QFRecs - Recommending Features in Feature-Rich Software based on Web Documentation

by

Md Adnan Alam Khan

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

MASTER OF SCIENCE

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
March 2015

Thesis advisor Andrea Bunt

Author Md Adnan Alam Khan

QFRecs - Recommending Features in Feature-Rich Software based on Web Documentation

Abstract

Prior work on command recommendations for feature-rich software has relied on data supplied by a large community of users to generate personalized recommendations. In this work, I explored the feasibility of using an alternative data source: web documentation. Specifically, the proposed approach uses QF-Graphs, a previously introduced technique that maps higher-level tasks (i.e., search queries) to commands referenced in online documentation. The proposed approach uses these command-to-task mappings as an automatically generated plan library, enabling our prototype system to make personalized recommendations for task-relevant commands. Through both offline and online evaluations, I explored potential benefits and drawbacks of this approach.

Acknowledgments

I am grateful to almighty God for giving me the strength that helped me to pursue my graduate degree staying away from my family for about two and half years.

I would like to thank my supervisor, Dr. Andrea Bunt, for her support and guidance throughout the entire duration of my masters program at UofM. I would also like to thank her for the financial support during these two and half years. Further, I am grateful to the department of computer science, UofM for the funding that I received in the form of guaranteed funding package (GFP) and would again like to thank Dr. Bunt for the assistance provided.

I would like to thank my committee members, Dr. Yang Wang and Dr. Jason Morrison, for their precious time and feedbacks. I am also grateful to Dr. James E. Young and Dr. Pourang Irani for their feedbacks on my work in our HCI lab meetings.

I am thankful to all of my lab-mates in the HCI lab for their support in many ways. Volodymyr Dziubak deserves a special mention for his support, especially before the IUI paper submission deadline. I would like to thank him again for presenting the paper in the conference instead of me. Thanks to Khaled Hasan, Barrett, Anik, Danial, Stela, joel and Noor for their ideas, comments and feedbacks on my work.

Finally, I would like to acknowledge the love and support of my family staying overseas and dedicate this thesis to my loving wife, Mahamuda Sultana for her patience and support.

Publications

Some ideas and figures in this thesis have appeared previously in the following publication by the author:

Md Adnan Alam Khan, Volodymyr Dziubak, and Andrea Bunt. Exploring personalized command recommendations based on information found in web documentation. In *Proceedings of the ACM Conference on Intelligent User Interfaces (IUI)*, 2015, 10 pages. to appear.

To my loving wife, "Happy"

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Feature-rich software applications (e.g., image-manipulation programs, spreadsheet software and statistical analysis packages) are highly versatile, in part owing to the hundreds (or even thousands) of commands or features that they make available. At the same time, this high volume of commands can make feature-rich software difficult for users to master [23]. For example, studies of long-term application use have shown that most users have fairly limited command vocabularies [21], typically using much less than one quarter of the available command set (e.g., [16, 36]). Moreover, most feature-rich software applications introduce more and more features with each new release. The presence of a large number of features in such applications introduces two problems: (1) they make it hard for a user to become aware of all available features and (2) they result in an increase in the size and complexity of the graphical user interfaces (GUIs) of the applications. These problems lead users to both frustration [29] and decreased performance [4].

One promising approach to increasing a user's awareness of the available command or feature set is to present them with intelligently generated, personalized command recommendations (e.g., [36, 37, 40]). Central to this approach is an understanding of the potential relationships between commands – knowledge that enables an intelligent system to recommend commands that could complement those currently being used. Most prior work has extracted this relevancy information from community usage logs, for example, by applying collaborative filtering algorithms to a large corpus of logged data (e.g., [36, 37, 40]). While these usage-data centric approaches have shown a great deal of promise, their practical success hinges on the existence of a large community of users who are willing to upload their usage data to a central repository.

On the other hand, to deal with the problem of interface complexity, introduced by the presence of a lot of features or commands in the interface, researchers have proposed personalized interfaces where interfaces are customized to better suit users' need [3, 11, 27]. Personalized interfaces aim to enhance a user's performance by providing access to a subset of available features. Although research has shown that personalized interfaces can improve users' task performance, they can decrease users' awareness of system features [12, 13].

My thesis investigates whether online resources (e.g., tutorials, online documentation etc.) could be used as data source to generate personalized recommendations to increase user's feature awareness. The use of online resources has the potential to eliminate prior recommendation system's dependency on community usage log. Further, the use of online resources has not been previously explored for feature recommendations. In my thesis, I also investigate different presentation techniques focused on how the personalized recommendations could be incorporated into the graphical user interface (GUI) with the aim of minimizing interface complexity. Specifically, my thesis involves following research questions:

- Does generating personalized feature recommendations based on collections of features found in online resources (e.g., web tutorials, online learning resources etc.) provide useful recommendations that have the potential to enhance feature-awareness?
- How can we effectively present relevant unknown or less frequently used system features into the application's user interface without introducing additional interface complexity?

To answer these questions I designed and developed the QFRecs system, an alternative approach to personalized command recommendations that uses command-to-task mappings mined from online documentation. Specifically, the QFRecs system uses Query-Feature Graphs (QF-Graphs) [17], a technique that maps common internet search queries to collections of interface elements referenced in the resulting online documentation. Within the context of the GNU Image Manipulation Program (GIMP), I illustrate how the proposed recommender system uses a QF-Graph as an automatically generated plan library. Through offline experimentation with previously collected usage data, I explore this documentation-centric approach's potential to both increase the user's command awareness through task-relevant recommendations, and to enable a system to provide efficient access to needed commands. Further, in a controlled laboratory experiment, I also explore the impact of two alternative recommendation presentation techniques on immediate task performance as well as on incidental awareness of relevant commands not selected during the primary task [12].

The structure of the remainder of my thesis is as follows. In chapter 2, I explore related work. In chapter 3, I describe the process and implementation of the proposed recommender system which I call “QFRecs”. In chapter 4, I report the offline analysis I performed to evaluate the QFRecs system’s performance and present the results. In chapter 5, I describe the study I conducted to evaluate two alternative recommendation presentation techniques and present the results of the study. I conclude in chapter 6 by summarizing the contributions of my thesis and discussing its limitations and possible feature directions.

Chapter 2

Related Work

Assisting novice as well as expert users during their learning phase with feature-rich software has been a widely studied topic. Assistance can be provided through an engaging improved tutorial interface, through in-application help, or through task-specific GUI customization. Therefore, I explore this research work from three main areas. First, I discuss the research work aimed at designing engaging step-by-step tutorials to facilitate application learning. After that, I describe prior research work that is focused on in-application assistance. Then, I discuss various approaches that prior research has proposed for user interface customization in feature-rich applications. In addition to these three areas, I also describe the research work that is focused on extracting information on software use from web documents.

2.1 Improved tutorial interfaces

Prior work has shown that tutorials play a large role in feature-rich application learning. Users of feature-rich software applications often consult documentation, web tutorials, video tutorials, and supporting resources to learn and understand feature-rich software's usage and capabilities. Users consult tutorials and supporting resources mainly for "in-task help", when users need to accomplish a part of a task immediately [29]. Authoring effective tutorials and supporting resources can be difficult. Therefore, prior research has focused on automated or semi-automated tutorial authoring. Automated or semi-automated tutorials focus mostly on improving the utility of the generated tutorials through combining static tutorials (e.g., image and text based tutorials) and video tutorials with dynamic interactions (e.g., [5, 20]).

Research has also focused on creating novel and engaging tutorial formats to help user to learn about feature-rich software (e.g., [10, 24]). Grossman *et al.* [24] introduced a system named *Chronicle* that captures the entire workflow history of a tutorial and allows novice users to traverse the captured workflow so that users can better understand the segment of interest of the workflow. Their system also enables users to find out more about the tools and settings used in that segment of interest. Another example of an interactive learning tutorial is Sketch-Sketch Revolution [10], an in-product, content centric interactive system that allows a user to follow an already existing workflow of an expert user. Along with helping a user to gain the confidence that they can regenerate expert content, Sketch-Sketch Revolution also helps users to interact effectively with the application's user interface.

Further, research has introduced many game-based novel and engaging tutorial systems to help users to learn about new tools and techniques. Examples of such tutorial system include Jigsaw [8] and GamiCAD [35]. GamiCAD, proposed by Li *et al.* [35], is an in-product interactive tutorial system to help first-time AutoCAD users to become familiar with new features and improve their performance with AutoCAD. In GamiCAD, new users both learn and improve their performance through real time feedbacks from the gamified tutorials for their successes and failures. On the other hand, Jigsaw, proposed by Dong *et al.* [8], is a discovery-based interactive tutorial to help users learn new tools and techniques in Adobe Photoshop. In Jigsaw, users learn about new tools through solving jigsaw puzzles using Adobe Photoshop tools.

Research has also explored ways to harness crowds or community contributions to improve the utility of web-based tutorials through integrating community refinements [4], by augmenting tutorials with community demonstrations [33], and by using crowds to help segment video tutorials into steps to permit easier tutorial navigation [28]. To integrate community refinement, Bunt *et al.* [4] introduced the *TaggedComments* system that aims to enhance the role of the comments to create more engaging tutorials. *TaggedComments* allows users to tag tutorial comments with the appropriate section of the tutorial content and provide direct access to all the comments from the tutorial content. Further, Lafreniere *et al.* [33] proposed *FollowUs*, a web-tutorial system that integrates the application within the tutorial and allows improvement of the tutorial contents through community contribution. Specifically, *FollowUs* captures the workflow of a tutorial from expert users (community members) and embed the captured video within the original tutorial for other users to follow along with these community contributed videos.

Another example of tutorial enhancement through community contribution is the work proposed by Kim *et al.* [28] which aims to provide step-by-step annotation of how-to videos with the aim of helping users to navigate easily through the video timeline (e.g. skip unwanted segments, go to the video segment containing segment of interest). They proposed *ToolScape*, a step-aware interactive video player that displays step-by-step annotations along with intermediate result thumbnails in the video timeline. Their approach also introduced a novel crowdsourcing workflow to add step-by-step annotations to the existing how-to videos in YouTube.

All the above described tutorial enhancement approaches either aim to create more engaging tutorials to help users with their workflows or try to increase utility of the existing tutorials through community contribution. Therefore, the primary focus of these tutorial interfaces is not the improvement of users' feature awareness. In contrast, my proposed system aims to improve users feature awareness through recommending useful features based meta-data found in these available tutorials.

2.2 In-Application Assistance

As is the case with my work, prior research has also sought to improve software learning from within the application itself. The *Lumiere* Project is an early example of providing assistance to application users using Bayesian user modeling [27]. Using a hand-crafted model, the *Lumiere* Project attempts to predict a user's future action based on his observed actions and queries. Further, in terms of assisting a user during the primary task, Grossman *et al.* [22] proposed *ToolClips*, a system that provides contextual video along with traditional tooltips to improve learnability of feature-rich software. In other work, Matejka *et al.* [38] proposed IP-QAT, a community based in-product question answering system that shows contextually relevant user posts to assist other users during their primary tasks.

Most relevant to our work are systems that provide unobtrusive personalized command recommendations by mining large corpuses of community usage data. The *OWL* system, proposed by Linton *et al.* [37], is one of the first such systems to recommend relevant unknown features and present those features in the user interface to enhance users' feature awareness. The *OWL* system uses long-term usage history of individuals within an organization to support an individual learning through feature recommendations. One of the drawbacks of the OWL system is the following assumption: users within an organization exhibit similar command usage patterns. This assumption may fail due to the variance in expertise level within a user group.

Research has also focused on collaborative filtering based command recommendations. For example, *CommunityCommands* [36, 40] uses collaborative filtering algorithms on a large corpus of usage data to generate personalized command recommendations. *CommunityCommands* suggests unused or less frequently used features in AutoCAD based on explicitly collected usage data from a large AutoCAD user community, which means their approach requires a large active user community to capture the usage data. Further, Emerson *et al.* [42] extended the approach of *CommunityCommands* to the Eclipse integrated development environment (IDE) to suggest useful development features (e.g., code formatter, build tools, etc.) to enhance the software developers' fluency. Another example of community-dependent approach is *Patina* [39], which provides subtle command recommendations by overlaying heat maps on the interface to highlight commands commonly used by the community.

