Supporting Users After Software Deployment through Selection-Based Crowdsourced Contextual Help

Parmit K. Chilana
June 2013

The Information School
University of Washington
Seattle, WA 98195

Thesis Committee:
Amy J. Ko, The Information School (Co-chair)
Jacob O. Wobbrock, The Information School(Co-chair)
Wanda Pratt, The Information School/ Biomedical and Health Informatics
Tovi Grossman, Autodesk Research, Canada
David Farkas, Human-Centered Design and Engineering (GSR)

Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy

University of Washington

Abstract

Supporting Users After Software Deployment through
Selection-Based Crowdsourced Contextual Help

Parmit K. Chilana

Chairs of the supervisory Committee:
Assistant Professor Amy J. Ko
The Information School

Associate Professor Jacob O. Wobbrock
The Information School

Millions of users struggle every day in using and configuring software applications to meet their needs. Even with the best efforts in user-centered design and usability, not all use cases and nuances in user interaction can be anticipated at design-time as applications increasingly become feature-rich and customizable, and development cycles become shorter. As users turn to software support to find help, they often struggle in expressing software problems using the application terminology necessary for accessing relevant help. Software teams also struggle in providing one-on-one support and learning about the prevalence of problems reported by individual users.

This dissertation introduces LemonAid, a new approach to software help that embeds users' questions and answers directly into the user interface (UI), allowing users to retrieve help by selecting a label, widget, link, image, or another UI element without ever leaving the screen. The key insight that makes LemonAid work is that users tend to select similar UI elements for similar help needs and different UI elements for different help needs. The evaluation of LemonAid's underlying retrieval algorithm showed that LemonAid can retrieve a result for 90% of help requests based on UI selections and, of those, over half have relevant matches in the top 2 results.

LemonAid was also evaluated in the field through live deployments on four web sites used by thousands of users to capture the perspectives of end users and software teams. Data was collected over 15 weeks, culminating in over 1,200 usage logs, 168 exit surveys, and 36 one-on-one interviews. Results indicated that over 70% of users found LemonAid to be helpful, intuitive, and desirable for reuse. Software teams found LemonAid easy to integrate with their sites and found the analytics data aggregated by LemonAid to be a novel way of learning about users' frequently asked questions.

The thesis demonstrated in this dissertation is:

A selection-based contextual help system that allows users to find questions and answers from other users and support staff can be helpful, intuitive, and desirable for reuse for end users and can provide new insights to software teams about frequently asked questions.

Table of Contents

List of Figures

List of Tables

Acknowledgements

CHAPTER 1

Introduction

In human-computer interaction (HCI) research and usability practice, a major focus has been on understanding users and improving the design of software before deployment. While usability practice is becoming a standard[1] in industry, millions of users continue to struggle in using and configuring software applications to meet their needs. For example, customers must decipher cryptic error messages after failed e-banking transactions, office workers wrestle with adding attachments to their company wikis, scientists struggle to export data in new formats, and new users have to interpret complex privacy settings on social networking sites. Despite upfront usability efforts by software teams, not all application-related issues can be anticipated or prevented at design-time as today's applications increasingly become customizable, and development cycles become shorter.

To resolve errors, configure settings, or complete their tasks, end users often seek software-specific help. However, most forms of software help are simply not helpful. For example, one-on-one technical support is time-consuming, frustrating, and rarely puts users in touch with the experts who have answers (Kajko-Mattsson, 2004). Documentation, tutorials, and tooltips rarely provide the task-specific help that people need (Kearsley, 1988; Rettig, 1991; Sellen & Nicol, 1995) because software designers cannot anticipate at design time the full range of *how*, *why*, *what*, and *where am I* questions that people ask (Sellen & Nicol, 1995). Even crowdsourced solutions such as technical support discussion forums require users to browse lengthy, scattered, and often irrelevant conversations in search for even one helpful reply (Lakhani & Von Hippel, 2003; Singh & Twidale, 2008). Worse yet, many users do not find these discussions at all, as the search queries that users

[1] ISO standard: ISO/TR 16982:2002, ISO 9241

write often lack precision (Belkin, 2000) and lack key application terminology (Furnas et al., 1987) necessary for retrieving relevant help.

As end users struggle in finding appropriate help, software teams face a parallel challenge in managing help requests and learning from users after deployment (Glerum et al., 2009; Hartson & Castillo, 1998; Hilbert & Redmiles, 2000). Since hundreds of questions may appear across different help mediums every day, knowing whether 5 or 5000 users have expressed the same issue can be an important factor for prioritizing support and design efforts. Some software teams instrument applications and log user interactions to understand usage of their applications and guide design changes (Hilbert & Redmiles, 2000). However, this type of logged information is not necessarily useful for understanding users' intentions related to a task (Hartson & Castillo, 1998) or knowing how many users are affected by a particular problem or confusion. Often, software teams resort to intuition (Chilana et al., 2012a) rather than a systematic way of understanding the prevalence of users' reported issues.

This dissertation is motivated by the concerns of both software teams and end users and argues for the importance of better understanding and supporting users in the post-deployment phase of software development. From the perspective of software teams, this dissertation looks at the challenges of achieving user-centered design and providing support to users in different contexts. From the perspective of end users, this research investigates the difficulties related to describing software problems and finding relevant help. Building on findings from these studies and limitations of prior approaches for software help, this dissertation introduces a new selection-based crowdsourced contextual help approach and evaluates this new approach in the field with end users and software teams.

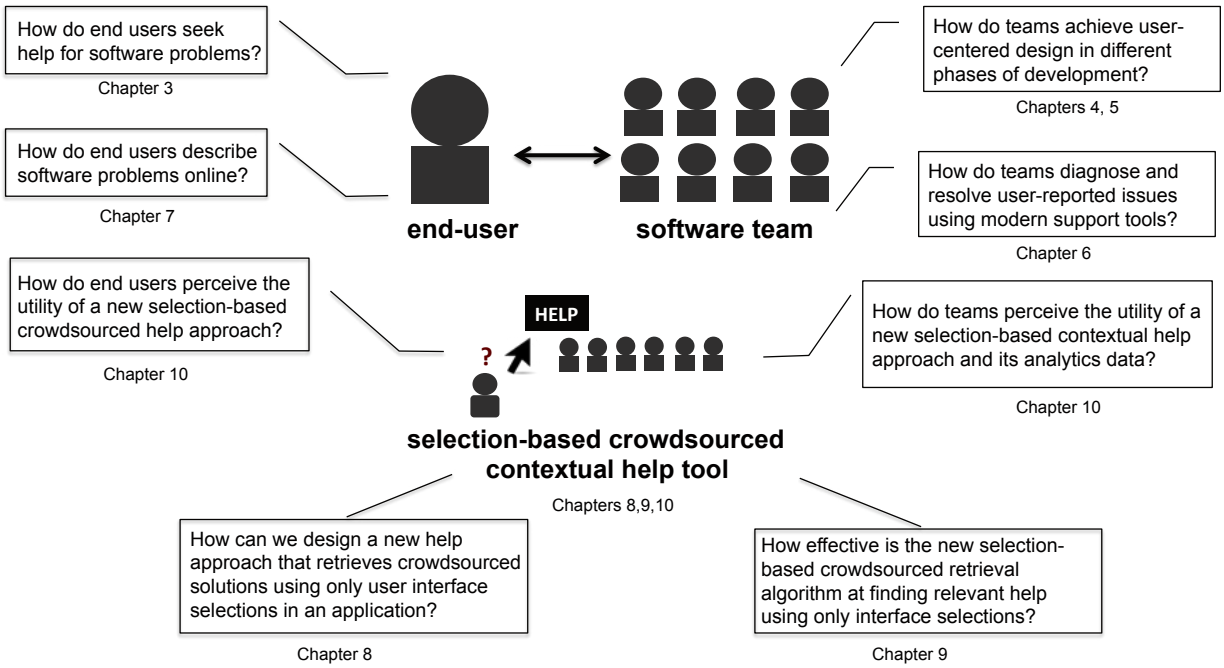


Figure 2.1 Main research questions addressed in this dissertation

The thesis demonstrated in this dissertation is that:

A selection-based contextual help system that allows users to find questions and answers from other users and support staff can be helpful, intuitive, and desirable for reuse for end users and can provide new insights to software teams about frequently asked questions.

Although the main focus of this dissertation has been to investigate this claim by designing, deploying, and evaluating a new selection-based crowdsourced contextual help system, this research has also considered the larger social and organizational factors of software design and software support practices. In addition to introducing the design of a novel help system, this dissertation includes several studies that highlight the work of software teams in different phases of software development and the interactions that occur between end users and software teams. Figure 2.1 shows an overview of the main research questions addressed in this dissertation. The sections that follow illustrate the main themes of this dissertation and its contributions.

1.1 Understanding User-Centered Software Design in the Context of Modern Organizations

By some estimates[2], there are close to 50,000 full-time usability professionals in the world and thousands of others who have part-time usability responsibilities. Despite the maturation of usability as an industrial practice, historically, usability professionals have faced an uphill battle in gaining credibility and having impact within software organizations (Gould & Lewis, 1985). To justify the use of usability guidelines and methods in software development, the concept of *cost-justifying usability* (Bias & Mayhew, 2005) was proposed to quantify the costs and benefits of usability activities. One of the main arguments made was that an upfront investment in usability would simplify interfaces and eventually reduce the need (and costs) for help and support after deployment. For example, consider this popular example cited by (Bias & Mayhew, 2005):

> …Ford Motor Company has developed an accounting system for their small car dealerships. In a usability study they found that the car dealers needed an average of three calls to the help line to just get started. The problem was discovered to be that the commands used to enter credits and debits were designed by the engineers without first consulting users to learn the commonly used abbreviations. As a result of the usability study, Ford changed not only the abbreviations but 90% of their accounting system. The new system was so easy to use that the calls to the help line dropped to zero. It was estimated that this new version saved the company $100,000. This one study more than compensated for the cost of $70,000 to set up the usability lab (Kitsuse 1991).

While a plethora of such success stories have emerged over the years and clearly show the benefits of having an upfront user-centered design focus, the reality is that help and support channels still play an important role in every organization (Nambisan & Baron, 2007). In fact, by some estimates, calls for support have been increasing[3] over the years and millions of users seek software help in online forums (Lakhani & Von Hippel, 2003; Singh & Twidale, 2008). One reason for this demand for support is that despite

[2] http://www.nngroup.com/articles/25-years-in-usability/
[3] http://www.supportindustry.com/2009supportmetrics.html

usability efforts, it simply is difficult to capture all the nuances in user interaction upfront given the diversity of applications and platforms that users use and the constraints that usability professionals face on the job (Vredenburg et al., 2002). Chapter 2 summarizes findings from a number of studies of usability and user-centered design (UCD) practice that show the difficulties of achieving upfront usability. In Chapter 4, I present a case study of design at Facebook Inc. that sheds light on other dimensions that make it difficult to achieve user-centered design in the context of modern software development, such as the scale of the user base. While some software designers in this study had the perspective that it is possible to get the design "right" upfront, most others recognized the limits of upfront design and the importance of offering help and support to end users.

