36(2), September 2002.

[19] Evan Broder. su's "authentication failure" error should help users discover sudo. https://bugs.launchpad.net/ubuntu/+source/shadow/+bug/667509, October 2010.

[20] Declan Butler. When google got flu wrong. *Nature*, 494(7436):155–156, February 2013.

[21] Ed H. Chi, Peter Pirolli, Kim Chen, and James Pitkow. Using information scent to model user information needs and actions and the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '01, New York, NY, USA, 2001. ACM.

[22] Eun Kyoung Choe, Nicole B. Lee, Bongshin Lee, Wanda Pratt, and Julie A. Kientz. Understanding quantified-selfers' practices in collecting and exploring personal data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 1143–1152, New York, NY, USA, 2014. ACM.

[23] Hyunyoung Choi and Hal Varian. Predicting the present with google trends. *Economic Record*, 88:2–9, 2012.

[24] Charles L. A. Clarke, Gordon V. Cormack, and Thomas R. Lynam. Exploiting redundancy in question answering. In *Proceedings of the 24th annual international ACM SIGIR conference on research and development in information retrieval*, SIGIR '01, page 358–365, New York, NY, USA, 2001. ACM.

[25] Charles L. A. Clarke, Gordon V. Cormack, and Elizabeth A. Tudhope. Relevance ranking for one to three term queries. *Information Processing and Management*, 36(2):291 – 311, 2000.

[26] Purvis Crystale Cooper, P. Kenneth Mallon, Steven Leadbetter, A. Lori Pollack, and A. Lucy Peipins. Cancer internet search activity on a major search engine, united states 2001-2003. *J Med Internet Res*, 7(3):e36, July 2005.

[27] Patrick Copeland, Raquel Romano, Tom Zhang, Greg Hecht, Dan Zigmond, and Christian Stefansen. Google disease trends: an update. In *International Society of Neglected Tropical Diseases 2013*, page 3, 2013.

[28] Apple Corporation. Apple - apple watch - technology. https://www.apple.com/ca/watch/technology/, 2015.

[29] Google Corporation. Using keyword matching options. https://support.google.com/adwords/answer/2497836?hl=en.

[30] Google Corporation. Features: Google suggest. http://www.google.com/support/websearch/bin/answer.py?hl=en&answer=106230, 2010.

[31] Google Corporation. Google AdWords keyword tool. https://adwords.google.com/select/KeywordToolExternal?forceLegacy=true, 2010.

[32] Google Corporation. Google suggest : FAQ. http://labs.google.com/intl/en/suggestfaq.html, 2010.

[33] Google Corporation. Google custom search APIs and tools. http://code.google.com/apis/customsearch/, April 2011.

[34] Google Corporation. Close variant matching for all exact and phrase keywords, August 2014.

[35] Google Corporation. Explore google trending search topics with google trends. https://www.google.ca/trends/, 2015.

[36] Microsoft Corporation. Microsoft customer experience improvement program. http://www.microsoft.com/products/ceip/EN-US/default.mspx, February 2009.

[37] Microsoft Corporation. Microsoft word online - work together on word documents. https://office.live.com/start/Word.aspx, 2015.

[38] Microsoft Corporation. MSDN forums – microsoft. https://social.msdn.microsoft.com/forums/, 2015.

[39] Microsoft Corporation. Office training and tutorials - office support. https://support.office.com/en-US/article/Office-training-and-tutorials-b8f02f81-ec85-4493-a39b-4c48e6bc4bfb, 2015.

[40] Sarah Davison. Create a holiday e-Card in gimp, December 2008.

[41] Morgan Dixon and James Fogarty. Prefab: Implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 1525–1534, New York, NY, USA, 2010. ACM.

[42] Michael Ekstrand, Wei Li, Tovi Grossman, Justin Matejka, and George Fitzmaurice. Searching for software learning resources using application context. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, page 195–204, New York, NY, USA, 2011. ACM.

[43] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, January 2006.