Thus, researchers have focused on feature recommendations based on users' long-term application usage data collected from large user communities. In my thesis, I have proposed the QFRecs system that uses online resources (e.g., web tutorials, online documentation) to generate feature recommendations in order to eliminate the need for large user communities contributing data. The use of online resources has not been explored as an alternative way to build the underlying knowledge base of feature recommender systems. Therefore, this research extends the prior work by exploring a new data source for task-relevant command recommendations.

2.3 Providing Efficient Access to Needed Commands

Modern software applications (e.g., Photoshop, GIMP, and AutoCAD) offer hundreds of system features, which are accessed by their user communities. Prior research has shown that users of feature-rich software tend to use only a small subset of the available command set. For example, Lafreniere *et al.* [32] found that at least 90% of the users of the GNU Image Manipulation Program (GIMP) use only 27 of 352 available commands. Li *et al.* [36] also reported that 90% of Autodesk's AutoCAD users use less than 90 commands out of thousands of available commands. Therefore, previous research has examined ways to provide more efficient access to needed commands (most frequently used commands) by reducing the search space. This research has proposed various approaches that ranged from personalized interfaces (e.g., [2],[3], [41]), to community-authored task-specific interfaces accessible through in-application keyword search [30, 31], to interfaces that adaptively promote commands according to recency and frequency information (e.g., [11, 19]).

In general, personalized interfaces attempt to adapt the interface elements of feature-rich software applications to better suit users' needs in order to improve users' interaction efficiency. Interfaces can be personalized through two opposing ways: (1) system-controlled (adaptive) and (2) user-controlled (adaptable), which differs in terms of who performs the customization (the user or the system). Researchers found that, in most cases, user-controlled personalization has performed better than system-controlled personalization [11, 18]. For example, Findlater *et al.* [11] compared static, adaptive and adaptable menus and found that adaptable menus are preferred over the others and adaptable menus performed better than adaptive menus under certain conditions.

In addition to system or user-controlled personalization, prior research has also focused on reduced functionality interfaces. For example, McGrenere *et al.* [41] evaluated a customized interface of feature-rich software that enables a user to toggle between a feature-reduced interface, which contains an adaptably chosen subset of features and the full-featured default interface. Their results showed that the adaptable prototype with toggle capability results better user satisfaction than the default interface. Further, Lafreniere *et al.* [30] proposed *AdaptableGIMP*, a crowd-based approach for adaptable interface for the GNU graphical image manipulation program (GIMP). In their prototype interface, a user can access task-specific interfaces that are created by the application's user community. Upon selecting a task, a reduced functionality interface tailored for that specific task is provided to the user. Combining the adaptive and the adaptable approach, Bunt *et al.* [3] proposed a mixed-initiative approach for feature-reduced interfaces which provides customization suggestions (adaptively) to maximize the user's customization performance but the control of customization remains in the user's hands (adaptable). The result of their evaluation showed that the mixed-initiative approach improves task performance and is preferred over a purely adaptable approach. However, researchers also found that users often fail to customize their interfaces due to the lack of proper customization mechanisms in adaptable interfaces [2]. Further, research has also provided evidence that interface personalization through reduced functionality interfaces diminish a user's ability to learn new features by drawing attention to only a subset of features [13].

Apart from reduced functionality interfaces, a number of researchers have focused on adaptation techniques that adapt each feature's (e.g. menu items, toolbar items etc.) presentation in the interface. Those techniques can be divided into two main categories: (1) techniques that maintain spatial consistency but adapt visual and temporal presentation, and (2) techniques that adapt spatial position of the features in the interface. Examples of techniques that adapts spatial position of features include: Microsoft Smart menus, and Split menus [43]. Microsoft Smart menus hide the least useful menu items from initially visible ones to draw a user's attention to the frequently used menus. Split menus show top-n most frequent items in the top split to provide faster access to most frequently used items. The remaining menu items are displayed in the bottom split [43]. On the other hand, examples of techniques that alter visual and temporal representation include: Resizing or Morphing [7] Highlighting [18], and Ephemeral adaptation [14]. Resizing or morphing adaptively predicted menu items in long menus attempt to decrease a user's menu selection time [7]. Highlighting maintains spatial consistency and presents predicted items in a coloured background to draw a user's attention to these features [18]. Ephemeral adaptation uses abrupt appearance of predicted items followed by delayed onset of the non-predicted items to minimize a user's visual search time [14].

In case of my work, the feature recommendations generated by the QFRecs system are incorporated within the interface. Therefore, I explored interface personalization approaches proposed in prior research. Those interface personalization approaches aim to increase users' interaction efficiency with the interface rather than focusing on enhancing users' feature awareness. Further, prior research also showed that some forms of interface personalization (e.g., reduced functionality interfaces) can have negative impacts on users' feature awareness [12]. Therefore, I used ephemeral adaptation with highlighting [14] in the default menus instead of providing a reduced functionality interface.

2.4 Extracting Command-to-Task Mappings from Online Documentation

The QFRecs system uses online resources (e.g. tutorials, web documentation) as an alternative data source to community contributed usage data. Therefore, research work focused on automatic extraction of references to interface elements from online resources is important to my work. Automated recognition of references to interface elements from step-by-step tutorials has been an active research topic. For example, Fourney *et al.* [15] proposed a named-entity recognizer that identifies user interface elements (e.g. name of features, palettes, etc) from web page contents. Similar to Fourney *et al.*'s work, Laput *et al.* [34] proposed a conditional random field (CRF) based extractor for interface elements from step-by-step tutorials. Further, Ekstrand *et al.* proposed a context-aware search technique that extracts interface elements' names from each page of the search results. They presented the application icons of the extracted elements with each result so that the user can easily find the most relevant pages.

Given the prevalence of online documentation for feature-rich software, prior work has also explored the feasibility of using these resources to generate task-specific command groupings. For example, Fourney *et al.* [17] proposed Query-Feature Graphs (QF-Graphs) as a way to relate users' search queries for feature-rich applications to individual interface elements referenced in the resulting webpages. The QF-Graph is one of the components of the QFRecs system. Another example of the use of online documentation is *CommandSpace* [1], which uses web documentation to model the relationships between tasks and commands using a vector-space representation as opposed to a graph. *CommandSpace* used the CRF-based extractor proposed by Laput *et al.* [34] to model an application's domain language using deep learning techniques.

2.5 Summary

From the literature review, we see that significant research has been done on enhancing tutorial content to facilitate learning of feature-rich applications [4, 5, 8, 10]. However, these works mainly focused on either automating or semi-automating the tutorial generation process [5, 20] or introducing new and novel tutorial formats to support more engaging learning [4, 8, 10, 24, 35]. We see work focused on in-application assistance that aims to support users during their task [9, 22, 24, 39, 42]. Further, we found that a few of them focused on improving feature awareness through feature recommendations using community usage logs [36, 37, 38]. Therefore, my thesis work focuses on extending feature recommendation approaches using automatically extracted command-to-task mapping from online documentations.

Chapter 3

System Description

This chapter presents the QFRecs recommendation system, describing how QFRecs recommends novel and useful system features or commands to users based on data collected from online resources. The QFRecs system aims to recommend novel and useful features for two purposes. The first purpose is to enhance a user's feature awareness by visualizing novel and useful unknown features during the user's primary task. Its other purpose is to improve primary task performance through providing quick access to necessary features.

The input to the QFRecs system is a user's last n selected features and based on that, the QFRecs system generates novel and useful recommendations. After that, a feature-rich application adapts the interface based on the recommendations generated from the QFRecs system. More specifically, the QFRecs system provides an intelligent service that works below the feature-rich application's user interface and adapts the user interface based on the user's current context (e.g., previous feature selection history).

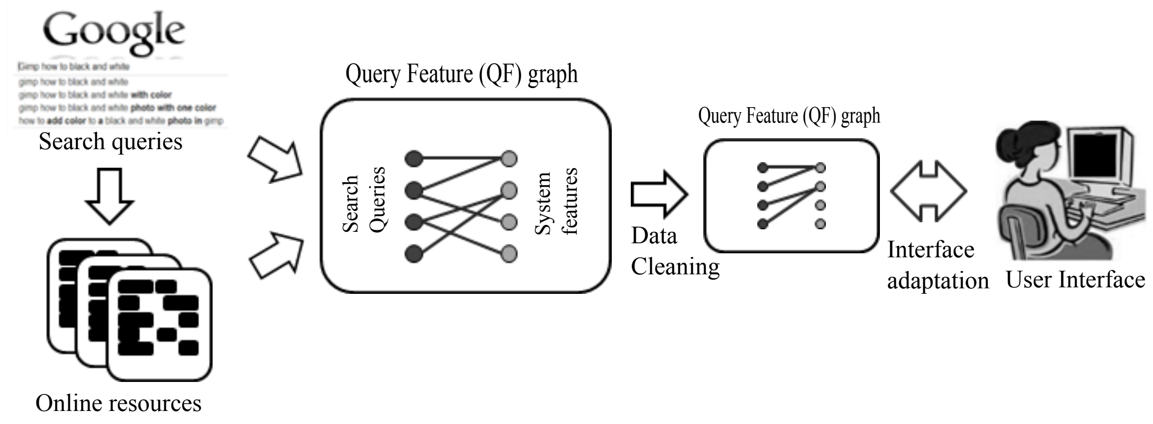

Google
Gimp how to black and white
gimp how to black and white
gimp how to black and white with color
gimp how to black and white photo with one color
how to add color to a black and white photo in gimp
Search queries
Online resources
Query Feature (QF) graph
Search Queries
System features
Data Cleaning
Query Feature (QF) graph
Interface adaptation
User Interface

Figure 3.1: The general architecture of the QFRecs system.

In the following sections, I present QFRecs, a feature recommender system built based on online resources. First, I describe the Query-Feature graph (QF-graph), which is a core component of the QFRecs system. Then I discuss how the information required to build a QF-graph can be collected from online resources (e.g. tutorials, online documentation, etc.). After that, I describe my data cleaning process to reduce the noise from the collected raw QF-graph data. Then, I follow this by describing the recommendation generation process using the noise-reduced QF-graph (generated from the cleaned raw QF-graph data). Finally, I discuss the types of recommendations that the system generates.

3.1 Query Feature Graph

A core component of the QFRecs system (see Figure 3.1), is the QF-graph originally introduced by Fourney *et al.* [17]. As illustrated in Figure 3.2, a QF-graph is a weighted bipartite graph, which associates internet search queries (i.e., natural language descriptions of high-level tasks [16], Figure 3.2 left side), with the features

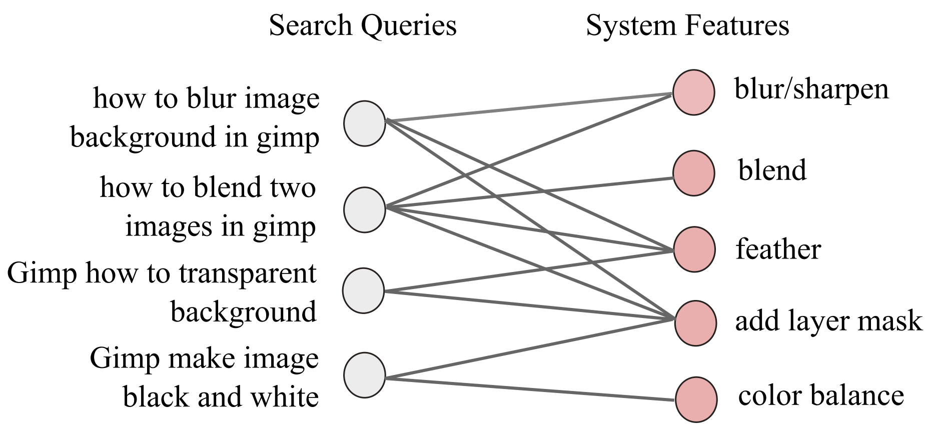
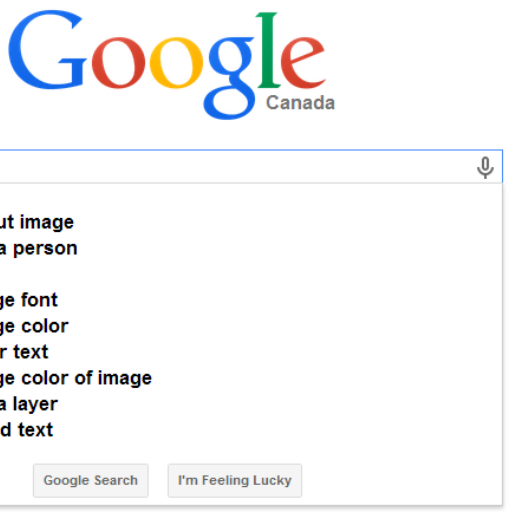

Search Queries
System Features
how to blur image background in gimp
how to blend two images in gimp
Gimp how to transparent background
Gimp make image black and white
blur/sharpen
blend
feather
add layer mask
color balance

Figure 3.2: An example of QF-graph for The GNU Image manipulation program (GIMP)

Table 3.1: Example connection weights between tasks and features.