Even though it may be difficult to uncover and solve all usability problems upfront, it appears that most companies still heavily invest only in upfront UCD methods (Nichols et al., 2003; Vredenburg et al., 2002). In Chapter 5, I discuss my work on understanding post-deployment usability activities, the role of usability professionals and the connection between usability and software support activities. The main finding from this study shows that the role of usability professionals appears to diminish in the post-deployment phase. I argue in this dissertation that organizations need to have an orientation towards *usability maintenance* (a concept I introduce in Chapter 5) that would encompass activities required to maintain the usability of an application in all phases of development. One approach for achieving usability maintenance may be to utilize data from help and support requests about users' problems during actual use of applications. I propose an idea for achieving this via LemonAid and its associated help analytics data in Chapter 10.

1.2 Understanding Modern Software Support Processes

Although usability professionals rarely communicate with software support specialists after deployment, these support specialists play a crucial role in helping users resolve

potential usability and technical issues after deployment. Software support channels are inundated with support requests from end users every day and cost commercial software companies millions of dollars ("Technical Support Cost Ratios," 2000). There are currently over half a million support specialists in the United States alone, with a projected 14% increase in employment by 2018. Over time, there has been a major shift in the provision of support, particularly with the advent of the web. Support specialists are no longer offering support through phone lines—they are also monitoring web-based tickets, email requests, and questions asked on discussion forums. In Chapter 6, I look at modern software support practices at Autodesk Inc. and focus on the use of modern formats in support requests, such as photos, videos, and source files.

Another way that the role of support specialists has evolved is that they are spending less time on providing one-on-one support and creating channels for users to participate in peer-to-peer or *crowdsourced* help through help forums (Steehouder, 2002; Tynan-Wood, 2010). By some estimates (Association of Support Professionals, 2012), forum visits now generate more than 30% of total support site traffic. Online help communities have proliferated, particularly with the rising popularity of "social media' and sophistication in technology used to host community-based forums. Despite the increased uptake of online forums by end users and organizations, several issues exist in the domain of software help. In Chapter 2, I summarize works related to commercial and open source support and specific challenges related to forum-based discussion of software problems. In Chapter 7, I investigate the problems that users have in describing software problems in online communities and how users' descriptions affect whether or not their issues gets resolved.

1.3 Developing a New Approach to Software Help

While crowdsourced help through forums is powerful at generating answers to help questions, locating useful answers from past discussions is a daunting task,

particularly for the less technically-savvy users who seek help. First, questions and answers are scattered across different resources: a user may post a technical help question on her social network, unaware that a similar question had already been answered on the application's forum site. Second, even if a user finds a discussion that potentially has the answer, the answer may be buried deep within long conversation threads that span multiple pages. Even though recent Q&A sites have incorporated strategies for promoting the best responses to the top of the list, users often cannot find these threads in the first place. To address these problems of discoverability, recent approaches have explored integrating crowdsourced help within the application's user interface (UI). For example, the *CHIC* framework (Stevens & Wiedenhöfer, 2006) for Eclipse adds links from each UI control to a wiki where users can author help. *TurboTax* help ("TurboTax Support," 2013) and *IP-QAT* (Matejka et al., 2011) display help discussions relevant to the current view in a sidebar within the application. Such *crowdsourced contextual help* systems have opened up a new space for integrating more relevant help in applications, but as described below, a key problem of software help retrieval remains unsolved.

1.3.1 The Problem

The problem with retrieving help from forums and user-generated discussions where end users are expected to search for relevant questions and answers is two-fold: 1) users' queries tend to be incomplete and imprecise (Belkin, 2000); and 2) queries are plagued with the vocabulary problem (Furnas et al., 1987), where different users provide different words to describe the same goal (i.e., "add a photo" *vs.* "insert an image"). While search engine algorithms can be used to mitigate some of the challenges in natural language retrieval, the onus is still on users to translate their help needs and problem contexts into keywords that result in an effective search. Apart from making help retrieval challenging, this diversity in users' vocabulary and

repetition in issue reports also makes it difficult for software teams to keep track how many users have the same question or are experiencing the same issue.

1.3.2 A Solution

Chapter 8 introduces LemonAid, a novel approach for retrieving software help that frees users from having to write textual queries, while leveraging the benefits of crowdsourcing. Through LemonAid, help is integrated directly into the user interface (UI) and can be retrieved by selecting a label, widget, link, image or another UI element without ever leaving the screen (Figure 1.2). Unlike other forms of static contextual help, the help content in LemonAid is generated dynamically based on questions and answers provided by other users. The key insight that makes LemonAid work is that users tend to make similar selections in interfaces for similar help needs and different selections for different help needs. This tight coupling of user needs to UI elements is central to LemonAid's effectiveness, reducing unnecessary variation in users' queries.

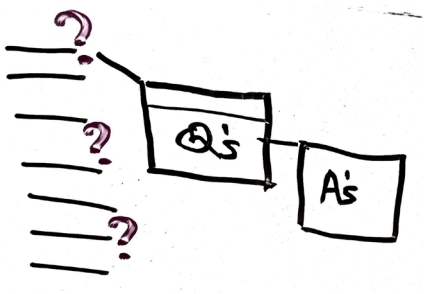


Figure 1.2: The main idea behind LemonAid is that users can retrieve help in the form of questions and answers asked by other users by selecting a label, widget, link, image or another UI element without specifying natural language search keywords

In Chapter 8, I discuss the system design and retrieval algorithm of LemonAid that makes the retrieval of relevant questions and answers possible in web applications. In Chapter 9, I present an evaluation of the underlying retrieval approach in LemonAid and, in Chapter 10, I discuss results from a large deployment study that exposed LemonAid to thousands of users. Chapter 10 also introduces help analytics information gathered from

LemonAid that aggregates information on the frequently asked questions by end users.

1.4 Research Approach

There are two main approaches to research used in this dissertation. First, I designed and developed the new selection-based crowdsourced help system using an iterative interaction design inquiry approach that is common in HCI research (Zimmerman et al., 2007). This form of research approach is grounded in the concept of *research through design* (Frayling, 1993) where the goal is to iteratively design and evaluate artifacts that can transform the "current state" to a "preferred state" for a given problem. For HCI research, new design inventions result from an integration of *true* knowledge in the form of models, theories, and formative studies and the *how* knowledge, such as engineering and technical skills (Zimmerman et al., 2007).

While there are many approaches for studying designs and user needs in HCI research, I have largely adopted an *interpretivist* research perspective for my studies. The main tenet of an interpretive research paradigm is that to understand social phenomena, we must understand the social contexts in which these phenomena are constructed (Kaplan & Maxwell, 2005; Walsham, 2006). The focus is on using qualitative and/or quantitative methods to understand the complexity of human sense-making as the situation emerges and not be confined to predefined dependent and independent variables. In adopting this research paradigm, I have often used an inductive, grounded theory approach for analyzing and making sense of data (Glaser & Strauss, 1967). Such an analysis is strongly grounded in the data and allows insights and theories to emerge from the data through different stages of inductive reasoning.

1.5 Contributions

This dissertation makes several contributions, mainly in the form of empirical findings and new design approaches:

- Synthesis and conceptualization of theoretical perspectives on software help-

seeking and derivation of design principles. (Chapter 3)

- Empirical findings that shed light on user-centered design practices in a modern software development context. (Chapter 4)
- Empirical findings that establish the current state of post-deployment usability practices. (Chapter 5)
- Empirical findings that illustrate how web-based support is practiced and how support issues are diagnosed and resolved using modern formats. (Chapter 6)
- Empirical findings that provide a conceptualization for understanding unwanted software behaviors that are reported online. (Chapter 7)
- A selection-based contextual help system that allows users to find help, and ask and answer questions within the user interface of an application. (Chapter 8)
- A new help retrieval algorithm that leverages contextual information and user interface selections to retrieve relevant help content. (Chapter 8)
- A flexible architecture for embedding the new selection-based contextual help system in any web application. (Chapter 8)
- A retrieval evaluation that demonstrates the feasibility of the selection-based contextual help approach for a crowdsourced corpus of selections. (Chapter 9)
- Empirical findings that demonstrate that the selection-based contextual help system can be helpful, intuitive, and desirable for reuse for users. (Chapter 10)
- Empirical findings that show that the analytics data gathered through the selection-based contextual help approach can provide new insights to software teams about frequently asked questions. (Chapter 10)

1.6 Organization of Dissertation

This dissertation is organized as follows:

Related Work and Theoretical Foundations
(Chapters 2-3)

Chapter 2 summarizes the works most closely related to the different themes of this dissertation. Chapter 3 discusses the theoretical and conceptual aspects of software help seeking that guide the design introduced in this research.

Organizational Context of Modern User-Centered Software Design
(Chapters 4-5)

Chapter 4 presents a case study of Facebook Inc. as an example of user-centered design in a modern software development environment. Chapter 5 focuses on post-deployment usability activities in modern software development.

Software Issue Reporting and Diagnosis
(Chapters 6-7)

Chapter 6 looks at modern software issue reporting and diagnosis by investigating support activities at Autodesk Inc. Chapter 7 looks at the challenges of reporting software problems online from the perspective of end users.

Design and Evaluation of a New Contextual Help Approach
(Chapters 8-10)

Chapters 8 introduces LemonAid and the selection-based contextual help retrieval approach. Chapter 9 presents results of evaluating LemonAid's underlying retrieval algorithm while Chapter 10 describes findings from a multi-site field deployment study of LemonAid. Both perspectives from end users and software teams are discusses in the field study.

Conclusions and Future Work
(Chapter 11)

This chapter reflects on this dissertation as a whole and some of the challenges and opportunities in the space of crowdsourced contextual help and post-deployment usability. Finally, this chapters presents ideas for future work that stem from research presented in this dissertation.

*Note that most of the work presented in this dissertation has been previously

published. I have made note in every chapter where significant portions were taken from my prior publications.

CHAPTER 2

Related Work

This dissertation builds upon a large body of work from many different fields, including human-computer interaction, computer-supported cooperative work, software engineering, technical communication, information science, management science, artificial intelligence, among others. (The theoretical and conceptual aspects of software help-seeking behavior that have helped inform the designs presented in this dissertation are presented in Chapter 3.) This chapter organizes and discusses related works in three key themes relevant to this dissertation:

1. **Software design and usability processes:** This subsection covers research on the process of user-centered software design and usability activities in organizations. Particular emphasis is placed on empirical studies of software designers and usability practitioners and their role in different phases of software development.
2. **Software support and issue reporting:** This subsection includes empirical studies of software support that highlight various organizational, process, management, and personnel issues in the provision of support in commercial and open source development contexts.
3. **Software help design approaches:** This subsection summarizes a wide range of approaches for designing software help that have evolved over the years, highlighting benefits and drawbacks of each approach.

2.1 Software Design And Usability Processes

(Gould & Lewis, 1985) were among the first to lay out principles of user-centered design and usability: an early and continual focus on users as well as empirical measurement and iterative design. A more detailed conceptualization of user-centered

design was proposed by Norman & Draper (1986) emphasizing the capture of users' perspectives in the design process and not making the users adapt to confusing user interfaces. Today, usability is an industrial standard defined as, "*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use*" (ISO 9241-11).