[44] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proc. ACL'05*, page 363–370, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[45] Fitbit. Sleep tracking FAQs. http://help.fitbit.com/articles/en_US/Help_article/Sleep-tracking-FAQs, November 2014.

[46] Adam Fourney, Ben Lafreniere, Parmit Chilana, and Michael Terry. InterTwine: creating interapplication information scent to support coordinated use of software. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 429–438, New York, NY, USA, 2014. ACM.

[47] Adam Fourney, Ben Lafreniere, Richard Mann, and Michael Terry. "Then click ok!": extracting references to interface elements in online documentation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 35–38, New York, NY, USA, 2012. ACM.

[48] Adam Fourney, Richard Mann, and Michael Terry. Characterizing the usability of interactive applications through query log analysis. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, page 1817–1826, New York, NY, USA, 2011. ACM.

[49] Adam Fourney, Richard Mann, and Michael Terry. Query-feature graphs: Bridging user vocabulary and system functionality. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, page 207–216, New York, NY, USA, 2011. ACM.

[50] Adam Fourney and Meredith Ringel Morris. Enhancing technical Q&A forums with CiteHistory. In *Proceedings of the Seventh International Conference on Weblogs and Social Media (ICWSM '13)*, ICWSM '13, 2013.

[51] Adam Fourney and Michael Terry. Mining online software tutorials: Challenges and open problems. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '14, page 653–664, New York, NY, USA, 2014. ACM.

[52] Adam Fourney, Ryen W. White, and Eric Horvitz. Exploring Time-Dependent concerns about pregnancy and childbirth from search logs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 737–746, New York, NY, USA, 2015. ACM.

[53] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, November 1987.

[54] Ryan Gavin. Bing brings the world's knowledge straight to you with insights for office | search blog. http://blogs.bing.com/search/2014/12/10/bing-brings-the-worlds-knowledge-straight-to-you-with-insights-for-office/, December 2014.

[55] Mike Gikas. Lab tests: Why consumer reports can't recommend the iPhone 4. *Consumer Reports*, July 2010.

[56] Jeremy Ginsberg, Matthew H. Mohebbi, Rajan S. Patel, Lynnette Brammer, Mark S. Smolinski, and Larry Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457:1012–1014, February 2009.

[57] Sharad Goel, Jake M. Hofman, Sébastien Lahaie, David M. Pennock, and Duncan J. Watts. Predicting consumer behavior with web search. *Proceedings of the National Academy of Sciences*, 107(41):17486–17490, October 2010.

[58] Max Goldman and Robert C. Miller. Codetrail: Connecting source code and web resources. *J. Vis. Lang. Comput.*, 20(4):223–235, August 2009.

[59] Qi Guo, Eugene Agichtein, Charles L. A. Clarke, and Azin Ashkan. In the mood to click? towards inferring receptiveness to search advertising. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '09, page 319–324, Washington, DC, USA, 2009. IEEE Computer Society.

[60] Björn Hartmann, Mark Dhillon, and Matthew K. Chan. HyperSource: bridging the gap between source and code-related web sites. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, New York, NY, USA, 2011. ACM.

[61] Steffen Hedegaard and Jakob Grue Simonsen. Extracting usability and user experience information from online user reviews. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 2089–2098, New York, NY, USA, 2013. ACM.

[62] Jeff Hendy, Kellogg S. Booth, and Joanna McGrenere. Graphically enhanced keyboard accelerators for GUIs. In *Proceedings of Graphics Interface 2010*, GI '10, page 3–10, Toronto, Ont., Canada, 2010. Canadian Information Processing Society.

[63] David M. Hilbert and David F. Redmiles. Extracting usability information from user interface events. *ACM Computing Surveys*, 32(4):384–421, 2000.

[64] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, page 168–177, New York, NY, USA, 2004. ACM.

[65] Jeff Huang and Efthimis N. Efthimiadis. Analyzing and evaluating query reformulation strategies in web search logs. In *Proc CIKM '09*, page 77–86, New York, NY, USA, 2009. ACM.