	Blur/Sharpen	Blend	....	Color Balance
how to blur image background in GIMP	9.33	6.82	....	0
....	....	....	....	....
....	....	....	....	....
Gimp make image black and white	0	0	....	8.77

or commands of a target feature-rich application (Figure 3.2 right side) that appear in the result web pages. The weight of the edge between a query node and a feature node represents their strength of association (see Table 3.1) determined by the scoring function of the QAP (Question Answering Passage) algorithm [6].



gimp how to c
gimp how to crop
gimp how to cut out image
gimp how to crop a person
gimp how to clone
gimp how to change font
gimp how to change color
gimp how to center text
gimp how to change color of image
gimp how to crop a layer
gimp how to curved text

Figure 3.3: Google suggestions for keyword "how to c ".

3.2 Extracting Interface Elements from Online Resources

The process of building a QF-graph starts with collecting common user search queries issued for a target application using the CUTS method [16], which leverages the Google Suggest API. As an illustration, the search query suggestions for a partial query "gimp how to c"is shown in Figure 3.3. Fourney *et al.* [16] showed that these search query suggestions made by Google Suggest represent a unit of task that can be accomplished in GIMP such as "gimp how to crop", "gimp how to change fonts", etc. Using the CUTS method, all possible search queries for a target application are generated. These queries are then executed and the resulting webpages are examined for occurences of application-specific features (using a list of features that can be generated semi-automatically using the application's localization data).

As a starting point, I used the raw QF-graph data generated by Fourney *et al.* [17] for the GNU Graphics Manipulation Program (GIMP). The target application is GIMP for two reason: it allows for greater modification possibilities than propietary software, and it is possible to perform offline evalutations of QF-based approach using data previously collected through the “Ingimp”project [32, 44].

3.3 Data Preparation

In moving from the original QF-Graph concept to a concrete application, I discovered two sources of noise that impacted the quality of the prototype's recommendations requiring that I “clean”this original QF-Graph for it to be suitable for my purposes. As an overview, the cleaning process took part in two steps. First, I pruned tasks from the left-hand side of the QF-Graph that were not representative of high-level tasks. Second, since the goal was to recommend specific GIMP commands, I made sure that all system features on the right-hand side of the graph corresponded to actual elements in the GIMP interface. I describe these source of noise and the methods for cleaning the data in further detail here to illustrate some of the challenges of using the document-based approach to command recommendation in practice.

In terms of the high-level tasks (see Figure 3.2, left for examples), manual exploration of $12,311$ search queries used to build the original QF-graph revealed that many of the queries are not actually representative of high-level tasks. Therefore, I removed queries from this original graph if they met any of the following criteria: they contained digits (e.g., “gimp review 2010 ”, “gimp 2.6 fonts”, etc.), operating system names (e.g., “gimp for *macOS* x”), or the “vs”string (e.g., “gimp *vs* adobe”, “adobe illustrator *vs* gimp"). This removal process left me with 9889 queries. An alternative would have been to restrict the search queries to those containing the string "GIMP how to", however, this strategy appeared to result in unnecessary information loss (e.g., this would have removed over 90% of the queries).

The second source of noise concerned the features themselves (see Figure 3.2, right for example features in a QF-Graph). I found that not all the features extracted from the Web documentation mapped to commands in the actual GIMP interface. The primary cause was minor textual differences in labelling (e.g., "by color select"vs. "select by colour"). Most differences were resolved automatically by string matching using regular expressions. The matching rules for the regular expressions were crafted manually based on actual command names from the GIMP menus. Using regular expression and string matching, I mapped 569 features out of the original 617 distinct features in raw QF-Graph to the actual commands present in the GIMP interface. For example, "rectangular select"was mapped to "Rectangle Select"of the actual interface. Since the QF-Graph doesn't guarantee the presence of all available features of the GIMP user interface, I could not map the remaining features of the actual interface using the matching rules. For these remaining features, the mapping was done manually based on our knowledge of the target application.

Manual inspection also revealed that a feature's parent entity (e.g., menu name) was sometimes present in the raw data in addition to the command itself. This is because Web documentation often specifies a command's full path. For example, the line "Tools > Paint Tools > Paintbrush"would lead both "Tools"and "Paint Tools"to appear in the graph. In the cleaned QF-Graph, I included the child menu items only

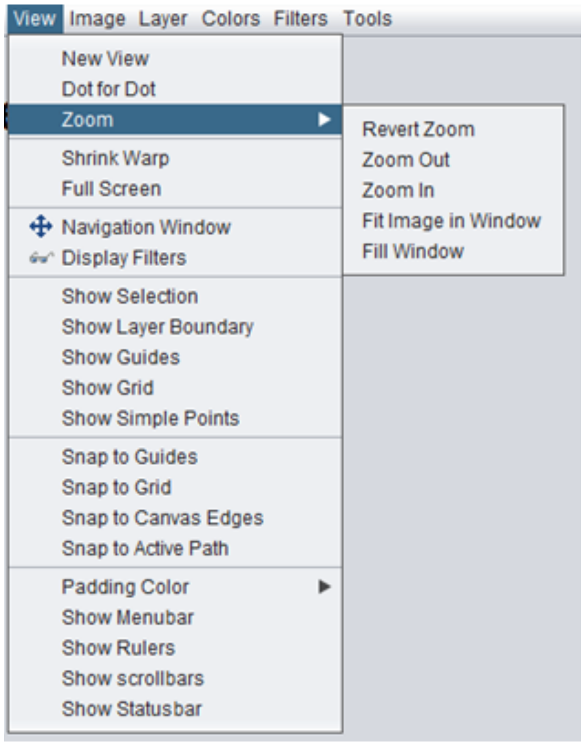


Figure 3.4: GIMP's view menu.

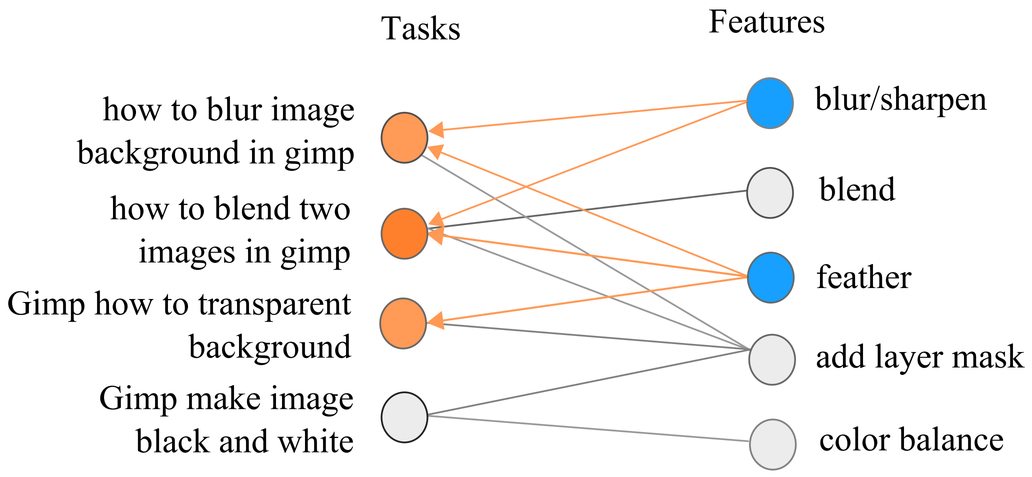
(e.g., "Paintbrush ").

On the other side, there were 67 features of the 417 actual GIMP interface elements not present in the raw QF-Graph. 27 of these missing features were under the "view"menu of the actual interface. The most probable reason could be that these features modify the presentation of the user interface of the GIMP than the input image (See Figure 3.4). Therefore, these features have less chance to be present in the actual tutorial's text. The remaining unmapped features were mostly from the "Filters"menu of the actual interface. Since the cleaned QF-graph doesn't contain 67 features of the actual GIMP interface, currently the QFRecs system is unable to recommand these features. In order to recommend these features to users, one way is to manually incorporate them into the QF-graph.

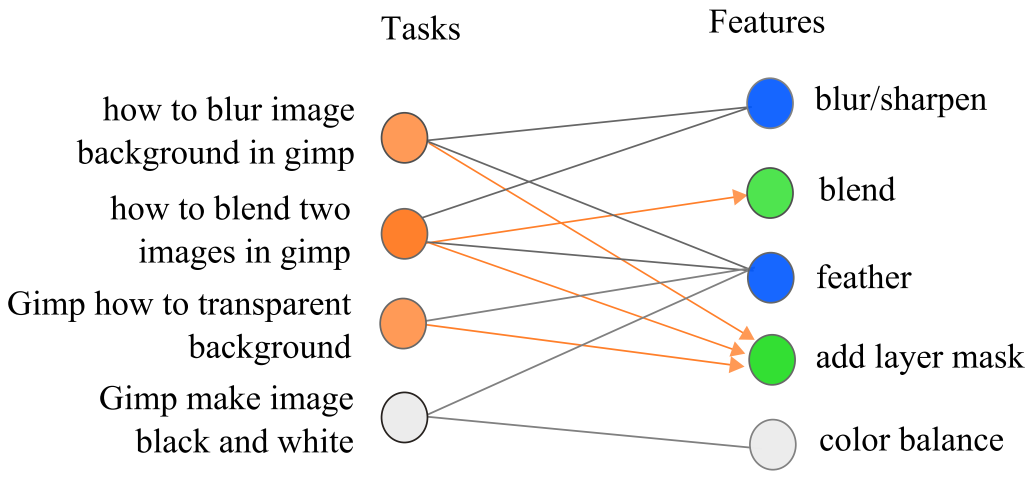
3.4 Generating Recommendations

To generate personalized, contextually relevant feature recommendations, the QFRecs system uses the cleaned QF-graph as an automatically generated plan library. Based on a user's last x command selections (e.g., the history size), the QFRecs system selects the corresponding nodes in the graph (i.e., the recently used features or commands). The system currently uses the last 5 distinct observed commands for this initial activation phase, however, this history size is a configurable parameter. Larger history sizes will mean recommendations tailored more to the user's overall usage than their current context. After the user's last x commands are selected, the QFRecs system determines possible task (query) nodes based on their connection weights. This step amounts to estimating which of the tasks in the QF-graph are most likely to be the user's current task. Using these estimations, the QFRecs system then selects other relevant commands for those candidate tasks.

This process is illustrated in Figure 3.5. In this example, the history is size 2 and the last two observed commands are: "Blur/Sharpen"and "feather". The QFRecs system first finds the set of tasks that are strongly associated those two features (see the left-hand nodes in Figure 3.5a) using the edge weights in Table 3.1. In the next step, the QFRecs system uses those strongly associated tasks to isolate other features (see the right-hand nodes in Figure 3.5b) associated with those tasks. These features are then ranked according to the summed weights of all of their associated tasks activated in the previous step, enabling the system to recommend the top k features. In this small example ($k = 2$) QFRecs recommends "blend"and "add layer mask"to the user. The pseudocode of the overall process is included in Appendix B.


Tasks
Features
how to blur image background in gimp
how to blend two images in gimp
Gimp how to transparent background
Gimp make image black and white
blur/sharpen
blend
feather
add layer mask
color balance

(a) Selection of possible tasks (in orange) based on current user context.


Tasks
Features
how to blur image background in gimp
how to blend two images in gimp
Gimp how to transparent background
Gimp make image black and white
blur/sharpen
blend
feather
add layer mask
color balance

(b) Selection of features to recommend (in green) based on tasks selected in Figure 3.5a.

Figure 3.5: A simple illustration of the QFRecs system.

3.5 Recommendation Types

Using the process described in section 3.4, the recommended commands are one of the following two types:

- "Familiar" Recommendations: Commands in the user's existing command vocabulary that are predicted to be most relevant to the current usage context.
- "Unfamiliar" Recommendations: Contextually-relevant commands not yet in the user's command vocabulary.