Despite a steady stream of new usability methods and techniques in HCI, many of these methods have not been commonly performed in practice (Gunther et al., 2001; Nielsen, 1994a). Organizations have historically been resistant to incorporating many of these usability methods into software design practice because of their perceived costs in time and resources (Bias & Mayhew, 1994). Over the years, attempts have been made to justify expenditures on usability support by bringing in quantitative cost and benefit data and showing the business value of usability (Bias & Mayhew, 1994). Jakob Nielsen also introduced the notion of "discount" usability methods (Nielsen, 1994a) that are supposedly faster and cheaper than some of the other heavyweight usability methods. He advocates for methods such as expert inspections or doing think-aloud usability tests with 5-8 users using even low-fidelity prototypes for uncovering the majority of usability problems (Nielsen, 1993).

With the continued uptake of usability in industry, many researchers have tried to formally understand what actually works in practice. The next subsection covers an overview of findings from surveys of usability professionals and UCD practice that highlight that merits and drawbacks of usability methods and assess the organizational context of usability. A summary of some prominent surveys of usability and UCD practice is presented in Table 2.1.

2.1.1 Lightweight or "discount" methods dominate practice

Surveys of usability practice over the years have consistently indicated that low-cost, informal or "discount" methods are most widely used. Even in the earlier surveys by

Hudson & Bevan (2000) and (Rosenbaum et al., 2000) the most commonly used method was discount usability testing. Surveys by Gunther et al. (2001) and Vredenburg et al., (2002) also found that the highest rated techniques were usability testing followed by prototyping and heuristic evaluations. Gulliksen et al. (2006) found in their survey of usability professionals in Sweden that field studies were ranked high on importance but used less frequently because of their costs in practice. These professionals' highest ranked methods were also usability testing and lo-fi prototyping.

The popularity of informal or discount usability methods is not surprising, considering the rapid pace of industry product cycles, the organizational constraints on design, and resistance from engineering cultures to employ formal methods. The survey findings discussed next shed light on some of these organizational constraints and how they impact the integration of usability into development.

2.1.2 Usability in the context of organizational constraints

In addition to looking at the use of different methods, some surveys have also investigated the perceptions of organizations towards usability and the interactions that occur between usability professionals and other members of software teams.

A study of software development in Switzerland by Vukelja et al., (2007) showed that the majority of development teams had no usability expert and only 8% of the developers had any formal HCI knowledge. In contrast, the survey results of (Vredenburg et al., 2002) showed that usability and UCD methods were becoming more widely used and gaining impact in industry in the US and parts of Europe. Although the results indicated that product usability had improved, it was uncertain what impact the usability integration saved product development time or costs. Some common characteristics of an ideal UCD process were not found to be used in practice, namely focusing on the total user experience, end-to-end user involvement in the development process, and tracking customer satisfaction.

Studies of usability practice in Sweden (Gulliksen et al., 2004, 2006) also found that user involvement had low priority in commercial projects and highlighted some of the challenges faced by usability professionals. For example, these challenges included working under time pressure and how, in order to maintain their credibility, usability professionals had to compromise their usability methods. Similarly, Rosenbaum et al. (2000) found that major obstacles to creating greater strategic impact included resource constraints, resistance to user-centered design or usability, lack of knowledge about usability. Venturi & Troost (2004) found that usability had a minor impact in industry and called for organizations take into account the factors related to management, infrastructure and communication of the UCD process.

With the uptake of more agile software methods in industry, some practitioners (Hussain et al., 2009; Meszaros & Aston, 2006; Chamberlain et al., 2006; Sy, 2007) have explored the space of usability in the context of agile methods. Results have indicated that these agile development methods are still lacking usability awareness and formal integration of usability or UCD approaches. These initial reports call for more research into looking at usability in agile development environments and how the existing "discount" methods can be further adapted.

2.1.3 Summary

Existing surveys of usability practice indicate that low-cost, informal methods are widely used by usability professionals. Although ethnography and other field methods are considered important, such methods are difficult to employ given the organizational constraints that practitioners face. Apart from these findings from traditional software development contexts, we are only beginning to understand usability in the context of agile software development. In particular, with the proliferation of web applications and mobile applications, product cycles are even shorter and the demands have increased for designing for millions or billions of users

STUDY	RESPONDENTS	MAIN FINDINGS
Gulliksen et al., 2004	194 usability professionals in Sweden	Commercial projects had low user involvement and emphasis on usability. Thinking aloud testing and low-fidelity prototyping were most common methods. Usability professionals were working under time pressure and felt that had to compromise their methods.
Gunther et al., 2001	100 usability professionals from HCI mailing lists (i.e., HFES, SIGCHI, STC)	Highest rated HCI techniques preferred by practitioners were usability testing followed by prototyping and heuristic evaluations.
Hudson, 2000	102 usability professionals from HCI mailing lists	Most commonly used methods were informal usability testing, user analysis/profiling, evaluating existing systems, low-fidelity prototyping.
Rosenbaum et al., 2000	134 usability professionals at CHI 1999	Usability testing was the most commonly used method. Organizational obstacles were resource constraints and resistance to user-centered design or usability.
Vredenburg et al., 2002	103 usability professionals from the US and Europe	Top UCD methods were iterative design, usability evaluation, task analysis, and informal expert review. UCD adoption is increasing but cost and time savings in development are unclear.
Venturi et al., 2004	83 usability professionals at NordicCHI	Interviews and prototyping were most frequently used methods. Overall low impact of usability in industry; impact is affected by management, infrastructure and communication issues.
Vukelja et al., 2007	Software developers in Switzerland	Majority of development teams were developing interfaces without formal usability experts; software developers rarely carried out usability testing.

Table 2.1 Summary of surveys of user-centered design and usability practices

across the world using different platforms. I explore these issues in my case study of Facebook Inc. presented in Chapter 4, looking particularly at constraints of doing user-centered design for a modern social networking application used by over a billion users.

While upfront user research and prototyping are crucial to designing user-centered

products, learning from users in the deployment phase about their *actual* use of the product is also valuable (Nielsen, 1993). However, my literature review indicated that research on usability practices has largely centered on upfront UCD methods, with little work substantiating how usability is actually practiced in the *post-deployment* phase. While surveys of software developers working on post-deployment maintenance tasks (Layzell & Macaulay, 1990; Singer, 1998) have been conducted for over two decades, we know little about the work of usability professionals in the post-deployment phase. In Chapter 5, I present a survey of post-deployment usability and show how the role of usability professionals appears to diminish in the post-deployment phase.

2.2 Software Support in Organizations

In the post-deployment phase, support specialists are at the frontlines of interacting with end users and helping them resolve their technical and usability issues. The concept of technical support or product support encompasses a wide range of services by which enterprises provide assistance to users of technology products such as mobile phones, televisions, computers, software products or other electronic or mechanical goods (Nambisan & Baron, 2007). The goal of such support services is to help the user solve specific problems with a product. Software support (the focus of this dissertation) is a form of technical support that is tailored for software applications (Tourniaire & Farrell, 1997). Most commercial organizations offer support for the software they sell, either freely available or for a fee and support can be delivered over the telephone, by e-mail, through web-based incident reporting services, or more recently, through discussion forums. Open source software products also have support available through online discussion forums where incidents can be reported and discussed with other users and software developers (Lakhani & Von Hippel, 2003).

Since studies focused solely on software support practices are scarce, I have drawn upon a large body of work investigating issue reporting and troubleshooting in other

forms of technical support, and organizational aspects of providing support to end users.

2.2.1 Problems with User-Reported Issues

A main theme in studies of issue reporting across different domains suggests that end users are usually not successful in: 1) understanding what issue to report; 2) how to report it; or 3) how to potentially resolve the issue.

For example, Crabtree et al. (2006) studied immediate and remote help giving at a library help desk and a call center of a printer manufacturing company. They explain that users lacked precise technical knowledge of the machine that they used and had limited understanding of how one might get from the observed features of a problem to some sort of a resolution. Similarly, Poole et al. (2009) studied remote troubleshooting in the domain of home networking and analyzed phone calls that customers made to networking support. Their findings illustrate that end users had a difficult time in being precise about the issue that they were experiencing. The findings further show the limitations of telephone-based support requests in that they prevented the customer and the support specialist from having a shared understanding of the problem. Fussell et al., 2000 also argue that a shared visual space is essential for collaborative repair because it facilitates *conversational grounding*. The web offers many opportunities for potentially improving conversational grounding through the use of shared screens, multimedia attachments, and live conferencing tools. I discuss some of these tools and their use in diagnosis of software issues in Chapter 6.

Jiang et al., 2009 performed a quantitative analysis of over 600,000 customer support cases from NetApp, a commercial storage and data management system and looked at issue reports of failures that occur in storage systems. Their findings suggest that cases that were reported with system logs attached were more likely to get resolved earlier; however, end users usually had a hard time in providing this information and the logs that were provided usually contained noisy information that was not pertinent to the diagnosis

of the underlying problem. Similarly, in the domain of bug reporting, studies show that reports containing stack traces get fixed sooner, are easier to resolve, and have increased probability of being fixed (Bettenburg et al., 2008; Breu et al., 2010; Aranda & Venolia, 2009). However, users have the most difficult time in providing stack traces, test cases and clear steps to reproduce, making it challenging for developers to diagnose the underlying issues.

2.2.2 Organizational Context of Front-End Support

In addition to issue reporting, several studies of support have looked at the work of front-end support specialists. These studies have been carried out in the context of help desks, commercial fee-based support, and web-based support.

One class of studies focuses on the customer satisfaction and quality issues in front-end support. For example, Tourniaire & Farrell, 1997 discuss various case studies and make several recommendations on how to create help desks and support centers that foster greater satisfaction among customers. In the context of web-based support, Negash et al., (2003) discuss dimensions of quality and effectiveness in web-based customer support systems. They test an existing quality dimension model for web-based customer support systems and offer guidelines for managing such systems and maintaining service quality in web-based support.

Another set of front-end support studies examine ways of improving retrieval of known issues from support requests. For example, Halverson et al. (2004) present a detailed analysis of help-desk support by gathering observational data at a large software company and illustrate organizational barriers in codifying and reusing help desk knowledge. They explain that FAQs appeared to evolve out of a "chaotic" process driven by different social and technical changes rather than a systematic knowledge management process. Das (2003) used log linear modeling technique to identify the bases of productivity in technical support work and the different types of knowledge that come

into play. This study was also done from a management perspective and suggested that more systematic and efficient information retrieval tools are necessary for searching for known issues and resolving them faster.

Some studies have also investigated organizational and process issues, and the roles of customers and support specialists in the provision of support. For example, a large-scale study by Kajko-Mattsson (2004) sheds light on various front-end support practices within Swedish organizations. Their findings indicated that complexity and variation in platforms and applications used by end users adds to the challenges of delivering support. The findings about process issues in support from multiple organizations allowed them to develop CM3, an upfront maintenance model specific to front-end support. In the context of web-based support, Wiertz & de Ruyter (2007) and Nambisan & Baron (2007) investigated the role of customers in online help communities from organization studies and marketing perspectives. Their findings suggest that customers' participation in support activities is affected by perceptions of how they may benefit from these interactions. And, customers' interactions and support experiences on the web also shape customers' overall attitude towards the organization. Similar findings about end user participation have also been discussed in the studies of open source communities (Lakhani & Von Hippel, 2003; Singh et al., 2006).