[66] Amy Hurst, Scott E. Hudson, and Jennifer Mankoff. Dynamic detection of novice vs. skilled use without a task model. In *Proc CHI '07*, page 271–280, New York, NY, USA, 2007. ACM.

[67] Melody Y. Ivory and Marti A Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, 33(4):470–516, December 2001.

[68] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32(1):5–17, April 1998.

[69] Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Inf. Process. Manage.*, 36(2):207–227, January 2000.

[70] Maryam Kamvar, Melanie Kellar, Rajan Patel, and Ya Xu. Computers and iphones and mobile phones, oh my!: a logs-based comparison of search users on different devices. In *Proceedings of the 18th international conference on World wide web*, WWW '09, page 801–810, New York, NY, USA, 2009. ACM.

[71] Melanie Kellar, Carolyn Watters, and Michael Shepherd. A field study characterizing web-based information-seeking tasks. *J. Am. Soc. Inf. Sci. Technol.*, 58(7):999–1018, 2007.

[72] Md Adnan Alam Khan, Volodymyr Dziubak, and Andrea Bunt. Exploring personalized command recommendations based on information found in web documentation. In *Proceedings of the 20th International Conference on Intelligent User Interfaces*, IUI '15, page 225–235, New York, NY, USA, 2015. ACM.

[73] Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J. Guo, Robert C. Miller, and Krzysztof Z. Gajos. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, page 4017–4026, New York, NY, USA, 2014. ACM.

[74] Soo-Min Kim and Eduard Hovy. Automatic identification of pro and con reasons in online reviews. In *Proceedings of the COLING/ACL on Main conference poster sessions*, COLING-ACL '06, page 483–490, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[75] Andrew Ko. Mining whining in support forums with frictionary. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, page 191–200, New York, NY, USA, 2012. ACM.

[76] Mozilla Labs. Ubiquity: An experimental interface based on natural language input. https://mozillalabs.com/ubiquity/, April 2011.

[77] Benjamin Lafreniere. *Task-Centric User Interfaces.* PhD, University of Waterloo, Waterloo, ON, Canada, April 2014.

[78] Benjamin Lafreniere, Andrea Bunt, Matthew Lount, Filip Krynicki, and Michael A. Terry. AdaptableGIMP: designing a socially-adaptable interface. In *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*, UIST '11 Adjunct, page 89–90, New York, NY, USA, 2011. ACM.

[79] Benjamin Lafreniere, Andrea Bunt, and Michael Terry. Task-Centric interfaces for Feature-Rich software. In *OzCHI 2014*, New York NY, December 2014. ACM.

[80] Benjamin Lafreniere, Andrea Bunt, John S. Whissell, Charles L. A. Clarke, and Michael Terry. Characterizing large-scale use of a direct manipulation application in the wild. In *Proceedings of Graphics Interface 2010*, GI '10, page 11–18, Toronto, Ont., Canada, Canada, 2010. Canadian Information Processing Society.

[81] Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. Community enhanced tutorials: Improving tutorials with multiple demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 1779–1788, New York, NY, USA, 2013. ACM.

[82] J. Laherrère and D. Sornette. Stretched exponential distributions in nature and economy: "fat tails" with characteristic scales. *The European Physical Journal B - Condensed Matter and Complex Systems*, 2(4):525–539, 1998.

[83] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):pp. 159–174, 1977.

[84] Gierad Laput, Eytan Adar, Mira Dontcheva, and Wilmot Li. Tutorial-based interfaces for cloud-enabled applications. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, page 113–122, New York, NY, USA, 2012. ACM.

[85] Tessa Lau, Clemens Drews, and Jeffrey Nichols. Interpreting written How-To instructions. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, IJCAI'09, pages 1433–1438, 2009.

[86] Ian Li, Anind Dey, and Jodi Forlizzi. A stage-based model of personal informatics systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 557–566, New York, NY, USA, 2010. ACM.