These recommendations serve different purposes. "Familiar" recommendations will not introduce users to new commands, but if promoted effectively, they have the potential to improve task efficiency. "Unfamiliar" recommendations, on the other hand, have the potential to enhance feature awareness. The proposed approach is capable of generating both types of recommendations raises a number of interesting interface presentation questions, which is explored in chapter 5. However, the QFRecs system is first explored for the accuracy and potential utility of these document-based recommendations using offline analysis in chapter 4.

3.6 Integrating into the Interface

To present the features recommended from the QFRecs system, I developed two prototypes of the QFRecs system each with a different presentation technique. Based on the related work described in section 2.3, I used the ephemeral menu adaptation technique [14] for presenting recommended menu items in these prototype interfaces.

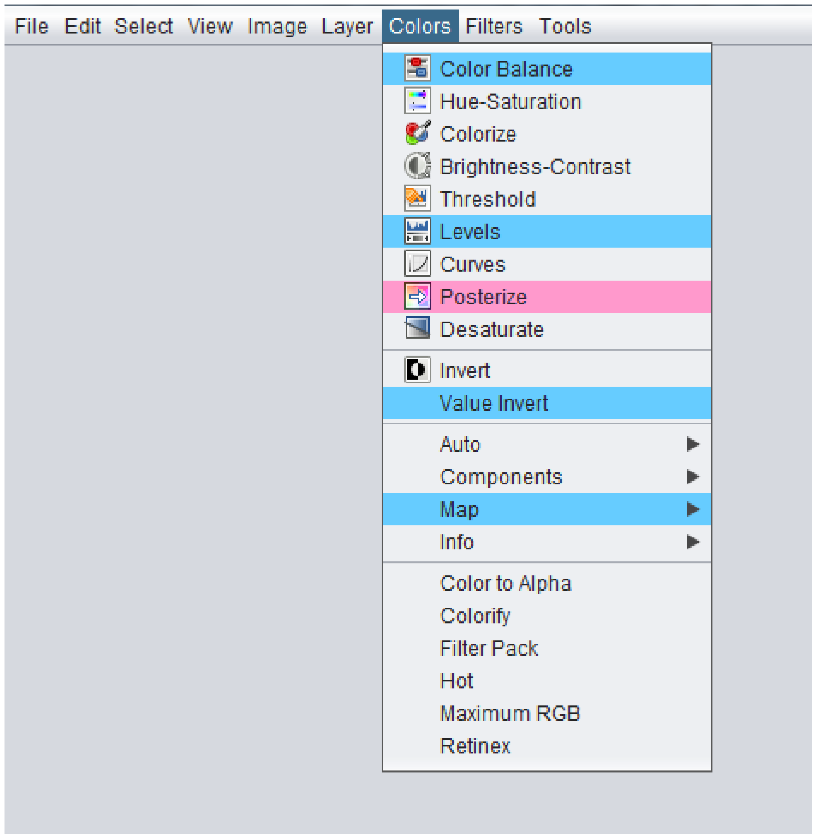


Figure 3.6: Combined interface prototype with the QFRecs system

In one prototype, named *Combined Interface* (Figure 3.6), I presented both types (familiar and unfamiliar features) of recommended features in the menus and used colored highlighting [18] to distinguish between the two types. In the second prototype, named *Separated Interface* (Figure 3.7), I presented contextually relevant and familiar features within the menus using the ephemeral highlighting technique but relevant and unfamiliar features are presented in a separate palette similar to the *CommunityCommands* [40]. In this prototype, I also used highlighting for the recommended features in the menus to distinguish them from the non-recommended features.

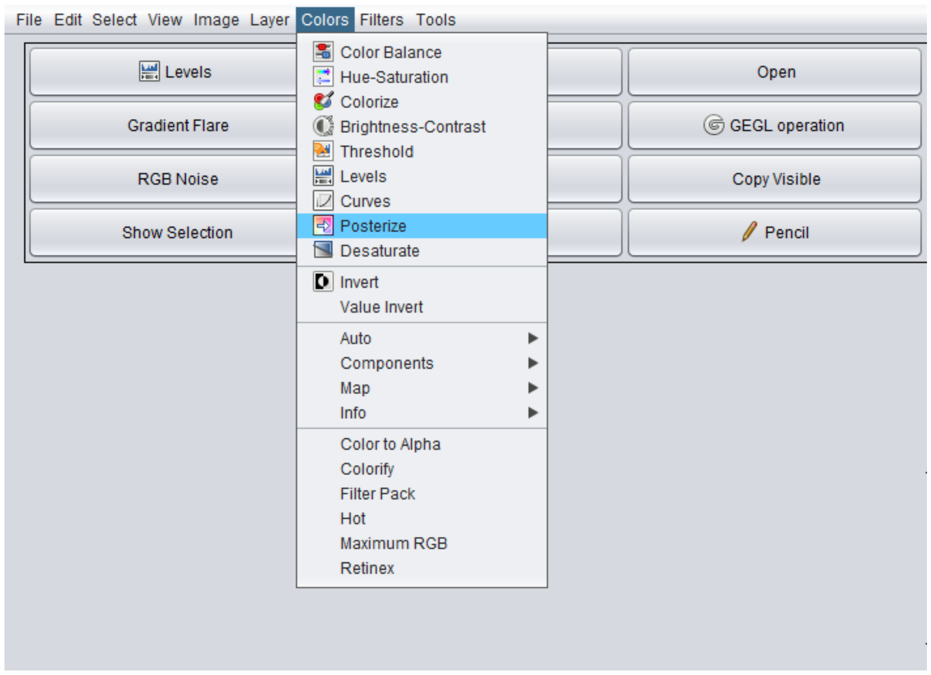


Figure 3.7: Separated interface prototype with the QFRecs system

3.7 Summary

In this chapter, I presented the QFRecs feature recommender system. First, I described the QF-graph. Then I discussed how the information required to build a QF-graph can be collected from online resources (e.g., tutorials, online documentation etc.). After that, I described the data cleaning process that I performed to reduce the noise in the collected raw QF-graph data. Then, I followed this by describing my system's recommendation generation process using the noise-reduced QF graph generated from the cleaned raw QF-graph data. After that I discussed the types recommendations that my system generates. Finally, I described how my system integrates generated recommendations into the application's interface.

Chapter 4

Offline Evaluation

In this chapter, I evaluate the QFRecs system's prediction accuracy on a corpus of GIMP usage data that was collected as part of the *Ingimp* project [33, 44]. This corpus contains feature usage histories (or logs) from 207 GIMP users, collected over a period of approximately two years. I evaluate the QFRecs system along two dimensions. The first is its ability to generate relevant recommendations, to gain an initial understanding of the approach's potential to improve users' feature awareness. Second, I evaluate the QFRecs system's accuracy in predicting a user's next feature or command selection and compare the QFRecs system's approach to frequency- and recency-based prediction algorithms (e.g., [11, 18]).

4.1 Potential to Promote Awareness of Relevant Commands

In the following subsections, I measure the QFRecs system's potential to generate recommendations that can improve users' feature awareness. To do so, I analyze the QFRecs system's performance on *InGimp* usage data from several aspects.

4.1.1 Mean Number of Familiar and Unfamiliar Features in the Recommended set

In assessing the QFRecs system's potential to make users aware of new commands, I examined how many of the generated recommendations could be classified as *unfamiliar*. As defined in section 3.5, *Unfamiliar* features are those that are predicted to be contextually-relevant but not yet observed in users' feature or command selection history. *Familiar* features are those that are predicted to be contextually-relevant and observed in users' feature or command selection history. Figure 4.1 illustrates the mean breakdown of the recommendations into the two types when the QFRecs system generates for 5, 10, 15 and 20 recommendations (recommendation size in Figure 4.1). Since the number of *unfamiliar* features is greater than the number of *familiar* features in all the cases, these results indicate that the QFRecs system tends to favour *unfamiliar* recommendations, particularly as the number of recommended features increases. For example, for the recommendation set size of 20 features, on average the QFRecs recommended about 13 *unfamiliar* features.

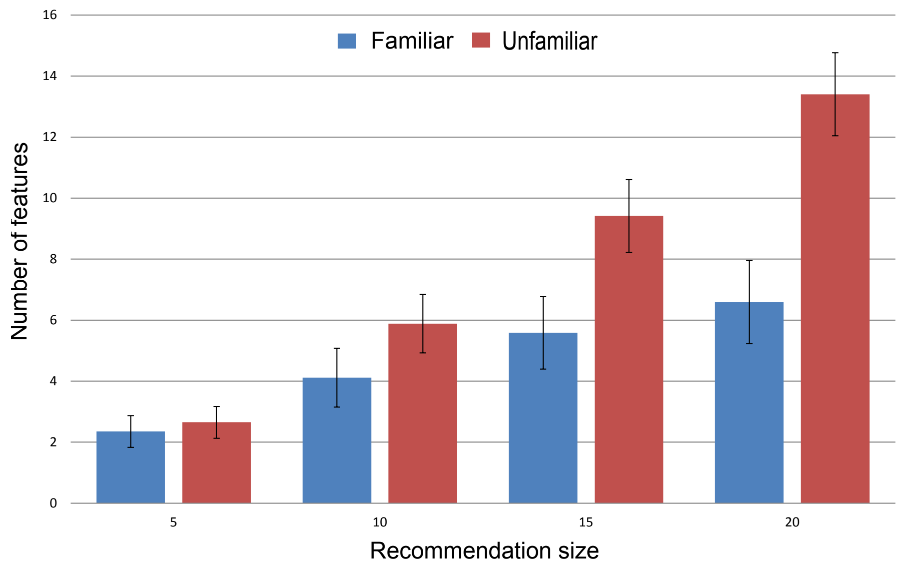


Figure 4.1: The means for the number of *familiar* features and *unfamiliar* features for different recommendation sizes (Error bars are in standard deviation).

4.1.2 Relevance Measure of the QFRecs System's Recommendations

As a measure of recommendation relevance, I used a modified version of the k-tail evaluation method, introduced by Li *et al.* [36] to evaluate their *Community-Commands* recommendation system [36, 40]. A k-tail evaluation divides a series of used features F into two sets: a training set (F_{train}) and a test set (F_{test}), such that the test set F_{test} contains k distinct features which are not in F_{train}. The training set is then used as the user's history to measure the prediction algorithm's performance based on how well it predicts those k distinct features in F_{test}. To focus on the relevance of the *unfamiliar* recommendations to the user's current usage context, the k-tail evaluation method was adapted as follows: I measured whether or not

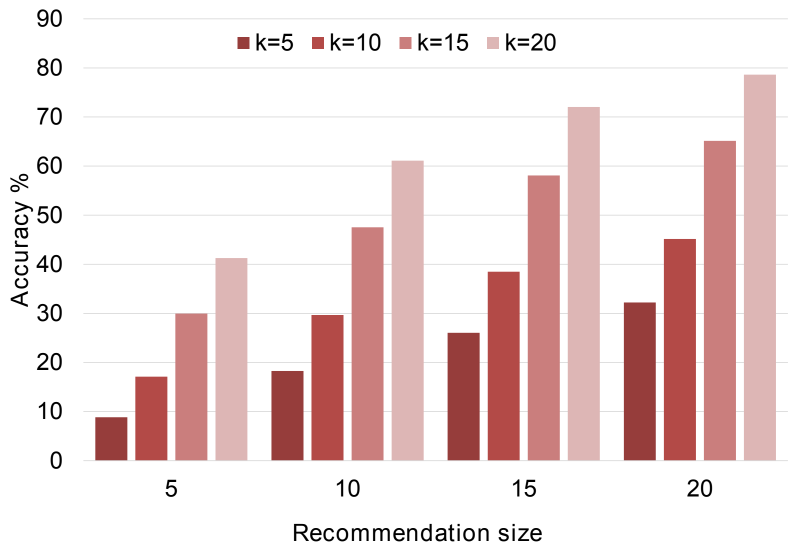


Figure 4.2: Performance of the QFRecs system recommending an *unfamiliar* feature that is then used in the next k feature invocations.

the QFRecs system's recommendations predict at least one new feature in the next k feature invocations (i.e., whether or not at least one *unfamiliar* recommendation appears in the next k feature invocations).