2.2.3 Summary

This section covered a wide range of studies on support practices and issue reporting from a variety of different domains. Despite the abundance of literature in this area, many opportunities exist for further understanding support practices in the context of web-based support and modern tools. In Chapter 6, I discuss my study of support specialists and web-based support at Autodesk Inc. that complements other studies in this space. In particular, this study focuses on the use of modern support tools and formats for submitting and diagnosing support issues. I also investigate issue reporting

online and users' expression of unwanted software behaviors and how they impact resolution of issues in Chapter 7. Lastly, I look at how support teams can learn more from users based on questions they are asking and benefit from the concept of "help analytics" in Chapter 10.

2.3 Design of Help Systems

The previous subsection looked at the process of software support and help provision in organizations. In this subsection, I present a synthesis of different approaches that have evolved in the design of help systems that offer end users ways of resolving issues on their own.

2.3.1 Documentation and Manuals

Paper-based documentation and manuals were among the first generation of help tools that provided mostly static help content (Rettig, 1991). Soon, these manuals were upgraded into online repositories of documentation and researchers have since spent much time in figuring out how to best represent, maintain, and evolve these help repositories (Comeau & Milton, 1993; Chamberland, 1999; Lee & Lee, 2007). Tutorial systems using documentation information have also been explored in-depth, teaching users how to use a system by example. For example, the Stencils tool (Kelleher & Pausch, 2005) shifts the user's attention on certain parts of the interface during a tutorial to prevent errors. Apart from forcing users to leave the context of their tasks, the main limitations of these forms of help are that they offer static help: users are forced to find help that relates to their situation, and when they do, the help rarely answers their specific question because it is too general (Turk & Nichols, 1996). Furthermore, since documentation is written and controlled by software teams, there is often a lag between when a problem is identified and when help is written for it (Ames, 2001; Grayling, 1998).

2.3.2 Context-Sensitive Help

Context-sensitive help or contextual help systems (Sukaviriya & Foley, 1990) evolved as the next generation of help systems where the idea was to attach help to specific UI controls. Researchers have explored a variety of ways to invoke this help, including tooltips, special modes as in the "?" icon in some Windows dialog boxes, Balloon Help (Farkas, 1993) in early Apple systems, pressing F1 over user interface elements, and even choosing a command name to see animated steps (Sukaviriya & Foley, 1990), and more recently videos in tool tips (Grossman & Fitzmaurice, 2010). Still, these forms of context-sensitive help are limited in that the help presented is static and limited to explaining the functionality of a widget. Another limitation is that software teams must *anticipate* where users might seek help, so that they can author it at design-time and attach it to UI controls. As will be discussed in Chapter 8, LemonAid addresses this issue by letting *users* decide where help should be embedded, authoring that help at run-time.

2.3.3 Adaptive Help

Adaptive help attempts to overcome context-insensitivity by monitoring user behavior for opportunities to help (Delisle & Moulin, 2002; Finin, 1983; Palanque et al., 1993; Pangoli & Paterno, 1995). These systems make an explicit effort to model users' tasks, often employing AI techniques to predict and classify user behavior, some even using speech recognition (Hastie et al., 2002). Perhaps the most well known is "clippy" in Microsoft Word (which was a simplified version of a more successful intelligent agent (Horvitz, 1999). Other systems have explored ways of modeling user intent and plans (Virvou & Kabassi, 2000). Although powerful, these systems are limited by their ability to model and infer users' intent, meaning that the static help that they provide can often be irrelevant. Moreover, these systems may interrupt at inappropriate times and are often perceived as being intrusive. An ambient and unobtrusive approach that has

recently emerged is feature recommendation based on monitoring of application usage (Matejka et al., 2009). Still, in all of these cases, help is tied to functionality rather than user's intentions and application use.

2.3.4 Automatic Help

Another class of help tools manifest as automatic help tools. Rather than inferring users' intent, such tools enable users to explicitly state their problems to obtain customized help. For example, SmartAidè (Ramachandran & Young, 2005) allows users to choose a particular application task, and AI planning algorithms generate step-by-step instructions based on the current application state. The Crystal system (Myers et al., 2006) allows users to ask "why?" questions about unexpected output by simply clicking on the output itself. While such help techniques are powerful in generating customized solutions to users' help requests, they can only answer a limited class of questions amenable to automatic analysis. They also often require significant adaptations to an applications' code to provide useful answers.

2.3.5 Crowdsourced Help

Among the long history of approaches to software help, perhaps the most powerful approach is *crowdsourced* help. With crowdsourced help (*e.g.*,Lakhani & Von Hippel, 2003; Morris et al., 2010; Singh et al., 2006; Steehouder, 2002) users can help each other answer questions in discussion forums, mailing lists, or within their online social networks. Such resources reinforce the social nature of technical support that users tend to prefer (Singh et al., 2009; Twidale & Ruhleder, 2004) and companies also benefit, as they have to expend fewer resources on support. The main idea is that the user community can generate solutions to help requests more quickly than any tool or in-house support team (Harper et al., 2008). Early research examples of this approach, such as AnswerGarden (Ackerman & Malone, 1990), focused on organizational support and exchange of expertise; similar ideas emerged in the open source

community in technical support forums (Singh et al., 2009).

2.3.6 Crowdsourced Contextual Help

Although crowdsourced solutions can be powerful at generating answers, discovering relevant help content from often lengthy, scattered, and irrelevant conversations in forums can be daunting. Worse yet, many users fail at finding anything relevant by searching because the search queries that they write often omit key application terminologies necessary for retrieval (Furnas et al., 1987).

To address these problems of discoverability, recent approaches have explored the integration of crowdsourced help within the application's user interface (UI). For example, the *CHIC* framework (Stevens & Wiedenhöfer, 2006) for Eclipse adds links from each UI control to a wiki where users can author help. *TurboTax* help ("TurboTax Support," 2013) and *IP-QAT* (Matejka et al., 2011) display help discussions relevant to the current view in a sidebar within the application. As will be discussed in Chapter 8, LemonAid goes further by letting users decide which aspect of the interface matters for particular problems and allows users to author and discover help. Furthermore, LemonAid is not tied to any particular application structure.

2.3.7 Summary

My analysis of the existing methods of software help suggests there are two major issues:

1. Sophisticated automated software help retrieval systems that can detect context based on a user's selection or even monitor the user's action in the background have a lot of potential in providing relevant and targeted help to users. However, the help content that these tools currently support is static and often too dependent on the technique used for automation.
2. The help that users get through crowdsourced or community-based help systems is dynamic, task-specific, and offers more *utility* to users. However, this type of help is

disconnected from the user's context and the only way for users to succeed in finding relevant help is if they use sophisticated search strategies.

In Chapter 3, I discuss the theoretical aspects of software help seeking behavior. Taking into account Chapter 2's analysis of existing help systems and their relative strengths and limitations, I derive a set of design principles that can help us design modern software help tools. And, in Chapter 8 I introduce my LemonAid system that is driven by these design principles.

CHAPTER 3

Theoretical Aspects of Software Help-seeking

In this chapter, I consider the end user's perspective as a help seeker and theories that inform software help-seeking behavior. To frame my discussion of theories relevant to software help seeking, I first consider the different stages involved in a help-seeking episode. I then discuss major theories and concepts from different disciplines that help us understand each stage of help seeking.

3.1 Stages of a Help-seeking Episode: An Overview

Software help is considered to be one of the most difficult problems in human-computer interaction (Covi & Ackerman, 1995): effective help provision requires an understanding of the application and system environment, the task at hand, and the user's individual characteristics (Kearsley, 1988). Kearsely depicts these three dimensions as individual axes in a cube model of help (Figure 3.1) and contends that a user may lack skills or knowledge in one or more of these dimensions.

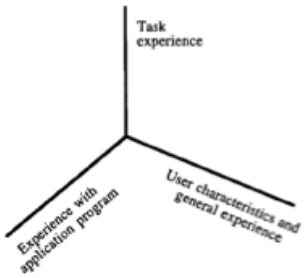


Figure 3.1: Three dimensions of help (Kearsley, 1988)

Despite an abundance of literature on help and documentation, information-seeking, troubleshooting, and software help systems, there are few conceptual explanations of software help-seeking behaviors that take into account modern software help tools. Today's end-users are not limited to looking up information in a manual or asking a friend: they can search online, post a question in a discussion forum, or have a

live synchronous chat with a support specialist. To better understand software help-seeking behaviors, I bring together theories and concepts scattered in different domains and consider stages that encompass a help-seeking episode.

An episode of help seeking is triggered when a user experiences a *breakdown* while interacting with a software application (Figure 3.2). The episode concludes with a user finding relevant information and applying that information to *resolve* the breakdown. In between the breakdown and the resolution stages are various information-seeking and diagnostic activities that the user may initiate individually or in collaboration with others to resolve the breakdown. The help-seeking episode is depicted as being cyclic rather than linear because the attempted resolution may not resolve the breakdown and could lead to further breakdowns (the concept has been adopted from Norman's (Norman, 1990) idea of continual feedback loops in user-system interaction).

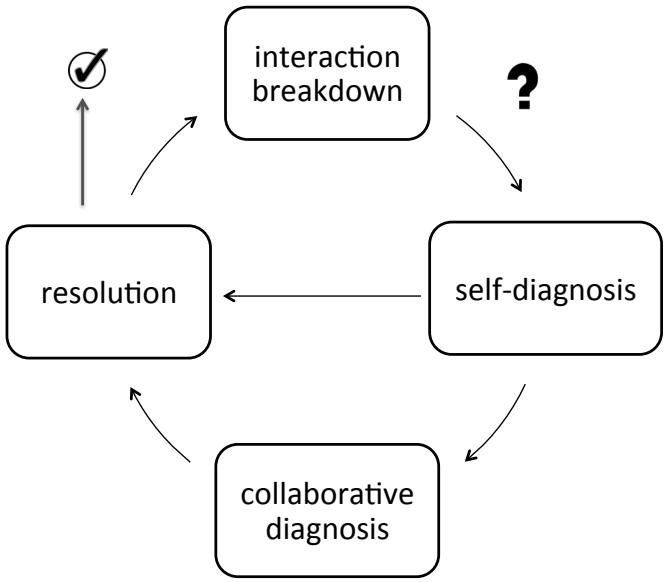


Figure 3.2: Stages of a help-seeking episode

The stages of software help-seeking behavior roughly consist of:

1. ***interaction breakdown***: the initial breakdown(s) in user-system interaction that trigger the help-seeking episode; help-seeking strategies are influenced

by users' goals, actions, and context.

2. ***self-diagnosis:*** self-directed diagnostic and information-seeking strategies that users engage in before seeking help from others; users may be able to achieve a resolution through their own diagnosis or may move on to the collaborative diagnosis stage.
3. ***collaborative diagnosis:*** when a breakdown occurs, users turn to ask an expert, a friend, or even another user for help; research shows that users tend to first wrestle with the breakdown on their own before engaging in a collaborative diagnostic **process.**
4. ***resolution:*** the processes involved in understanding and applying the acquired knowledge to resolve the breakdown.

To understand the theoretical basis of software help-seeking behavior, I look at each stage in the help-seeking episode in detail.

3.2 Interaction breakdowns—triggers for help seeking

There are many instances in a normal interaction with a software application when a user may experience the need to consult a help system or ask someone for help. Software help-seeking is not limited to resolving errors (Novick et al., 2007)—users' questions have been shown to fall under 5 broad categories (Sellen & Nicol, 1995):

Goal-oriented: What kinds of things can I do with this program?

Descriptive: What is this? What does this do?