[87] Greg Little and Robert C. Miller. Translating keyword commands into executable code. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, page 135–144, New York, NY, USA, 2006. ACM.

[88] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In Charu C. Aggarwal and ChengXiang Zhai, editors, *Mining Text Data*, pages 415–463. Springer US, January 2012.

[89] Justin Matejka, Tovi Grossman, and George Fitzmaurice. Patina: Dynamic heatmaps for visualizing application usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 3227–3236, New York, NY, USA, 2013. ACM.

[90] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. Community-Commands: command recommendations for software applications. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, page 193–202, New York, NY, USA, 2009. ACM.

[91] George A. Miller. WordNet: a lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[92] Elizabeth L. Murnane and Scott Counts. Unraveling abstinence and relapse: Smoking cessation reflected in social media. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, page 1345–1354, New York, NY, USA, 2014. ACM.

[93] Jakob Nielsen. 10 usability heuristics for user interface design. http://www.nngroup.com/articles/ten-usability-heuristics/, January 1995.

[94] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, page 249–256, New York, NY, USA, 1990. ACM.

[95] Don Norman. The next UI breakthrough: command lines. *interactions*, 14(3):44–45, May 2007.

[96] Donald A. Norman. Cognitive engineering. User Centered System Design: New Perspectives on Human-computer Interaction. Lawrence Erlbaum Associates, 1986.

[97] Donald A. Norman and Stephen W. Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction.* L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1986.

[98] Hartmut Obendorf, Harald Weinreich, Eelco Herder, and Matthias Mayer. 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*, CHI '07, page 597–606, New York, NY, USA, 2007. ACM.

[99] Michael J. Paul, Ryen. W. White, and Eric Horvitz. Search and breast cancer: On disruptive shifts of attention over life histories of an illness. Technical Report MSR-TR-2014-144, Microsoft Research, 2014.

[100] Michael J. Paul, Ryen W. White, and Eric Horvitz. Diagnoses, decisions, and outcomes: Web search as decision support for cancer. In *24th International World Wide Web Conference (WWW 2015)*, WWW '15, Florence, Italy, 2015.

[101] Pablo Pedemonte, Jalal Mahmud, and Tessa Lau. Towards automatic functional test execution. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, IUI '12, page 227–236, New York, NY, USA, 2012. ACM.

[102] Peter Pirolli and Stuart Card. Information foraging. *Psychological Review*, 106(4):643–675, 1999.

[103] Peter G. Polson, Clayton Lewis, John Rieman, and Cathleen Wharton. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies*, 36(5):741 – 773, 1992.

[104] Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. Pause-and-play: Automatically linking screencast video tutorials with applications. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, page 135–144, New York, NY, USA, 2011. ACM.

[105] Vidya Ramesh, Charlie Hsu, Maneesh Agrawala, and Bjoern Hartmann. ShowMeHow: translating user interface instructions between similar applications. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, New York, NY, USA, 2011. ACM.

[106] Matthew Richardson. Learning about the world through long-term query logs. *ACM Trans. Web*, 2(4), 2008.

[107] Daniel E. Rose and Danny Levinson. Understanding user goals in web search. In *Proc WWW '04*, page 13–19, New York, NY, USA, 2004. ACM.

[108] Peter J. Rousseeuw and Annick M. Leroy. Other techniques for simple regression. In *Robust Regression and Outlier Detection*, pages 67–78. John Wiley & Sons, October 2003.

[109] Stuart Russell and Peter Norvig. Artificial intelligence a modern approach. page 718. Prentice Hall, 2nd edition, 2003.

[110] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[111] Mark Sanderson and Susan Dumais. Examining repetition in user search behavior. In *Proceedings of the 29th European conference on IR research*, ECIR'07, page 597–604, Berlin, Heidelberg, 2007. Springer-Verlag.

[112] Paricia Correia Saraiva, Edleno Silva de Moura, Novio Ziviani, Wagner Meira, Rodrigo Fonseca, and Berthier Riberio-Neto. Rank-preserving two-level caching for scalable search engines. In *Proc SIGIR '01*, page 51–58, New York, NY, USA, 2001. ACM.