Figure 4.2 depicts the results for a range of recommendation and tail sizes. With a recommendation size of 20, the QFRecs system achieves a k-tail accuracy that is up to 80%. In some respects, this figure represents a lower bound on the relevance of the recommendations; users may be failing to use certain features not due to the lack of relevance, but because of lack of awareness. The k-tail accuracy of the QFRecs system for *familiar* recommendations is also evaluated, with Figure 4.3 illustrating similar trends that the QFRecs system achieves about 84% k-tail accuracy.

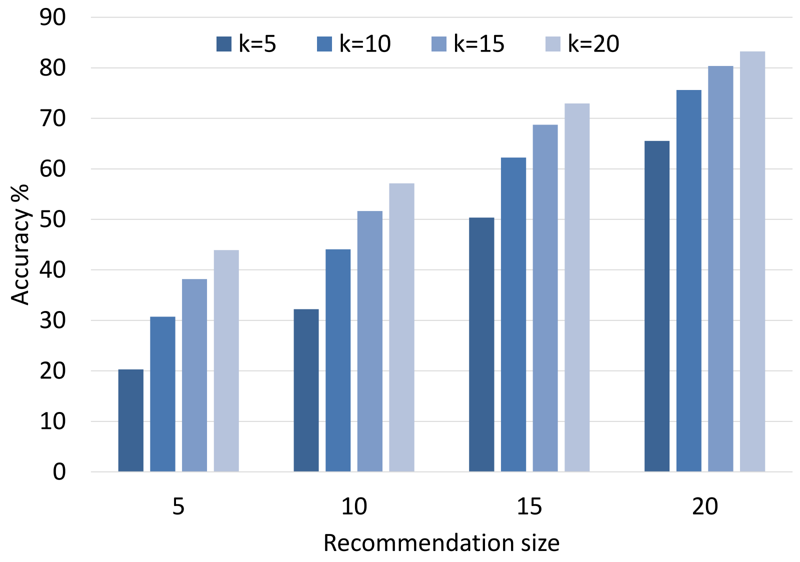


Figure 4.3: Performance of the QFRecs system recommending a *familiar* feature that is then used again in the next k feature invocations.

4.1.3 Percentage of "Accurate"Recommendations

To provide further insight into the relevance of the novel command recommendations, I examined how many of the *unfamiliar* recommendations are accurate according to the modified k-tail evaluation method. Table 4.1 illustrates that, on average, 3% - 9% of "unfamiliar"recommendations in a given set appear in the user's next k commands. The table also illustrates the large variability in accuracy, with up to 80% of the recommendations appear in the user's next k selections.

Figure 4.4 examines the "accuracy"of the unfamiliar recommendations for recommendation size 20 and tail size 20 in further details. In particular, I grouped all the generated recommendations based on how many *unfamiliar* recommendations appear in a user's next k feature invocations. Figure 4.4 shows that about 52% of the generated feature recommendations for recommendation size of 20 contain 2 or

	$K = 5$			$K = 10$			$K = 15$			$K = 20$		
	Accuracy %	Max %	SD	Accuracy %	Max %	SD	Accuracy %	Max %	SD	Accuracy %	Max %	SD
$R = 5$	3.3	60	10.79	5.79	80	11.95	6.68	80	12.75	7.25	80	13.76
$R = 10$	3.85	40	6.11	5.81	50	7.54	7.3	50	8.46	8.53	50	9.13
$R = 15$	4.39	33.33	5.05	6.39	40	6.29	7.68	40	7.07	8.7	40	7.64
$R = 20$	4.52	25	4.07	6.38	30	5.1	7.84	35	5.76	9	35	6.24

Table 4.1: Average percentage of correct recommendations for each tail (i.e., k) and recommendation size (i.e., R)

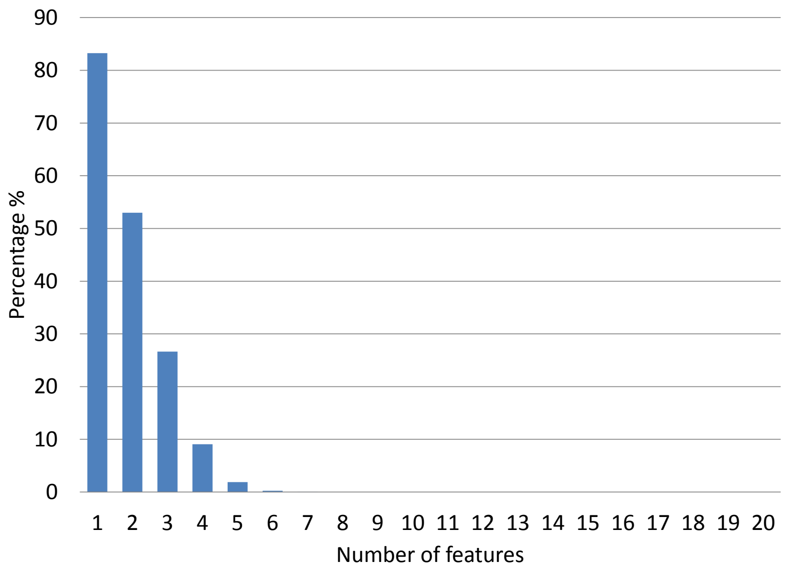


Figure 4.4: Percentage of recommendations by the number of useful features (recommendation size 20 and tail size 20)

more *unfamiliar* features ($>$ 1.8 expected number of unknown features) which were present in users' next 20 feature invocations.

4.1.4 Measure of Unknown Utility Features

In the recommendations generated by the QFRecs system, there exists a subset of features which are unfamiliar to users but do not appear in users' next k feature

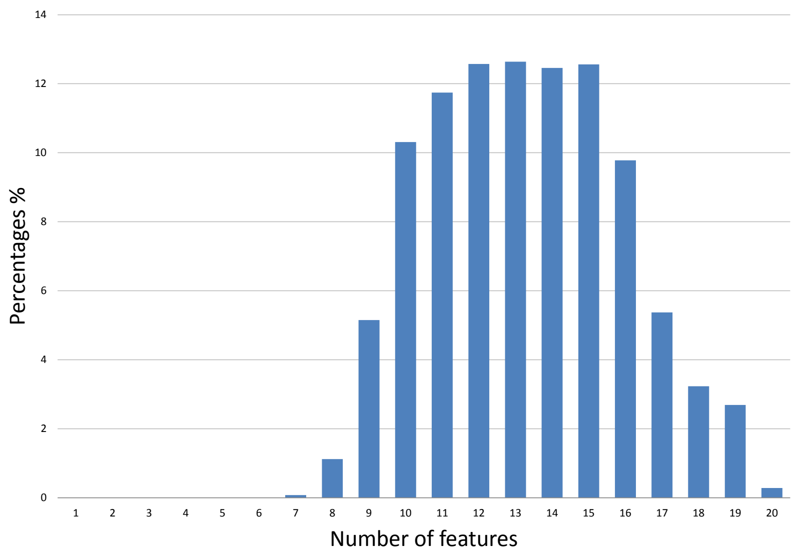


Figure 4.5: Recommendations grouped by number of *unknown-utility* features (recommendation size 20 and tail size 20)

invocation. Although these features are novel and unfamiliar to users, the performance measures described in figure 4.2 consider them as wrong predictions. These features are missing in users' next k feature selections, therefore the utility of these features are unknown in the evaluation on inGimp data. There may be concern for their usefulness, but these features have the potential to enhance users' feature awareness. I refer to these features as *unknown-utility* features. I measured the percentage of predictions grouped by the number of *unknown-utility* features. Figure 4.5 shows the results for recommendation size of 20 and tail size of 20. Figure 4.5 shows that about 13% of recommendations contain 15 *unknown-utility* features. Figure 4.5 also indicates that every set of recommended features contains at least 8 *unknown-utility* features.

4.1.5 Summary

Combined, for recommendation size 20 and tail size 20, these results suggest that the QF-based approach (as currently instantiated in the QFRecs system) is able to recommend at least one relevant and unfamiliar feature in about 80% of the recommended sets. Further, about 83% of the recommendations contain at least one known relevant feature which helps user to recall features. Results also indicate that each recommended set of 20 features contains about 8 to 19 *unknown-utility* features, which may be useful to users. Given the above results, if the recommended features are promoted effectively within the interface, the recommendations could potentially aid users in completing their current task and enhance their feature-awareness. The question of presentation is explored further in my laboratory study chapter 5.

4.2 Predicting the Next Feature

The results described in section 4.1 provide an initial indication that the QFRecs system does generate some user-relevant recommendations in that they appear later in the user's command stream. In this section, I explore the approach's potential to immediately streamline access to needed commands, by analyzing the degree to which the recommended set accurately predicts the next command in the stream. I also compared the QFRecs system with two algorithms commonly used in prior work on adaptive interfaces (e.g., [11, 18]): frequency-based predictions, and recency-based predictions. In this section, I define an *accurate* prediction as one where the user's next action is within the recommended set of commands.

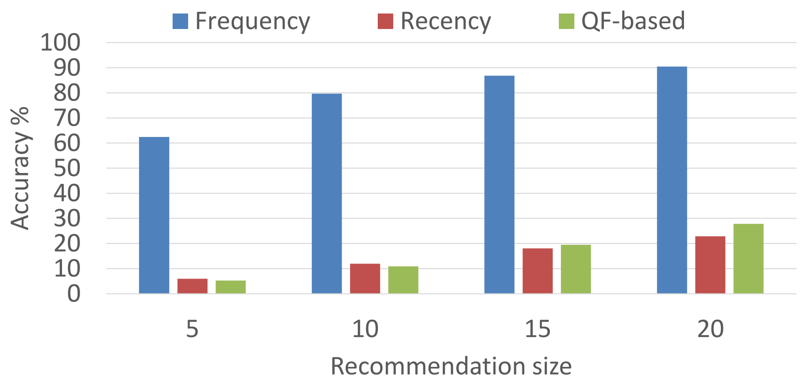


Figure 4.6: Accuracy of the Frequency-based, the Recency-based and QF-based approach for an average diverse session.

In case of frequency-based approach, I only considered the user's usage data (not all user's usage data) to calculate his feature usage frequency. I found that the frequency-based algorithm dramatically outperformed the others when it came to predicting the user's next command (see Figure 4.6). When examining the reasons why, I found that the users in this particular dataset tended to have very homogenous and stable command usage, which naturally favours the frequency-based approach. As an example, consider a feature invocation sequence of length 149, but that consists of only 10 distinct features. With a recommendation size of 10, once these 10 features are observed, the algorithm will never fail. To better characterize and explore the dataset, I define a user's feature usage diversity, R_d, as follows

$$R_d = \frac{NumberofDistinctFeaturesUsed}{SequenceLength} \tag{4.1}$$

In the *InGimp* dataset, the mean R_d for all 178 users with usage sequences longer than 20 is 0.1668 (standard deviation of 0.1329). The max(R_d) was 0.6666 (sequence

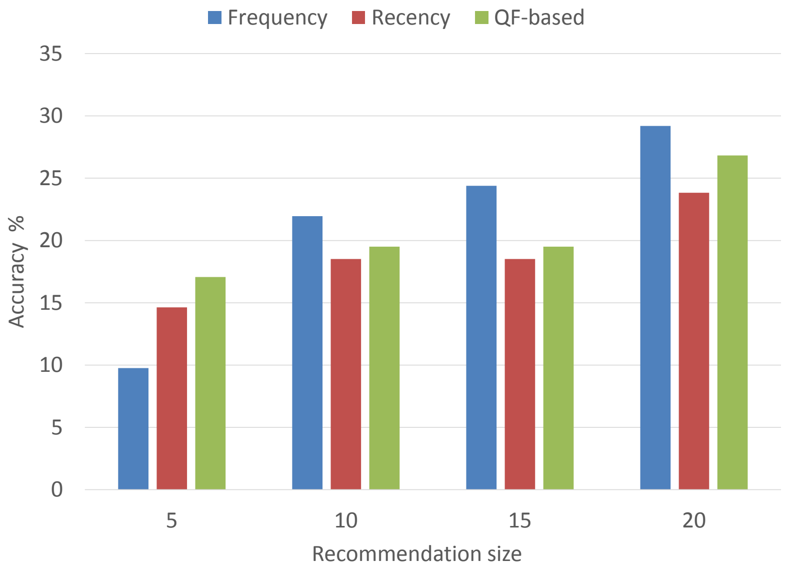


Figure 4.7: Accuracy of the Frequency-based, the Recency-based and QF-based (QFRecs) approach for an average diverse session.

length = 42 and number of distinct commands = 28) and min(R_d) was 0.0043 (sequence length = 23459 and number of distinct commands = 101). Whereas the frequency-based approach substantially outperforms the QFRecs system for users with low feature diversity, the results are much more promising for users with high feature diversity. As an example, Figure 4.7 compares the accuracy of the different algorithms for a user with near mean feature diversity (0.1667). In this case, the QFRecs system actually outperforms the alternatives when the recommendation size is 5 or less.