Procedural: How do I do this?

Interpretive: Why did that happen? What does this mean?

Navigational: Where am I?

One way to understand these questions and related help-seeking activities is to understand the psychology of user-system interaction. Literature from cognitive science and psychology that guided much of the early work in HCI can help us understand the

breakdowns that occur in user-system interaction and how users' goals, situated actions, and context influence help seeking.

The notion of a breakdown has been proposed by many philosophers, such as Heidegger (Heidegger, 1978), and adopted in HCI. For example, Winograd and Flores (Winograd & Flores, 1986) build on Heidegger's notion that objects and properties are not inherent in the world, but arise only in an event of breaking down, in which they change from "ready-to-hand" to "present-at-hand" (Winograd & Flores, 1986). Therefore, a breakdown occurs with "transparent" operations when something impedes their execution. Breakdowns have been instrumental in the analysis of human–computer interaction and facilitate the detection of usability problems of applications with traditional computer-based systems. The breakdown uncovers an aspect of the design task and is a source of learning. Schön (Schon, 1992) argues that breakdowns in action function as catalysts for interpretive reflection. In the context of software use, when a user sees a cryptic error message, cannot figure out what to do next in a task, or wants to customize an unknown setting, she is experiencing such a breakdown in interaction: normal software use may be transparent, but all of a sudden there is awareness and need to figure out the software to resolve the breakdown.

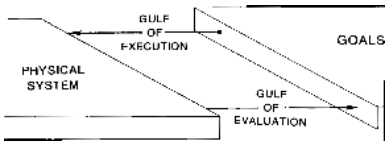


Figure 3.3: Gulfs of execution and evaluation (Norman, 1990)

Norman (1990) provides another perspective in understanding the theory of how users interact with systems in terms of goals and intentions. A user starts off with a goal expressed in psychological terms, whereas the system presents its current state in physical terms. These differences in goals and states create *gulfs* that need to be bridged for the user to successfully use the system. (Figure 3.3). The *gulf of execution* represents the difficulty the

user has in translating a psychological goal into a physical action or the difference between the intentions of the users and what the system allows them to do or how well the system supports those actions. Many help systems are designed to help with the gulf of execution: teaching users how to perform actions, primarily to learn about a command they already know the name of, or learn how to perform tasks. The *gulf of evaluation* represents the user's difficulty in evaluating whether the response of the computer system meets the desired goal or the intentions or expectations of the user. Few help systems allow users to interpret what they are seeing on the screen and determine how to fix it if it is not what they intended. To explain how users overcome these gulfs, Norman (Norman, 1990) describes the planning model: in interacting with a system, users have a goal, a plan to achieve the goal, and a strategy for executing the plan. User's actions alter the state of the world; the new state is perceived, the perceptions are interpreted, and an evaluation is made with respect to the user's original goals.

This "plan-based" approach described by Norman and other cognitive scientists has been contested by other scholars who advocate that a user's ability to interact with a system depends on distributed task knowledge (Suchman, 1994). For example, Suchman's theory (Suchman, 1994) emphasizes *situated actions* rather than the planning model. Suchman argues that plans and explicit procedures are not the only dimensions of guiding our behavior and insists that plans are only resources for situated actions. According to her, a situated action is usually "transparent," and only "when situated action becomes in some way problematic, rules and procedures are explicated for purposes of deliberation and the action" (p. 22). Suchman's main point is that people often have plans of action mapped out in their heads, but may need to change that plan depending on what is actually happening in a specific situation by using their skills or past experiences.

Theories both from cognitive and sociological perspectives are valuable in understanding the breakdowns and triggers that underlie users' help-seeking needs and

how these triggers influence users' help-seeking strategies.

3.3 Self-Diagnosis and Search for Resolution

When users experience a breakdown, they tend to begin a process of self-diagnosis and trial-and-error to find a resolution before seeking help from other sources or people. Carroll calls this the "paradox of the active user" (Carroll & Rosson, 1987) because even though it would be more efficient for users to learn the proper way of resolving their breakdowns, they opt to tackle the task on their own first (Carroll, 1998; Novick et al., 2007). Many studies have confirmed that users resist using formal documentation and manuals and continue to struggle on their own (Patrick & McGurgan, 1993; Rettig, 1991). The limitation of documentation-oriented forms of help is that they are decontextualized and static: users are forced to find help that relates to their situation, and when they do, the help may be too general and not specific to the task at hand (Knabe, 1995). Moreover, because documentation is written usually at design time (Goodall, 1992), the proposed solution in the documentation may in fact be outdated (Goodall, 1992; Patrick & McGurgan, 1993; Rettig, 1991).

The advent of online help documentation with search capabilities has offered an improvement over printed documentation, but still the static nature of the help content remains to be a problem (Goodall, 1992). Recently, the web has become a new platform for exchanging software troubleshooting experiences in discussion forums and thousands of users ask questions and post answers on these forums every day (Lakhani & Von Hippel, 2003; Singh & Twidale, 2008). Although these forums offer a repository of rich context-specific troubleshooting solutions, they are not easily navigable or searchable (Singh & Twidale, 2008). (The section on collaborative diagnosis discusses the challenges of online forums in more detail.)

We can begin to understand the different dimensions and challenges of retrieving help from the user's perspective by looking at the vocabulary problem in human-system

communication and models and theories of information-seeking.

3.3.1 The Vocabulary Problem

In all types of user-system interactions, vocabulary and naming conventions used in the system interface play a key role for the user in understanding the system's functionality. However, research shows that two users rarely describe an object using the same name: in fact, (Furnas et al., 1987) found that when people were asked to spontaneously name objects in different domains, the probability that two people would pick the same term was less than 20 percent. Figure 3.4 shows the results of multiple experiments that Furnas et al. (1987) carried out in 5 different application domains: text editor, message decoder, common objects, classified, and recipe keywords. They found that the probability of two people applying the same term to an object was 0.07-.18. Furnas et al. (1987) contend that this fundamental property of language constrains design methodologies for vocabulary-driven interaction. The vocabularies that system designers select may not have any overlap with users' preferred terms and can thus lead to interaction bottlenecks and breakdowns in communication. The vocabulary problem has been a major cause of problems in information retrieval because indexing and searching methods rely on the underlying vocabularies.

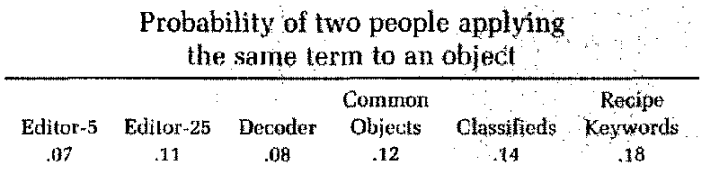


Figure 3.4: Probability of two people applying the same term to an object is 0.07-.18 (Furnas et al., 1987)

3.3.2 Models of Information-Seeking

In addition to the vocabulary problem, the overall process and strategy employed by the user to find relevant information influences the success of help-seeking initiatives.

The classic information retrieval model is understood as a simple equation where you have an information need (represented as a query) that needs to be matched against a document in a collection (Bates, 1993). However, information-seeking models based on empirical studies have contested this classical retrieval approach and revealed the nuances and complexities of information-seeking in practice.

3.3.2.1 The berry-picking model

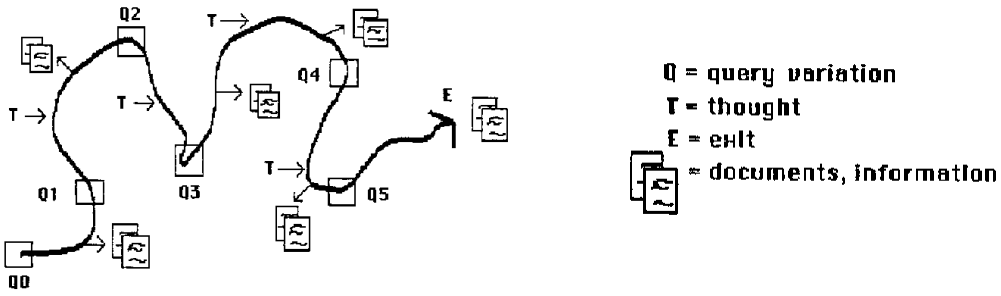


Figure 3.5: Model of a "beryypicking", evolving search (Bates, 1993)

Bates (1993) proposed a different perspective to the classic information retrieval model: the *berry-picking model* for information-seeking. Bates claimed that search for information was rarely a static event requiring the use of one strategy to find an answer. She disputed even the retrieval models that accounted for feedback and query refinement because in these models the underlying information-seeking process was still linear, focusing on an incremental progress towards a goal. In studies of information-seeking behavior, Bates observed that the information-seeking path was rarely linear: it was a looping and meandering path, resembling more of a process that someone may use when picking huckleberries (Figure 3.5). Bates claimed that information-seeking begins as a path and users pick bits of information one at a time and head in different directions depending on where they anticipate more 'berries' or 'patches', which are clusters of berries. Users tend to not only refine queries, but also modify their search and retrieval strategies as a whole. Also, users may get diverted

from their original path because of other information they encounter along the path by serendipity.

3.3.2.2 Information foraging theory

Another popular theory that describes information retrieval behavior is *information foraging* proposed by Pirolli and Card (1995). The main premise of the information foraging theory is that information-seeking is a rational, goal-driven activity: users will tend to minimize the amount of energy, time, and thinking they have to spend to get the information they need via searching or browsing. This theory is rooted in the optimal foraging theory in biology that explains an animal's food preferences and feeding strategies. When users specify a search query, they have a particular goal and anticipated results (even though the query may not reflect that in natural language). In cases where users are browsing, they follow an iterative pattern of understanding the state they are in, setting the next goal and plan for the next interaction, and evaluating the results simultaneously. If users are browsing with no particular goal, they stumble upon information by serendipity because users are still driven by an intention and are constantly evaluating the environment they are in. When users sense that the probability of finding relevant information is high, their engagement remains high and they remain aggressive in following the *scent* of information which they sense from navigation cues and metadata.

3.3.2.3 Browsing-based information seeking:

Although a lot of focus in Information Science has been on optimizing query-based information retrieval, some researchers have extensively studied browsing activities and shown their importance in information-seeking (Belkin, 2000; Choo et al., 1999; Marchionini, 1997). Empirical studies (Marchionini, 1997) have shown that there are three types of browsing that may be differentiated by the object of the underlying information need and by the information-seeker's tactics: 1) *directed browsing* that

occurs when browsing is systematic, focused, and directed by a specific object or target (e.g., scanning a list for a known item); 2) *semi-directed browsing* that is in play when users browse in a predictive or purpose way, but the target is less definite and browsing is less systematic (e.g., example is entering a single, general term into a database and casually examining the retrieved records); and 3) *undirected browsing* that occurs when there is no actual goal and very little focus (e.g., . flipping through a magazine and "channel-surfing"). Browsing also facilitates a type of exploratory learning and discovery not possible in query-based search, but modern information retrieval systems focus a lot on natural language querying (Choo et al., 1999; Marchionini, 1997). One advantage of browsing over query-based searching is that users can often recognize the subject of their need rather than feeling the burden to specify it precisely (*recognition vs. recall)*.