[113] Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Q2C@UST: our winning solution to query classification in KDDCUP 2005. *SIGKDD Explor. Newsl.*, 7(2):100–110, December 2005.

[114] Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Query enrichment for web-query classification. *ACM Trans. Inf. Syst.*, 24(3):320–352, July 2006.

[115] Blacktree Software. Quicksilver: OS x at your fingertips. http://qsapp.com/, April 2011.

[116] Betsy Sparrow, Jenny Liu, and Daniel M. Wegner. Google effects on memory: Cognitive consequences of having information at our fingertips. *Science*, 333(6043):776–778, 2011.

[117] Anselm Strauss and Juliet Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* Sage Publications, 3rd edition edition, 2008.

[118] Huifeng Tang, Songbo Tan, and Xueqi Cheng. A survey on sentiment detection of reviews. *Expert Systems with Applications*, 36(7):10760–10773, September 2009.

[119] Jaime Teevan, Eytan Adar, Rosie Jones, and Michael A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval*, SIGIR '07, page 151–158, New York, NY, USA, 2007. ACM.

[120] Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, page 449–456, New York, NY, USA, 2005. ACM.

[121] Stefanie Tellex, Boris Katz, Jimmy Lin, Aaron Fernandes, and Gregory Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of the 26th annual international ACM SIGIR conference on research and development in informaion retrieval*, SIGIR '03, page 41–47, New York, NY, USA, 2003. ACM.

[122] Michael Terry, Matthew Kay, Brad Van Vugt, Brandon Slack, and Taehyun Park. Ingimp: introducing instrumentation to an end-user open source application. In *Proc CHI '08*, CHI '08, page 607–616, New York, NY, USA, 2008. ACM.

[123] Mikalai Tsytsarau and Themis Palpanas. Survey on mining subjective data on the web. *Data Mining and Knowledge Discovery*, 24(3):478–514, May 2012.

[124] Andrew Turpin and Falk Scholer. User performance versus precision measures for simple search tasks. In *Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval*, SIGIR '06, page 11–18, New York, NY, USA, 2006. ACM.

[125] Daniel M. Wegner. Transactive memory: A contemporary analysis of the group mind. In Brian Mullen and George R. Goethals, editors, *Theories of Group Behavior*, Springer Series in Social Psychology, pages 185–208. Springer New York, 1987.

[126] Tim Weninger and William H. Hsu. Text extraction from the web via Text-to-Tag ratio. *Database and Expert Systems Applications, International Workshop on*, pages 23–28, 2008.

[127] Robert West, Ryen W. White, and Eric Horvitz. From cookies to cooks: Insights on dietary patterns via analysis of web usage logs. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, page 1399–1410, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[128] R. W. White, R. Harpaz, N. H. Shah, W. DuMouchel, and E. Horvitz. Toward enhanced pharmacovigilance using Patient-Generated data on the internet. *Clinical Pharmacology & Therapeutics*, 96(2):239–246, August 2014.

[129] Ryen W. White, Nicholas P. Tatonetti, Nigam H. Shah, Russ B. Altman, and Eric Horvitz. Web-scale pharmacovigilance: listening to signals from the crowd. *Journal of the American Medical Informatics Association*, pages amiajnl–2012–001482, March 2013. PMID: 23467469.

[130] I.H. Witten and T.C. Bell. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *Information Theory, IEEE Transactions on*, 37(4):1085 – 1094, July 1991.

[131] Koji Yatani, Michael Novati, Andrew Trusty, and Khai N. Truong. Analysis of adjective-noun word pair extraction methods for online review summarization. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, page 2771–2776, Barcelona, Catalonia, Spain, 2011. AAAI Press.

[132] Koji Yatani, Michael Novati, Andrew Trusty, and Khai N. Truong. Review spotlight: a user interface for summarizing user-generated reviews using adjective-noun word pairs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 1541–1550, New York, NY, USA, 2011. ACM.