A second potential downside of the frequency-based approach is that it requires usage patterns to stabilize before it can be effective. For example, Figure 4.8 compares the frequency-based approach's accuracy over its first 30 feature invocations to its overall performance. The QFRecs system, on the other hand, requires less start-

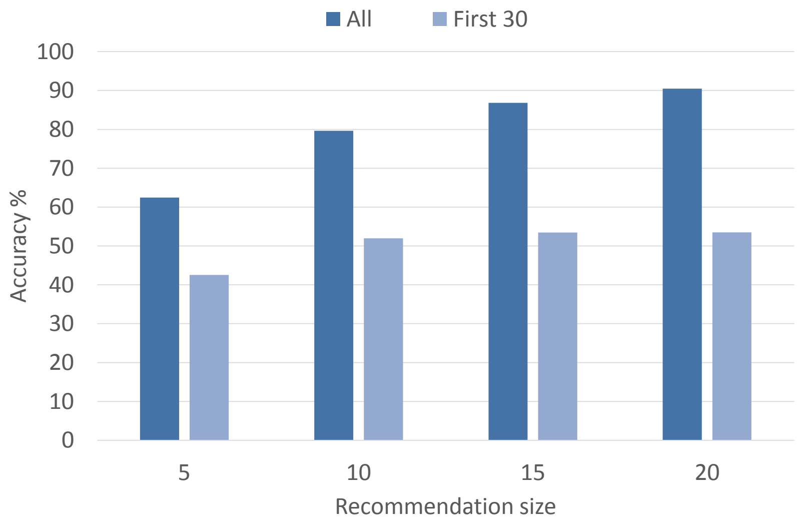


Figure 4.8: The accuracy of frequency-based approach over users first 30 command invocations and over all command invocations.

up time, as it is currently set to generate recommendations based on the last five commands.

4.3 Discussion

The offline evaluation indicates that for my target application (GIMP), the QF-based approach was able to generate at least some contextually relevant recommendations. For example, when providing the user with 20 recommendations (which would be distributed throughout the entire menu hierarchy), previously collected GIMP usage data indicated that at least one recommended command would subsequently be used within a user's next 20 selections. Further work is required, however, to assess the relevance and utility of recommended commands that do not later appear in the user's command stream, since their omission does not imply a lack of utility.

A potential first step in this direction would be to collect relevance ratings from application experts; however, a longer-term experiment is necessary to fully assess the value of these recommendations from the user's perspective. A longer-term experiment would also enable us to compare the QF-based approach to the collaborative filtering approaches explored in prior work [36, 40].

4.4 Summary

In this chapter, I discussed the evaluation of QFRecs system on a corpus of GIMP usage data that was collected as part of *Ingimp* project [33, 44]. I found that frequency-based or any history-based approach is sensitive to the usage diversity and session length. While the QFRecs system may not be an effective predictor given long, homogenous sessions, it has advantages for short, diverse user sessions. I also illustrated that the QFRecs system can recommend useful and novel features (unknown previously) which are relevant to a user's context (used in next *k* selections) whereas a frequency- or recency-based approach is (by definition) unable to recommend *unfamiliar* commands. Given the potential for the QFRecs system to produce recommendations that are both needed and novel, in Chapter 5, I explore how the system might present these recommendations to the user.

Chapter 5

Laboratory Study

As described in the section 3.5, the QFRecs system's recommendations can be divided into two types: features or commands that are *familiar* to the user (i.e., they are part of the user's usage history) and those that are *unfamiliar* (i.e., they are not part of the usage history). Since the goal of my laboratory evaluation was to explore two different ways to present the system's recommendations, I conducted a formal laboratory study with 18 participants comparing two prototype user interfaces for presenting the recommendations of the QFRecs system to the control interface (with no adaptive behavior). In the study, I also explored the usefulness of the QFRecs system on users' primary task performance and incidental awareness. This laboratory study was approved by University of Manitoba's Research Ethics Board. The certificate of approval is included in Appendix A.

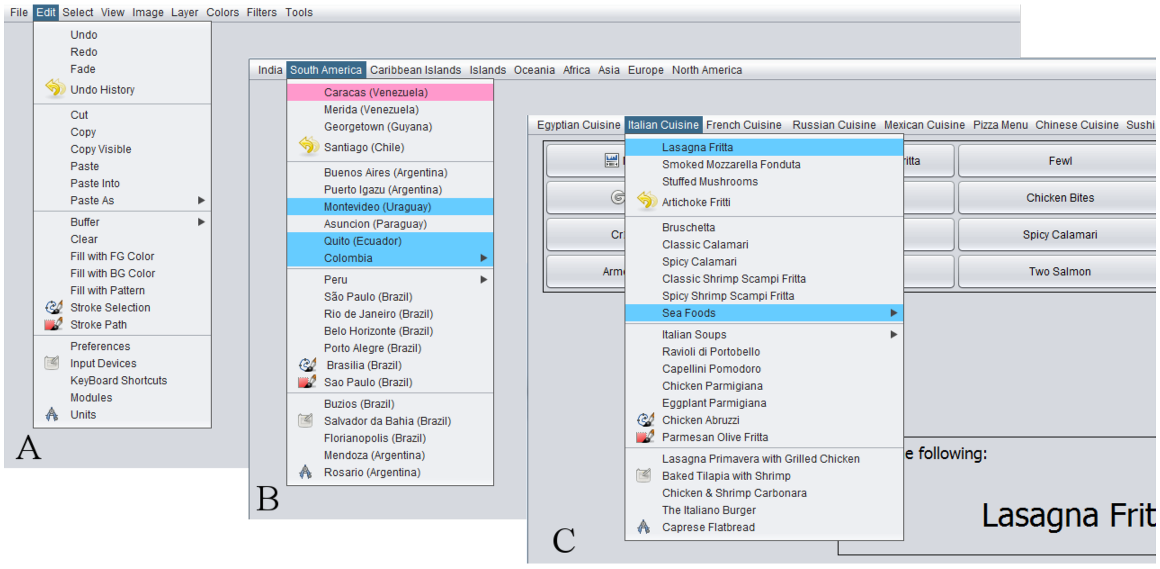


Figure 5.1: Sample snapshot of the interface variants used in study: Basic interface (A), Combined interface (B), and Separated interface (C). The features highlighted in blue were “familiar”and the features highlighted in pink were “unfamiliar”features.

5.1 Participants

Eighteen participants (two females) were recruited from a university campus. Participants were between the ages of 18-25 and were provided with a $15 gift card.

5.2 Apparatus

An Intel Core i7 desktop with 8 GB of RAM and Microsoft Windows 7 was used for the experiment. The system was connected to a 22”LCD monitor with a 1920x1080 resolution. The experiment software recorded all timing and selection data.

5.3 Conditions

Our three interface variants were as follows:

1. **Basic Interface**: A control condition with traditional static menus (Figure 5.1 A).

2. **Combined Interface**: Recommendations for both "familiar" and "unfamiliar" features were highlighted in place (i.e., within the menus) using one of the best known visual highlighting techniques: ephemeral adaptation [14]. With ephemeral adaptation, recommended items appear immediately when a menu is opened, with the remaining items gradually appearing after an initial delay (500 ms as in [14]). The two types of recommendations ("familiar" vs. "unfamiliar") were distinguished only by colour (Figure 5.1 B). In figure 5.1 B, the features highlighted in blue were "unfamiliar" and the features highlighted in pink were "familiar" features.

3. **Separated Interface**: "Familiar" recommendations were ephemerally highlighted within the menus but "unfamiliar" recommendations were presented in a separate palette (as well as appearing as non-recommended features in the menus). In figure 5.1 C, the features highlighted in blue were "familiar" features. The full menu path of a feature was also available on mouse hover over the feature in the palette. In comparison to the Combined Interface, with the Separated Interface, users could choose to ignore these "unfamiliar" recommendations completely in favour of focusing on their primary task. This palette-based approach has been commonly explored in prior work on novel command recom-

mendations (e.g., [36, 37]).

To enable both the combined and separated user interface to make recommendations using real GIMP usage data (see the subsection 5.4), and QF-graph to be built from actual web queries and resulting documentation, the menu hierarchy in all interfaces was modeled after The GNU Image Manipulation Program (GIMP) version 2.8.6. To simplify the interface slightly for the participants, I excluded the "Windows"and "Help"menus, resulting in 9 top-level menus containing a total of 368 features.

5.4 Design, Tasks, and Procedure

The experimental task was a sequence of menu selections using each of the three interfaces described in section 5.3. In other words, the study used a within-subjects design, where all participants experienced all three interface variants. The menu selections were based on a real user's data from the Ingimp dataset described in the section 4.1. I selected data from a user with a sufficiently long sequence that was also close to the data set's mean diversity. The selected usage sequence was 74 selections long and had a diversity of 0.2065 (defined in the section 4.1). From this sequence, I used the first 20 selections as "training", and the next 50 features as the main task (discarding last 4 selections in the interest of participant time). The reason behind asking participants to select a sequence of feature selections rather than performing a real task is time. In a real task, participants may take longer time based in the complexity of the task and may also devote a large portion of the time to figure out a certain step of the task rather than selecting features.

For each feature selection, the experimental interface provided participants with the name of the feature, but not the menu name. As a result, participants had to explore the interface (using the categorization as a guide) to find their needed commands. Once the participant correctly selected the displayed feature, the next feature to be selected was displayed. I used the same selection sequence in all interface variants, but used different interface *masks* (the GIMP menus, Geography-related menus, and Cuisine-related menus) to mitigate learning effects between conditions. The structure of all three masks was identical. The order of interface and the assignment of masks to the interface were counterbalanced using a Latin square.

After each main task, I measured incidental command awareness [12], by having participants perform a recall test. During this recall test, participants selected 24 distinct commands that were recommended by the QFRecs system but that were not part of the main task.

The procedure for the 1.5 hour experiment was as follows: participants first completed a background and demographics questionnaire. Then, for each interface variant, participants completed a training task consisting of 20 selections, followed by the main task consisting of 50 selections. After the main task, participants completed the NASA-TLX [25], which measures perceived workload. Participants then completed the recall test described above prior to repeating the above steps with the next interface variant. The session concluded with a comparative questionnaire.

In the Combined and Separated interfaces, the QFRecs system presented its top 20 recommendations based on the user's last 5 command selections. With this particular usage stream, this recommended set accurately predicted the next command in the stream 18% of the time.

5.5 Hypotheses

Given the low predictive accuracy of the QFRecs system's recommended set in comparison to those studied in prior work (e.g., [11]), I did not have any apriori hypotheses on the effect of the interface on selection speed during the primary task. I did, however, have the following hypotheses with respect to recall selection speed, perceived workload and user preference:

- H1 (*Recall Speed*): The Combined interface will have faster recall times than both the Basic interface and the Separated interface. I expect no difference between the Basic interface and the Separated interface.
- H2 (*Perceived Workload*): Perceived workload will be lower with the Combined interface than with the Separated interface.
- H3 (*User Preference*): Users will prefer the Combined interface over the Separated interface.

5.6 Results

The result was analyzed with a one-way RM-ANOVA with Interface (Basic, Combined, Separated) as the within-subjects factor. $p < 0.05$ was used as the threshold for significance, and Bonferroni corrections were applied to all post-hoc comparisons. Error bars in figure 5.2 represent Standard Error.