In summary, information-seeking models reveal the complexity of users information-seeking strategies (beyond just formulating a precise query). Few help systems currently help users mitigate the vocabulary problem or offer alternate navigation paths or browsing mechanisms (Sellen & Nicol, 1995), making it difficult for users to self-diagnose their software issues.

3.4 Social interactions and cooperative work in diagnosis

Given the difficulty of self-diagnosis and challenges of finding help online, many users opt to ask an expert, a friend, or even another user for help. Help-seeking and problem resolution is increasingly being recognized as a social and collaborative process (Crabtree et al., 2006; Poole et al., 2009; Twidale & Ruhleder, 2004).

I discuss how diagnosis of software issues is a social process and how we can understand different aspects of help giving in co-located and remote contexts.

3.4.1 Diagnosis as a social process

Crabtree et al. (2006) have suggested that the use of help systems relies on *articulation*

work. Articulation work is defined as a "temporarily unfolding stream of talk that (if successful) illuminates problems and solutions (Crabtree et al., 2006), p. 220). The articulation can be formal or informal and the conversation can occur between a user and an expert (such as a support specialist), or a user and another peer user.

Orr's (1996) work on machine troubleshooting has shown that successful diagnosis happens through a "narrative process" where the user and the technician create a coherent description of the troubled machine through conversation. The type of diagnosis process that technicians engage in (i.e., conversations) is not necessarily a part of their official job description, but vital to the success of their work. The narrative descriptions are helpful not only in remedying the problem at hand, but also in becoming part of a larger social process: stories of users and machine troubles circulate among technicians and become one of the primary means for technicians to stay informed of the developing subtleties of machine behavior. Orr emphasizes that technical knowledge should be viewed as a socially distributed resource that is stored and diffused primarily through an oral culture. Other studies on troubleshooting have confirmed that experts who are trained in diagnosing and resolving technical issues tend to focus more on experimental knowledge and conversations, rather than documentation or codified knowledge in databases (Halverson et al., 2004; Yamauchi et al., 2003).

Studies have shown that users often prefer to ask other users within their environment when they experience a breakdown (Crabtree et al., 2006;Poole et al., 2009; Twidale & Ruhleder, 2004). The diagnosis and resolution also relies on conversations and experimental knowledge between users, as observed by Orr. With the advent of the web and online discussion forums and social networking sites, the scope of peer-to-peer help has expanded (Lakhani & Von Hippel, 2003; Singh & Twidale, 2008). Studies show that most questions posted by a help seeker are answered quickly (often within a few minutes) and most answers received are judged to be valuable by the help seekers (Lakhani & Von

Hippel, 2003; Singh & Twidale, 2008). When a help request is initiated by a help seeker, another user may try to help and the request may be resolved. But, often the interaction is non-dyadic: several other people pitch in to try and help diagnose the problem, propose alternate solutions, or try to clarify what's being discussed.

Even though diagnosis is increasingly being recognized as a social process and tools are being built to facilitate conversations among help seekers and specialists or other users, a major hurdle in effective conversations is the effort required to achieve "common ground" (Clark & Brennan, 1991) and shared understanding of the context of the problem.

3.4.2 Establishing common ground

One of the key challenges that emerges in diagnosis based on conversations, particularly remote conversations, is the issue of establishing *common ground*. Common ground is defined as the knowledge, beliefs and suppositions participants share about the activity, and that accumulate over the course of actions. Clark and Brennan's (1991) theory suggests that participants in a conversation generally seek to minimize the effort required to communicate or develop common ground. In each joint activity or conversation, participants face "coordination problems" which cause difficulty in inferring actions, words, and expectations of other participants in relation to the goal of the joint activity. To solve coordination problems and establish common ground, participants may use different linguistic strategies or coordination devices. From the perspective of computer-mediated communication, different communication media may have different costs associated with them and may affect different parts of the grounding process (Fussell et al., 2000). For example, sending an email message requires more effort than speaking on the phone. However, it may be easier to read complex instructions from the screen than listening to someone read them on the phone. In the context of software help, researchers have discussed how to improve grounding in discussion forums (Singh & Twidale, 2008) and collaborative visual

repair tasks (Fussell et al., 2000).

3.5 Applying knowledge to resolve breakdown

So far, we have seen what is meant by an interaction breakdown, how people have strategies for resolving an issue on their own, or in collaboration with other users or specialists. The last part in help-seeking episode shown in Figure 3.1 is the application of the acquired knowledge to actually resolve the breakdown. Whether or not a user will be able to apply the knowledge from a manual, a discussion board, or a verbal conversation is influenced by a number of factors, including the complexity of the involved steps, conceptual understanding of the software, and the user's technical knowledge and skills (Kearsley, 1988). Although few works provide any direct theoretical perspective on how users apply knowledge to resolve breakdowns, we can develop an understanding of the process from theories that shed light on how users appropriate technical instructions and learn new software behaviors.

3.5.1 Minimalism in Explanations

Carroll offers the "minimalist theory" in writing technical instructions for users (Carroll, 1998). He manifests this theory in the *minimal manual* (Carroll et al., 1987) that was designed to address difficulties users had with other types of self-instruction manuals in learning to use complex applications. Carroll contends that this minimal approach helps new learners of software to coordinate their attention between the system and the manual; it specifically trains error recognition and recovery; it better supports reference use after training. The main concepts proposed in the minimalist theory are that (1) all learning tasks should be meaningful and self-contained activities, (2) learners should be given realistic projects as quickly as possible, (3) instruction should permit self-directed reasoning and improvising by increasing the number of active learning activities, (4) training materials and activities should provide for error recognition and recovery and, (5) there should be a close linkage between the training

and actual system. The overarching theme in the minimalist theory is to minimize the extent to which instructional materials obstruct learning and focus the design on activities that support learner-directed activity and accomplishment.

3.5.2 Learning through demonstration

Eales & Welsh (1995) explored the idea of explicitly designing for collaborative learnability where incremental improvement was sought in the course of using software. They emphasized the central importance of demonstration to the success of the learning episode, noting that visual demonstrations can be accompanied with verbal commentary to highlight significant events in an explanation. In addition to minimalism, Eales and Welsh (1995) emphasized the importance of software tutorials to incorporate text and graphics demonstrations of how to interact with the system. They contended that demonstrations were more effective in situations where there was some measure of mutual understanding and commitment along with shared or similar problems. They particularly focused on the importance of making tool use visible by capturing, storing and sharing activity. Several software help tools have explored the idea of animated demonstrations (Harrison, 1995; Palmiter et al., 1991; Spannagel et al., 2008) and become mainstay in documentation and online help tutorials.

3.5.3 Situated and Contextual Learning

For learning complex tasks with software, demonstrations can certainly be effective. However, still if they are tied to static documentation, their use will be minimal (Rettig, 1991). As mentioned in the previous section, since diagnosis and help-seeking is a social activity, one of the most effective ways to quickly learn new complex instructions are through over-the-shoulder-learning (Twidale & Ruhleder, 2004) or through examples (Tomasi & Mehlenbacher, 1999). Anderson (Anderson et al., 1984) has explored this idea particularly in the design of "computer tutors" and found that learning in the problem context should decrease the difficulty of encoding procedural

knowledge. Instruction is more effective when provided in context. Has been shown in the context of computer tutors. The kind of learning has characteristics of being brief, ad hoc, informal, strongly situated (Lave & Wenger, 1991). Situated learning is a general theory of knowledge acquisition that emphasizes that learning is a gradual acquisition of knowledge and skills as novices learned from experts in the context of everyday activities (Lave & Wenger, 1991). The key idea is that people do not employ formal approaches to solving problems in everyday thinking or interacting with objects (as they may do in a formal classroom environment). Thus, to facilitate everyday problem solving, the theory of situated learning suggests that knowledge needs to be presented in an authentic context, i.e., settings and applications that would normally involve that knowledge and that learning should be facilitated by social interaction and collaboration.

3.6 Design Principles for Modern Software Help

The episode of help seeking and the relevant theories that I have discussed provide an initial framework for understanding the nuances that underlie help-seeking behavior. As shown, no one theory completely explains the intricacies of software help-seeking; thus, I have borrowed theories from cognitive psychology, sociology, information science, education, among others to begin to form a conceptualization of the different stages that manifest in a help-seeking episode. Taking into account the strengths and weakness of existing help systems discussed in Chapter 2, I now synthesize these theoretical concepts into design principles for designing modern forms of software help. In Chapter 8, I will introduce a new approach to software help that is fundamentally driven by these design principles.

1. **Minimize the burden of switching between tasks and help:** A key phenomenon that characterizes users' behaviors with applications is the "paradox of the active user" (Carroll & Rosson, 1987) where users will tackle

their breakdowns on their own first before leaving their task to seek help. Any form of help that requires users to leave the application disconnects them from their task at hand, even if useful information is contained in that help resource. Contextual help approaches that were discussed in Chapter 2 are designed to minimize the burden of switching between a task and getting help within the application's interface.

2. **Minimize the vocabulary burden in searching for help**: Many forms of modern help systems today consist of repositories of detailed information and offer search features for users. While powerful, search based on natural language is plagued by the vocabulary problem (Furnas et al., 1987), leaving users struggling to find anything relevant using keywords. An ideal help system should allow users to find relevant help (if it exists in a repository) even if they do not know the precise keywords to accurately describe their issue. Few help systems discussed in prior work currently support this type of help retrieval.
3. **Support the social aspects of help**: The benefits of social and collaborative forms of diagnosis have long been documented (Orr, 1996; Twidale & Ruhleder, 2004). Even though one-on-one help from an expert specialist would be the ideal way of resolving any help issue, it is difficult for organizations to provide such level of individualized support. Therefore, it is useful for users to be able to communicate with or learn from another user about application functionality. As discussed in Chapter 2, there already is a surge in online community-based and crowdsourced help resources.
4. **Foster minimalism in help content:** Minimalist approaches for help content have been advocated for over three decades (Carroll et al., 1987) because early

forms of narrative-based manuals were quickly abandoned by end users. Although many help systems have adopted a minimalist approach and strive to offer concise and task-specific help content, Q&A help forums are often cluttered with back-and-forth interactions and repetitive posts. I argue that the minimalist concept should also be adopted in community-based help resources to minimize the burden of locating task-specific help for end users.

CHAPTER 4

How Modern Organizations Understand and Support Software Users: A Case Study

In this chapter, I present a case study of user-centered design at Facebook Inc. that focuses on the perceptions of software designers and product stakeholders as they build the world's largest social networking application with over a billion users[4]. This study shows that while usability and user experience insights were highly valued in the upfront design process, software designers were also increasingly turning to large-scale usage and interaction data to understand users and design software that can scale.[5]

4.1 Background and Motivation

Gould and Lewis' (1985) classical work on user-centered design in organizations in 1985 was among the first to investigate software designers' perceptions of users and design principles and to recommend key principles for designing usable software. Although modern software development practices have since evolved with the proliferation of more agile development methods (Hussain et al., 2009), we know little about methods used to understand and support users in these modern contexts. Inspired by Gould and Lewis' initial work, I investigated the context of user-centered design in a modern software development environment at *Facebook* Inc.