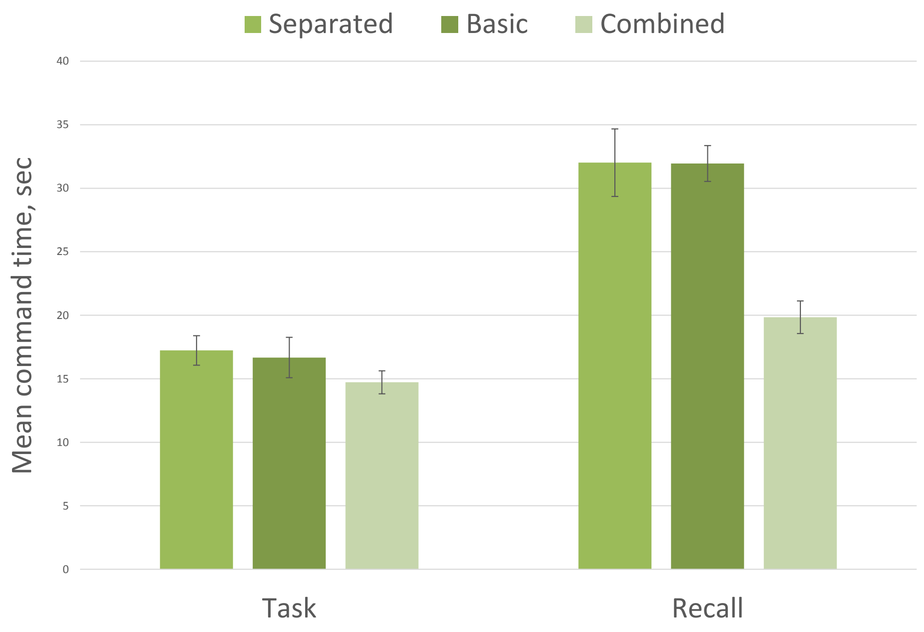


Figure 5.2: Mean selection time between commands (with standard errors) for the main task (left) and the recall task (right)

5.6.1 Primary Task Selection Time

As expected, there is no significant main effect of interface on primary task selection time ($F_{2,34} = 1.064$, $p = 0.356$, $\eta^2 = 0.059$, Figure 5.2 left). The fact that the recommendations did not significantly improve immediate task performance is consistent with prior results on low accuracy predictors (e.g., [14]). Despite having only limited immediate accuracy, the recommendations did not hurt task performance, and perhaps even helped it slightly (as indicated by the means and effect size), when presented in-place (in combined interface).

5.6.2 Recall Speed (H1)

In the case of the incidental awareness task (i.e., the Recall test), the main effect of Interface was statistically significant ($F_{2,34} = 12.731$, $p < 0.001$, $\eta^2 = 0.428$, Figure 5.2 right). Moreover, the post-hoc comparisons revealed significantly faster selections using the Combined interface (19.8s, se 1.3s) when compared to either the Basic (32.0s, se. 1.4s, $p < 0.001$) or Separated (32.0s, se. 2.7s, $p = 0.005$) conditions. The difference between the Basic and Separated conditions was not statistically significant ($p = 1.00$). Therefore, the result provides evidence to support H1.

5.6.3 Perceived workload (H2)

As a measure of a perceived workload, the data from NASA-TLX questionnaires were used. Figure 5.3 shows significant main effects of Interface on two of the NASA-TLX categories: *hard work* and *frustration*. For these categories, participants reported experiencing lower workload with the Combined interface, however, the only significant pairwise difference revealed by the post-hoc comparisons was that of frustration for the Combined and Basic interface variants ($p = 0.038$). Therefore, the result could not fully support the H2.

5.6.4 User Preference (H3)

Regarding subjective preferences, in the post study questionnaire participants were asked to rank the different interface types based on their overall performance. The analysis of results showed high inclination towards the Combined option, with 13 interviewees ranking it as their most preferred interface variation. For comparison,

	Basic	**Separated**	**Combined**	**F**	**Sig**
Mental demand	13.11 (3.32)	12.78 (4.07)	11.28 (3.98)	1.756	0.188
Physical demand	4.5 (4.25)	4.89 (4.00)	3.94 (3.44)	1.125	0.336
Temporal demand	10.78 (5.51)	10.78 (4.57)	9.72 (4.56)	0.98	0.386
Success	6.22 (4.84)	4.56 (4.03)	4.78 (4.33)	1.717	0.195
Hard work	**14.28 (4.32)**	**12.94 (4.36)**	**10.39 (5.24)**	**3.754**	**0.034**
Frustration	**10.89 (3.76)**	**9.39 (4.6)**	**7.11 (4.24)**	**5.251**	**0.01**

Figure 5.3: Mean (st. err.) NASA-TLX values (1=low, 20=high). Rows in bold indicate significant differences.

3 users rated the Separated interface as their primary choice and only two preferred the Basic one (this difference was significant with $\chi^2 = 16.0$, $p = 0.002$). Therefore, the result supports the H3.

5.7 Discussion

This laboratory evaluation is one of a few systematic explorations of how to present command recommendations designed to promote command awareness (as opposed to short-term efficiency). The results indicate that presenting these types of recommendations in-place can significantly improve incidental command awareness over a palette-based approach. This is perhaps not surprising given that this presentation technique is more obtrusive. What is perhaps more surprising is that the extra visual complexity introduced into the main interface did not appear to negatively impact short-term task efficiency. Users also preferred this in-place presentation strategy and reported lower levels of frustration. Further exploration is needed to determine the sensitivity of these results to factors such as the number of recommendations and their distribution across the menus.

5.8 Summary

In this chapter, I presented the study that I conducted to evaluate the QFRecs system against the control interface, which involved measuring the effect of feature recommendations on participants' task performance, recall speed, perceived workload and user preference. According to the results, even the low prediction accuracy of QFRecs does not hurt users' primary task performance. I found the combined interface of the QFRecs system significantly improves users' feature awareness over the control interface. In the user study, participants generally liked the combined interface with the QFRecs system more than the other interface prototypes.

Chapter 6

Conclusion

Feature-rich software applications (e.g., image-manipulation programs, word processor programs) contain hundreds (or even thousands) of commands or features. Moreover, the number of features are increasing with each new version. Prior research showed that most users of feature-rich software use a small fraction (less than 10%) of available features [29, 36] and are unfamiliar with most of the functionalities. This thesis proposes a recommendation system, the QFRecs system, which leverages a new form of information on command relevance: command-to-task mappings mined from Web documentation, to suggest relevant and unfamiliar commands to users. The offline evaluation of the QFRecs system suggests that this technique has the potential to expose users to a number of new and relevant commands, while the laboratory evaluation suggests value in integrating the recommendations within the main interface.

6.1 Contributions

The first contribution of my thesis is a novel approach to command recommendation in feature-rich software that uses web documentation (e.g., Web tutorials) as a knowledge source. Prior approaches of feature recommendations tend to collect command usage data from the user community to build the knowledge base [37, 36]. In contrast, this approach intends to eliminate the need of collecting command usage data by creating knowledge base from online documentations. I have also demonstrated the potential strengths and weaknesses of this approach given a variety of command usage patterns.

The second contribution of my thesis is an empirical exploration of how such a system should promote its recommendations within the interface. I have designed and implemented two prototypes (combined interface and separated interface) embedded with the QFRecs system by mirroring the GNU image manipulation program (GIMP). Further, I have evaluated the prototypes with the control interface in a formal laboratory study with 18 participants. The results of the user study indicate that combined interface has statistically significant effect on users' incidental feature awareness than control interface and separated interface. However, the results do not show any significant main effect on users' primary task performance between interface prototypes.

6.2 Limitations and Future work

Combined, the results of the offline and laboratory evaluations suggest that recommendations generated based on information mined from web documentation is a promising approach to improve command awareness in feature-rich software. They also highlight a number of important considerations moving forward.

6.2.1 Prototype Extensions and Improvements

Motivated by this initial feasibility study, there are a number of system-related improvements worth exploring. Aside from culling queries from the original QF-Graph that were clearly not representative of high-level tasks, I did little to optimize the graph's suitability to act as a recommender. More sophisticated lexical analysis or machine learning could enable the system to focus its recommendations on a more informative set of high-level tasks. It also possible that restricting the documentation set to specific tutorial repositories would improve the precision of the command-to-task mappings. Finally, there are numerous avenues that could be explored to improve the approach's predictive capabilities, such as incorporating more frequency information into the recommendations.

6.2.2 Presentation Techniques

While the results of the user study show initial promise for an in-place presentation technique, there are a number of open questions concerning how to present recommendations in a way that will eventually lead to their adoption. For example, with the palette approach, it would be easier to provide rich supplemental information on why the command is recommended and how it might be used in practice. In a palette, the system could display its confidence in each recommended command, the list of tasks to which the command relates, and links to documentation that illustrate how to use the command. Such information could also potentially be integrated within the main interface (available, for example, on mouse over), but at the risk of impacting immediate task performance. Understanding these types of presentation-level tradeoffs will be important to the ultimate success of all approaches to command recommendation, not just ones based on Web documentation. It is also possible that a more static presentation technique is desirable. For example, the system could recommend entire task-centric interface that corresponds to the user's most probable high-level tasks [30, 31].

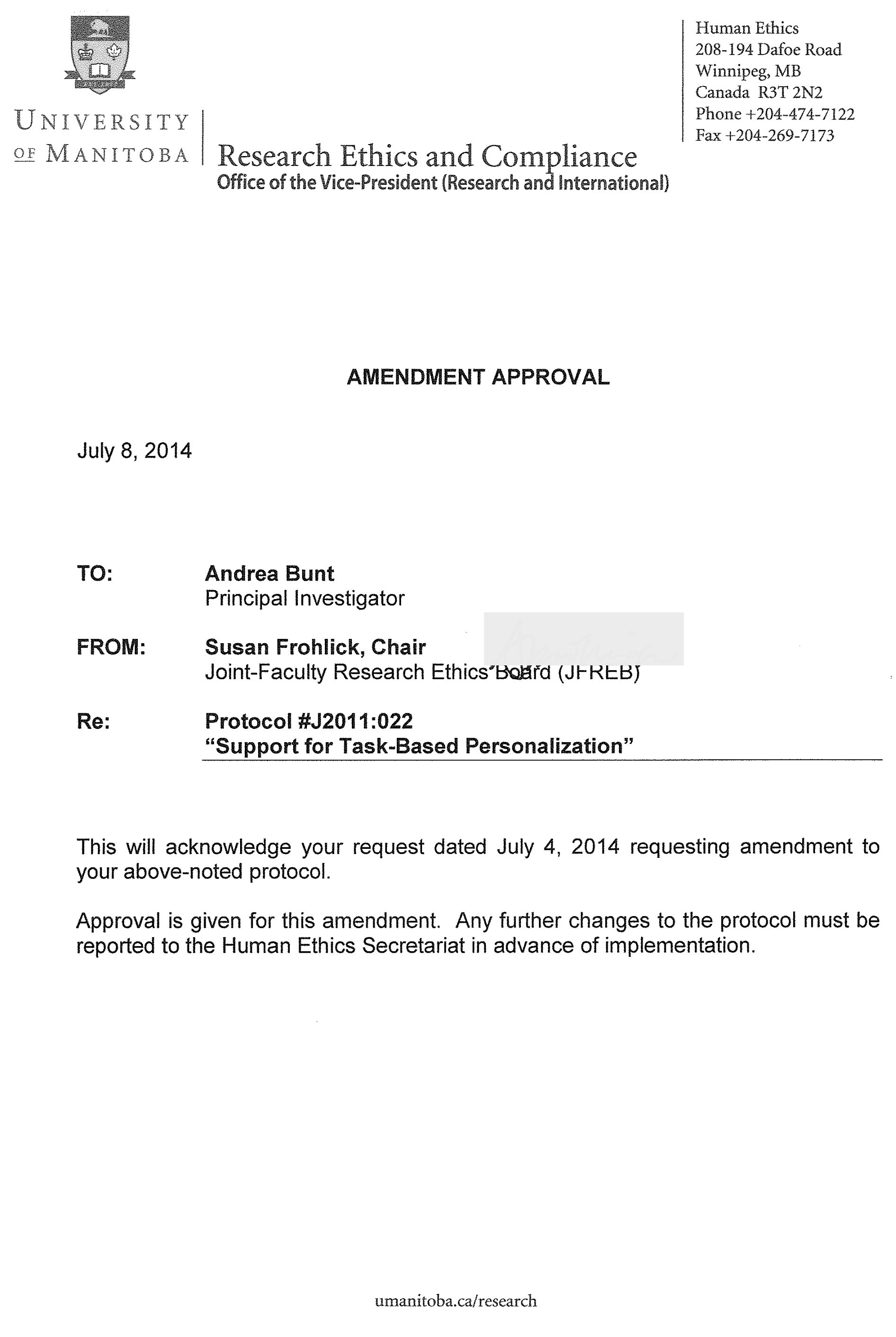
There is also the potential to make the interaction between the system and the user more of a mixed-initiative one [26]. In particular, the system could leverage the fact that the high-level tasks are in a human-readable form and display its task assessments to the user. The user could then refine these assessments to obtain more refined recommendations.