Facebook is the world's largest social network. It connects over a billion users worldwide and continues to grow at a phenomenal rate. Users on Facebook today come from all walks of life, live in different countries, interact in over 70 different languages, have varying levels of computer expertise, and have individual expectations when engaging with their social networks. At the time of the study, more

[4] In September 2012, Facebook's user base crossed the one billion mark.
[5] This chapter has been adapted from my CHI 2012a publication (Chilana et al., 2012a).

than half of the active users log on to Facebook every day and interact with over 900 million objects, such as pages, groups, and events[6]. Such high growth and engagement is intriguing on several levels for Human-Computer Interaction (HCI) researchers and practitioners.

In this study, I take a behind-the-scenes look at how various product stakeholders at Facebook understand users and tackle design decisions. Although previous reports from usability practitioners highlight some of the challenges of doing user-centered design in agile development environments (Chamberlain et al., 2006; Detweiler, 2007; McInerney & Maurer, 2005; Sy, 2007), my case study considers additional dimensions (Figure 4.1) that influence design and engineering at Facebook. For example, one dimension is the emphasis on creating useful and usable products that scale—this is one of the first studies to look at the challenges of designing for a billion users. Facebook is also unique because it is innovating and competing in the agile social networking space and has short design cycles. In addition, my focus is not just on upfront design, but also on how product decisions evolve after deployment and how information from users becomes useful for improving designs in the long run.

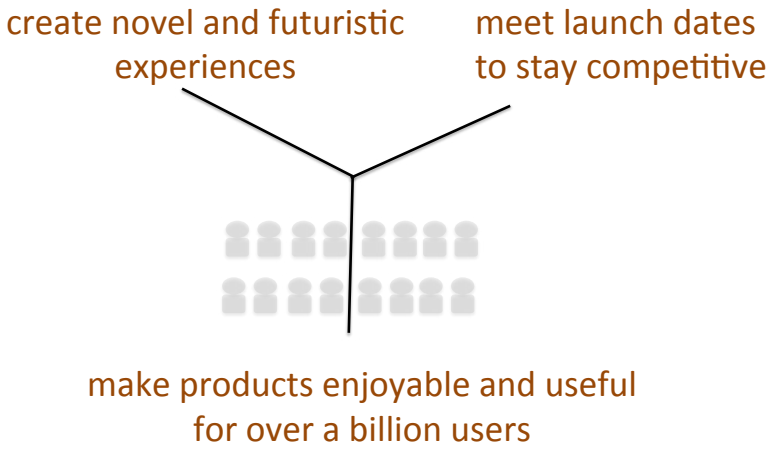


Figure 4.2: Three dimensions that influence design and engineering activities at Facebook

[6] http://investor.fb.com

These findings and analysis will be useful for HCI researchers, practitioners, and educators who are interested in optimizing methods for agile development environments and for training the next generation of UX researchers to design for a billion users.

4.2 Method

I carried out one-on-one interviews with 17 product stakeholders at Facebook working across different projects in the summer of 2011. The participants (Table 4.1) included: 5 Engineering Managers/Directors, 4 Product Managers/Directors, 3 Product Designers, 3 Software Engineers, and 2 Product Marketing Managers. On average, the stakeholders had been at Facebook for about 3 years.

Each interview lasted around 30-45 minutes. The interview questions focused on design decisions that stakeholders had to make when launching a new product or feature and how these decisions fit in with business priorities. My next set of questions focused on sources used to gather information about users and how user information was used in the design process. Lastly, I probed into stakeholders' perspectives on how users interacted with the products that they created and the role of user education and help.

P1	Product Designer
P2	Product Marketing Manager
P3	Engineering Manager
P4	Product Designer
P5	Director, Product Management
P6	Engineering Manager
P7	Director, Engineering
P8	Product Marketing Manager
P9	Software Engineer
P10	Manager, Engineering
P11	Software Engineer
P12	Manager, UI Engineering
P13	Product Designer
P14	Product Manager
P15	Software Engineer

Table 4.1: List of interview participants at Facebook

Based on perspectives that emerged in my conversations with product

stakeholders, I carried out focus groups and follow-up interviews with 6 members of the User Experience (UX) team and 6 members of the User Operations (UO) team that handles support requests from users.

All interviews were audio-recorded and transcribed. I organized, coded, and analyzed all the transcripts using a qualitative data management and analysis software. I followed an iterative process of applying *open coding* and *axial coding* to discover relationships among emerging concepts in my data, followed by *selective coding* to integrate the results (Strauss & Corbin, 1998). Through this analysis process, I continually explored different facets of the data and identified recurring themes.

4.3 Study Results

4.3.1 Designing for Users

I first asked product stakeholders to identify key decisions they had to make when designing a new product or feature. Gould and Lewis (Gould & Lewis, 1985) had found in their studies that the majority of product designers and engineers were not aware of key design principles, such as a focus on users, user testing, and iterative design. However, I found that over half of the product stakeholders in fact identified user experience as a key factor driving design decisions.

To design for user experience, most product stakeholders highlighted *minimalism* as a design approach. Designers particularly emphasized the importance of providing a clean and efficient user experience:

> *I think obviously the number one thing is being sensitive to user needs and making sure we're providing the best and most efficient, cleanest user experience possible. Cleanliness and efficiency is very important. I think the whole social aspect is very important… when I'm designing, a big thing to consider is anxiety... what the button says or where the button is placed may cause people anxiety. Just anything like that.*

Even most of the engineers who I interviewed appealed to an iterative design

approach and valued user input in creating new functionality:

> *Sometimes it's not that hard to build a particular functionality…we try to do UX and iterate on various designs versus just getting something out there, make sure it's reasonable, and then iterate on the design based on how people are using it…not just the design but the how the whole interaction flows…*

Product stakeholders in managerial roles who were involved in setting the overall product strategy and vision considered a focus on users to be a natural part of the process:

> *There [are] essentially two broad strokes that I think along…with the existing products, what are the pain points…what are the use cases that we are not capturing right now…secondly, is there a completely adjacent experience that we should be striving for [i.e., in mobile]…*

Overall, I found that product stakeholders were much more aware of key design and usability principles than what Gould and Lewis (Gould & Lewis, 1985) had found in their earlier studies. This change may not be that surprising given that user-centered design principles have been adopted in industry for over two decades now and there is formal training offered in user-centered design at all levels of education.

4.3.2 Tackling Tradeoffs in Design

Facebook and many other modern software companies today appeal to *agile* development processes (Cockburn, 2006). Unlike the traditional linear "waterfall" software development lifecycle, the agile approaches emphasize iteration, continuous feedback, and incremental mini-releases. Given the shorter release cycles, the product stakeholders in this study felt that they had to tackle a number of tradeoffs when it was time to actually make design decisions.

For example, stakeholders in managerial roles highlighted tensions between balancing deadlines and resources, and achieving desired product goals:

There's often a problem we're trying to solve or there is some opportunity…so we try to figure out if it meets our goals and is it at a quality level that's good enough for the company and for the users...there often are tradeoffs when we try to get something out the door, in terms of the number of features and the completeness of features…you know sort of the engineering versus shipping tradeoffs…biggest tradeoffs are near-term design and long-term design…

Similar to the above narrative, other product stakeholders also cited tradeoffs in designing for the "near-term" versus the "long-term." Although most software design arguably involves tradeoffs in time and resources (Gould & Lewis, 1985 ;Poltrock & Grudin, 1994), an agile environment compounds the effect of these tradeoffs:

…design is hard…really hard. Just doing our best with very smart people, we screw up plenty…design and consistency takes time…we really work to make each experience as good as it can be…design and simplicity are often after thoughts…designers propose designs, engineers go build it…then they work iteratively to improve it…at a certain point, it's just good enough and we go with it…

Product stakeholders explained that it was important to be able to tackle these tradeoffs because the company was a leader in producing innovative social networking products. Producing "novel" and "futuristic" designs that users had not seen before involved some risk, but most stakeholders felt that thinking about the long-term vision and impact rather than short-term disruptions was important for progress:

I think what I like the most about this company, we will make changes based on how people are using our product…but we are not afraid of taking risks and having a big gamble…we have a sense of where this thing is going and maybe in the short-term lose engagement or users…but even if no user is asking for this feature, we're going to go ahead and build it…

On a related note, product stakeholders pointed out that some degree of trial and

error was inherent in the design process as traditional methods of task analysis or requirements gathering were less relevant in this domain. In the absence of well-defined requirements, product stakeholders focused on the types of feelings a certain user experience may elicit:

> *…there are experiences you want to make possible and states you want people to get into and feelings you want people to feel more of or less of…and so, in my mind, a lot of the decisions that you make are pretty easy if you have a very clear sense of that at a high level. You know once you get into nitty-gritty, the decisions are much more about what you can rid off, essentially…what you can take off.*

To make engineering-level changes based on user reactions, product stakeholders were largely data-driven and cited usage numbers as being a crucial factor in optimizing design choices. Product teams often worked closely with data scientists and user experience researchers to learn about users and their usage patterns. However, when it came to gauging the reactions to a novel experience, some product stakeholders felt that it was difficult to capture the long-term impressions:

> *I would want data but it's pretty hard to always get data, particularly for this look and feel kind of stuff…it's slow right…if I make a change, it may change your impression of the product over months perhaps…not a short period of time…so it's hard to get data that corroborates that…*

Overall, despite being cognizant of user experience, product stakeholders described various complexities that surrounded decision-making about product features and design. It was interesting to see that constraints on design in terms of development lifecycles identified by Gould and Lewis (Gould & Lewis, 1985) still persisted after two decades.

4.3.3 Accommodating User Diversity

There was a sense among product stakeholders that they were striving to design a

product usable by anyone, in any part of the world. However, achieving universality with an increasingly diverse user base that was growing by the millions every week was viewed as a key challenge by the stakeholders:

> *You know we're trying to make a universal product and we think some very core pieces of Facebook are universal, so we feel like there is a solution that works for everybody for the very key things…like how you manage identity…the profile should look basically the same for everyone…but once you get farther away from the core products, it's not necessarily obvious to us that there is a magic way that a feature can work and everyone can find value in it…*

Gould & Lewis (1985) raised the issue of user diversity in their analysis and found that designers either over-estimated user diversity or under-estimated user diversity. They also found that designers and engineers often relied on their intuitions and hypotheses about users. Many of the product stakeholders in my study also cited their own intuitions and experiences as being important in pushing for certain designs; however, as explained in the following account, increasingly the stakeholders realized that their own intuitions were less relevant:

> *…when I started here the demographic and socio-economic makeup of the company was very similar to the user base…our own feelings were an excellent proxy for users…what we thought was cool, many users would agree was cool…we're now many moons from that time…when we make a new photo upload button, it needs to be equally intuitive to a 90 year Mongolian grandmother to a 14 year old Brazilian soccer player…*

To accommodate user diversity in the face of innovation and product cycles, some product stakeholders used the approach of addressing use cases for the "least common denominator:"

Given the broad sophistication of our users…with 750 million users, it's very diverse. So, often times you have to design for the least common denominator. Obviously, that's over simplifying it…often times, we reject things that we could make because the adoption rate would be so miniscule that it wouldn't be worth making. I think ease of use and thinking about broader scope of people is our priority.

Other product stakeholders felt that it was hard to discern what the least common use cases would be for all products, particularly as millions of new users were joining the network every week. New users, for example, took time in getting up to speed on even the basics of social networking:

I think understanding who can see what you share is a huge area of confusion for new users. Like the distinction between groups and posting a status update on your wall…and just understanding how to tweak to your audience is quite complex for new users. I also think just fundamentally understanding how all the different pieces fit together, the different elements of the user experience…people don't get right away.