6.2.3 Generalizability to Other Applications

Finally, it would be interesting to explore the generalizability of the QF-based approach to command recommendations to feature-rich applications other than GIMP. Fourney et al.'s original QF-Graph results suggest that the technique will extend to other applications with a large Web presence [17]. Exploring generalizability to other applications, however, could provide insight on how properties of the graphs themselves affect their abilities to generate useful recommendations, such as the range of high-level tasks present, and the connectedness of the graphs.

Appendix A

Ethics Approval Certificate



UNIVERSITY OF MANITOBA | Research Ethics and Compliance
Office of the Vice-President (Research and International)

Human Ethics
208-194 Dafoe Road
Winnipeg, MB
Canada R3T 2N2
Phone +204-474-7122
Fax +204-269-7173

AMENDMENT APPROVAL

July 8, 2014

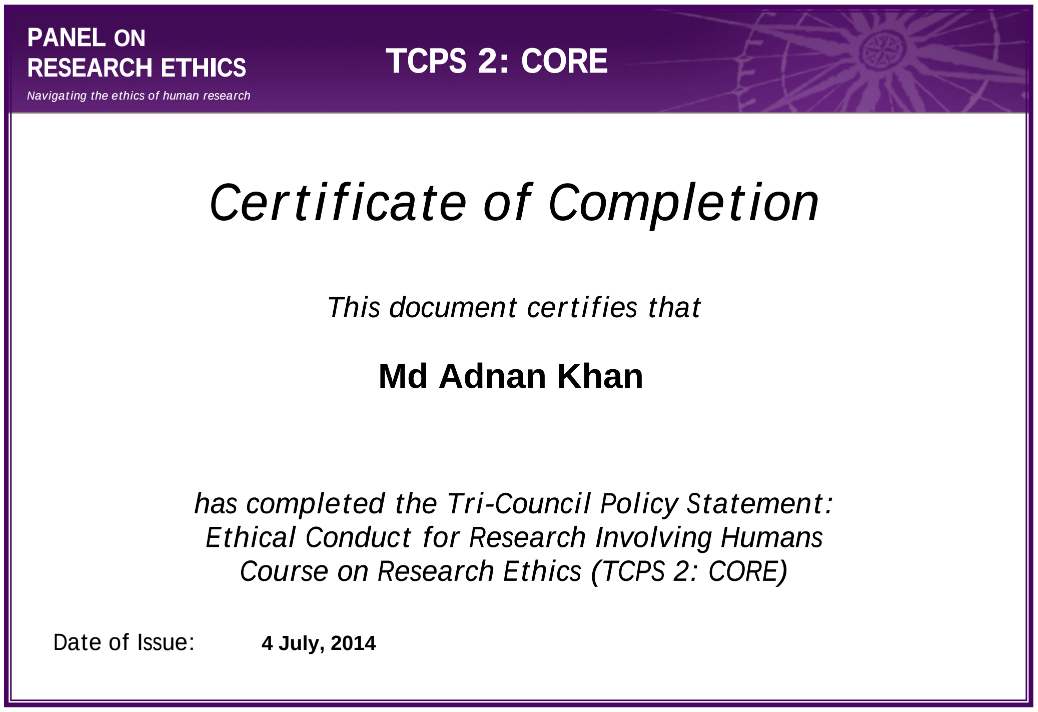
TO:	**Andrea Bunt** Principal Investigator
FROM:	**Susan Frohlick, Chair** Joint-Faculty Research Ethics Board (JFREB)
Re:	**Protocol #J2011:022** **"Support for Task-Based Personalization"**

This will acknowledge your request dated July 4, 2014 requesting amendment to your above-noted protocol.

Approval is given for this amendment. Any further changes to the protocol must be reported to the Human Ethics Secretariat in advance of implementation.

umanitoba.ca/research

Figure A.1: Ethics Approval Certificates



PANEL ON
RESEARCH ETHICS
Navigating the ethics of human research

TCPS 2: CORE

Certificate of Completion

This document certifies that

Md Adnan Khan

has completed the Tri-Council Policy Statement:
Ethical Conduct for Research Involving Humans
Course on Research Ethics (TCPS 2: CORE)

Date of Issue: **4 July, 2014**

Figure A.2: TCPS 2: CORE Completion Certificates

Appendix B

Pseudocode of the QFRecs feature recommendation

Perform step 1 - 6, whenever a user selects a menu item (e.g., feature):

1. Select x most-recently used features from the user's history.

2. Initialize the weight of the x most-recently used features in the QF-graph using following:

$$F_i = \begin{cases} 1 & \text{if } F_i \text{ is in } n \text{ most-recently used features.} \\ 0 & \text{otherwise.} \end{cases} \tag{B.1}$$

3. Calculate relevance of the queries using the edge weight between features to queries. The relevance weight of each queries Q_j is calculated using following:

$$Q_j = \sum_{i=1}^{n} E_{ij} * F_i \tag{B.2}$$

where n = total number of features in the QF-graph.

4. Update the weight of each feature F_i, using following:

$$F_i = \sum_{j=1}^{m} E_{ji} * Q_j \tag{B.3}$$

5. Rank all features based on the updated weights and recommend top r number of features to users.

6. Initialize all features weight to 0

Bibliography

[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, pages 167–176, New York, NY, USA, 2014. ACM.

[2] Nikola Banovic, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. Triggering triggers and burying barriers to customizing software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2717–2726, New York, NY, USA, 2012. ACM.

[3] Andrea Bunt, Cristina Conati, and Joanna McGrenere. Supporting interface customization using a mixed-initiative approach. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, IUI '07, pages 92–101, New York, NY, USA, 2007. ACM.

[4] Andrea Bunt, Patrick Dubois, Ben Lafreniere, Michael A. Terry, and David T. Cormack. Taggedcomments: Promoting and integrating user comments in online application tutorials. In *Proceedings of the SIGCHI Conference on Human*

Factors in Computing Systems, CHI '14, pages 4037–4046, New York, NY, USA, 2014. ACM.

[5] Pei-Yu Chi, Sally Ahn, Amanda Ren, Mira Dontcheva, Wilmot Li, and Björn Hartmann. Mixt: Automatic generation of step-by-step mixed media tutorials. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 93–102, New York, NY, USA, 2012. ACM.

[6] 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, pages 358–365, New York, NY, USA, 2001. ACM.

[7] Andy Cockburn, Carl Gutwin, and Saul Greenberg. A predictive model of menu performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 627–636, New York, NY, USA, 2007. ACM.

[8] Tao Dong, Mira Dontcheva, Diana Joseph, Karrie Karahalios, Mark Newman, and Mark Ackerman. Discovery-based games for learning software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2083–2086, New York, NY, USA, 2012. ACM.

[9] 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, pages 195–204, New York, NY, USA, 2011. ACM.

[10] Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. Sketch-sketch revolution: An engaging tutorial system for guided sketching and application learning. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 373–382, New York, NY, USA, 2011. ACM.

[11] Leah Findlater and Joanna McGrenere. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 89–96, New York, NY, USA, 2004. ACM.

[12] Leah Findlater and Joanna McGrenere. Evaluating reduced-functionality interfaces according to feature findability and awareness. In *Proceedings of the 11th IFIP TC 13 International Conference on Human-computer Interaction*, INTERACT'07, pages 592–605, Berlin, Heidelberg, 2007. Springer-Verlag.

[13] Leah Findlater and Joanna McGrenere. Beyond performance: Feature awareness in personalized interfaces. *Int. J. Hum.-Comput. Stud.*, 68(3):121–137, 2010.

[14] Leah Findlater, Karyn Moffatt, Joanna McGrenere, and Jessica Dawson. Ephemeral adaptation: The use of gradual onset to improve menu selection performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1655–1664, New York, NY, USA, 2009. ACM.

[15] 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, pages 35–38, New York, NY, USA, 2012. ACM.

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

[17] 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, pages 207–216, New York, NY, USA, 2011. ACM.

[18] Krzysztof Z. Gajos, Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. Exploring the design space for adaptive graphical user interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '06, pages 201–208, New York, NY, USA, 2006. ACM.

[19] Krzysztof Z. Gajos, Katherine Everitt, Desney S. Tan, Mary Czerwinski, and Daniel S. Weld. Predictability and accuracy in adaptive user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1271–1274, New York, NY, USA, 2008. ACM.

[20] Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. Generating photo manipulation tutorials by demonstration. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 66:1–66:9, New York, NY, USA, 2009. ACM.

[21] Saul Greenberg. *The Computer User As Toolsmith: The Use, Reuse, and Organization of Computer-based Tools*. Cambridge University Press, New York, NY, USA, 1993.

[22] Tovi Grossman and George Fitzmaurice. Toolclips: An investigation of contextual video assistance for functionality understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1515–1524, New York, NY, USA, 2010. ACM.

[23] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A survey of software learnability: Metrics, methodologies and guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 649–658, New York, NY, USA, 2009. ACM.

[24] Tovi Grossman, Justin Matejka, and George Fitzmaurice. Chronicle: Capture, exploration, and playback of document workflow histories. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 143–152, New York, NY, USA, 2010. ACM.

[25] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. *Human mental workload*, 1(3):139–183, 1988.

[26] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 159–166, New York, NY, USA, 1999. ACM.

[27] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 256–265, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[28] 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, pages 4017–4026, New York, NY, USA, 2014. ACM.

[29] Ben Lafreniere, Andrea Bunt, Matthew Lount, and Michael A Terry. Understanding the roles and uses of web tutorials. In *ICWSM*, 2013.

[30] 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, pages 89–90, New York, NY, USA, 2011. ACM.

[31] Benjamin Lafreniere, Andrea Bunt, and Michael Terry. Task-centric interfaces for feature-rich software. In *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: The Future of Design*, OzCHI '14, pages 49–58, New York, NY, USA, 2014. ACM.

[32] 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, pages 11–18, Toronto, Ont., Canada, Canada, 2010. Canadian Information Processing Society.

[33] 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, pages 1779–1788, New York, NY, USA, 2013. ACM.

[34] 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, pages 113–122, New York, NY, USA, 2012. ACM.

[35] Wei Li, Tovi Grossman, and George Fitzmaurice. Gamicad: A gamified tutorial system for first time autocad users. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 103–112, New York, NY, USA, 2012. ACM.

[36] Wei Li, Justin Matejka, Tovi Grossman, Joseph A. Konstan, and George Fitzmaurice. Design and evaluation of a command recommendation system for software applications. *ACM Trans. Comput.-Hum. Interact.*, 18(2):6:1–6:35, July 2011.

[37] Frank Linton and Hans-Peter Schaefer. Recommender systems for learning: Building user and expert models through long-term observation of application

use. *User Modeling and User-Adapted Interaction*, 10(2-3):181–208, February 2000.

[38] Justin Matejka, Tovi Grossman, and George Fitzmaurice. Ip-qat: In-product questions, answers, & tips. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 175–184, New York, NY, USA, 2011. ACM.

[39] 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, pages 3227–3236, New York, NY, USA, 2013. ACM.

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

[41] Joanna McGrenere, Ronald M. Baecker, and Kellogg S. Booth. An evaluation of a multiple interface design solution for bloated software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '02, pages 164–170, New York, NY, USA, 2002. ACM.

[42] Emerson Murphy-Hill, Rahul Jiresal, and Gail C. Murphy. Improving software developers' fluency by recommending development environment commands. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foun-*

dations of Software Engineering, FSE '12, pages 42:1–42:11, New York, NY, USA, 2012. ACM.

[43] Andrew Sears and Ben Shneiderman. Split menus: Effectively using selection frequency to organize menus. *ACM Trans. Comput.-Hum. Interact.*, 1(1):27–51, March 1994.

[44] Michael Terry, Matthew Kay, Brad Van Vugt, Brandon Slack, and Terry Park. Ingimp: Introducing instrumentation to an end-user open source application. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 607–616, New York, NY, USA, 2008. ACM.