Apart from thinking about new users, product stakeholders cited other examples of user groups that had different needs. For example, celebrities and politicians on Facebook had different expectations in setting up their fan pages compared to college friends trying to stay in touch. Also, as Facebook was expanding its platform products and allowing millions of software developers to integrate products with Facebook, stakeholders had to cater the expectations of end-users and developers alike.

Overall, product stakeholders indicated that designing for over a billion users required tough compromises and stakeholders realized that providing custom experiences to users was not an optimal solution:

It's really tough…you could imagine having different views for different users…one for like new users and one for more sophisticated users…but then you run into problems where not everyone is operating the same site…so how do we understand

it…I think it's a tough problem and then you run into problems of clutter…say if I put a question mark box next to everything [for help on the screen]...total [design] failure…

4.3.4 Learning about Users

As discussed before, most product stakeholders were aware that their own intuitions were far removed from the diverse experiences of users on the site. To more formally learn about users, product stakeholders relied on several initiatives around the company, such as UX activities, data about product use, and user feedback through support channels.

4.3.4.1 Learning From UX Insights

Product stakeholders described the UX research team as often playing a key role in helping understand "regular" users, capturing nuances of how users interacted with particular designs, and gauging user reactions before launching new products.

Product stakeholders cited usability tests as being particularly helpful for learning about different types of user behaviors and their interactions with current prototypes:

I have watched a series of live user studies where we had users in the room…we could watch how they move their mouse…it was eye-opening to see how many of your assumptions are wrong and to see all the things you take for granted because you're an engineer. You've been using computers every day all the time for 2 decades. To watch somebody who just has you know who is nowhere near there and not see a link or a button or start typing in the wrong place…and I'm not talking about the hilarious grandmother who writes an entire message in the URL bar…I'm talking about totally smart people who're can't upload a photo because they're not in the mind-set of I know there's a way to do it and I just need to find a button…we [engineers] know that there's a way to do it…but they're not sure that there's a way to do it…

In addition to usability tests, product stakeholders felt that it was valuable to get broader information about users and to get into the mindset of users outside the company:

> *I just find [UX] to be a strong resource whenever you just want to talk about something because [they] know users really well…and [have] good principles on how users would interact with this stuff…that for me is much more helpful for me than seeing [the] findings because I'm not a designer…my role is not to get [their] findings and then figure out how to incorporate them…for me getting into the frame of mind of the user is very valuable because I'm not the average user and I know that….working with that team to get that feeling is very valuable…*

Although several product stakeholders mentioned the desire to know about the "average" user of a certain product, they recognized that with over 800 million users, characterizing the average or a representative user was almost an intractable challenge. But, as advocated by Gould and Lewis (1985), many stakeholders believed that it was valuable to identify user issues even with small samples (i.e., in usability tests) than not incorporating the users' perspectives at all.

4.3.4.2 Learning From Usage and Support Data

Unlike traditional desktop software development, web application development usually consists of shorter product cycles. Increasingly, web companies are moving towards making constant improvements based on usage data and user feedback. The product stakeholders in my study also emphasized the value of understanding users based on product use and through reports from User Operations (UO), the team that deals with support requests and bugs.

Stakeholders explained that for new products, product teams had a number of hypotheses about user reactions:

It's certainly the case that debates either before or after product launches, people throw out these hypotheses and a lot of time the data just isn't there to back it up or and it would take a lot of instrumentation to do it…

Most product stakeholders in managerial roles considered data to be vital for all their decisions. Based on past experiences, stakeholders pointed out that there was a lot of variability in the long-term uptake of a product compared to short-term reactions. They felt that continuous monitoring of the usage was necessary to understand the trends and check against hypotheses:

Were we right? You know we do some small sampling of testing beforehand but once it's out the door, we have to see if the metrics hold up…some things look good initially, possibly because of the novelty effect, but once it goes out to everyone, the benefit wasn't actually there…and sometimes it's reserved…once we roll it out, the network effects will kick in…

Overall, it appeared that product stakeholders were learning a lot about product use through internal instrumentation and logging techniques. But, the instrumented flows did not always provide insight into *why* users were behaving a certain way or what could be causing particular breakdowns. To this end, product stakeholders pointed out that information from help tickets provided by the UO team was sometimes helpful. They could get a sense of where users were confused and what aspects of functionality they did not like or did not understand.

A few of the product stakeholders were skeptical of the information from the help tickets because they believed that the issues were not necessarily representative and came from a vocal minority. For example, one of the managers explained:

…you know sometimes users have a dissonance between what seems important to them versus how important something really is…so you constantly have to toll that line…there's also a skew in the reporting, there's a sampling bias…I've never seen

data on this, but I have a pretty good guess that people who are writing [to UO] are younger and skewed towards more active users…just by nature…so you have to be careful when you consider it given the long tail impact you could have…

In my focus group with the UO team, I learned that there was a lot of variation among products in terms of the reported issues that were bugs versus confusion points or user inquiries. One challenge that the UO team faced was in aggregating issues in a meaningful way without spending a lot of time in manually processing each request. Another challenge was conveying insights gleaned from the reported issues to engineers and designers:

We have more traction with engineers versus designers – because bug fixing is more tangible – designers are more like artists, they have a vision for the product… it's a challenge to approach them in the right way to convey user feedback… Engineers listen to numbers…easier to tell them something is broken in the flow than to tell them to do something differently…

4.3.5 Supporting Users Through Help and Education

Given the scale and diversity of the user base, product stakeholders agreed that help and user education were sometimes a necessary part of supporting the overall user experience.

A few of the product stakeholders believed that products could be designed to be intuitive and usable by everyone (such as an elevator) and that there was a way to get the design "right." (Gould & Lewis, 1985) also made a reference to this sentiment about design common among designers in many domains). However, other product stakeholders in my study disagreed with the idea of getting the design "right" upfront:

I mean you can't make a perfect design for 750 million people…you can't even design for one person properly because they'll tell you something and what they really want

is something else…I think user education is actually the way to go…I think most effective is proactive-in-place, tutorial-style things are the best way to go…

Some product stakeholders felt that in addition to educating users about the site's functionality, user education was also important in conveying the norms around the use of the site:

Most commonly it's about understanding the mechanisms of what drives Facebook…I think people don't understand what drives Facebook…people understand that you make friends but not much more sometimes…like people understand what the like [button] does, but they don't understand what the consequences are when they like something…

Apart from helping new users, product stakeholders in marketing roles explained that increasingly their focus was on messaging and helping all users leverage features for better managing their social networks:

I think you want people to orient to the product and understand what you're doing… privacy [is] an area that is obviously very sensitive…when it comes to privacy, it's not just teaching someone about the product, it's more about how we're positioning it…

Facebook currently offers a help center through which users can learn about products and submit inquires for help. Within the help center, users can also get help from other users in discussion forums. Most product stakeholders felt that in the spirit of social networking, users often learned best from their friends or other users. Also, through previous product launches and user education initiatives, stakeholders believed that contextual help and education initiatives benefited a large number of users and that they would continue to advocate for such initiatives:

I think it [contextual help] is very natural for users…rather than sending them to another page or a help center which is out of context…there's nothing more natural

than having a hover-over bubble telling them you can do this here…kind of the modern-kind mechanism for user education…it's minimalist, it's relevant, and it's in-context, so you can't beat that.

4.3.6 Pushing for User Experience

As shown above, product stakeholders were cognizant of user needs but had to deal with a number of tradeoffs in design. In the face of the challenges that come for designing for a billion users, it is somewhat amazing that the social network has such high growth and a highly engaged user base. In my focus group and interviews with UX team members, I learned that they had adapted their methods and approaches to work with designers and engineers to push for user experience in the face of constraints and tradeoffs.

In choosing methods and the type of data that was used to convey user information, UX team members felt that sometimes deep quantitative studies were necessary to answer some questions, but in other cases user reactions through usability testing were just as valuable for product teams. The choice of method depended on what question was being investigated:

Given the products that we have…there are different sets of users you can study and learn different things about the product. I think a lot of other industrial products have very defined and narrow audience…we have a complex product…we have an international brand…if we talk to people here vs. in Brazil, we will see different things…I think quantitative is definitely helpful for understanding user behavior at scale, but the problem with quantitative work and numbers is that you don't get the "why"…so, we have to design our questions appropriately…

User experience researchers felt that hybrid research was the best approach in tackling issues at a large scale. However, as known in the literature (McGrath, 1995), the researchers felt that this type of mixed-method research took much longer and was not

always amenable to constraints tied to agility in the development process. Still, the user experience researchers felt that hybrid research was necessary and they were planning more ongoing projects with other teams.

Regardless of the methods and the type of user data obtained, UX team members felt that a tight coupling between different product stakeholders was necessary in order to advocate changes. Also, UX team members felt that understanding and acknowledging the constraints of designers, engineers, and managers helped them to see when the injection of user data would be helpful for a given product. For example, one of the UX team members explained:

> *For a researcher to get an "in", you have to show an appreciation for the work that engineers do…because we [UX] are sometimes perceived as just critiquing and criticizing what's going on [in terms of design]…– I think it's good to give that feedback when they do something nice, like set up a test for you...we all get busy and it's not like my job, but it's good to give that feedback. I think with designers, there is need for showing even more consideration at the beginning for what they do and how hard they work…*

UX team members felt that they could make the most impact by working closely with designers, but they acknowledged that designers had a tough job in balancing their loads and negotiating with engineers:

> *Designers have more to hold in their head – engineers generally have to think about the implementation only – but designers have to think about the design and how it will work out…they can't hand off something that's going to be a pain to build, but they also have to keep the user experience in mind…I see the more we can take on that burden, they can pick up where we leave off…*

Overall, the UX team members were making adaptions and evolving their methods to fit the needs of the development process, similar to other reports of UX in agile

environments (Sy, 2007). UX team members agreed that constraints on design and engineering often made decision-making about users difficult for product teams. However, regardless of the challenges, the UX team was committed to working with product stakeholders in every way possible to better understand and illustrate the diversity of the user base.

4.4 Discussion

In this case study, I have illustrated the perceptions that product stakeholders at Facebook have about design, users, and user experience as they build and support the world's largest social network. I have highlighted various complexities that surround decision-making and the tradeoffs that have to be considered even when there is commitment towards designing for users. Although many of the findings concur with other studies of usability practices and software development (Curtis et al., 1988; Hammond et al., 1983; Poltrock & Grudin, 1994), the dimensions that are unique to Facebook are perhaps the scale of the user base and the focus on novelty and innovation. Thus, this case study raises some new issues for discussion that has recently started among HCI practitioners, researchers, and educators (Hong, 2011). In particular, the HCI community may find it useful to reconsider user research methods for agile development environments and new ways of training the next generation of user researchers to adapt to such environments.

4.4.1 The Larger Context Of The Findings

Although I mostly focused on drawing comparisons to the classical work by Gould and Lewis (1985), several other studies in the last two decades have also shed light on software design practices at various organizations (e.g., Curtis et al., 1988; Hammond et al., 1983; Poltrock & Grudin, 1994). While the domains investigated in these previous studies mostly consisted of business and office products, these studies laid an important foundation for understanding development and organizational constraints

on user-centered design. Recent surveys of usability practice and user-centered design (Gulliksen et al., 2006; Rosenbaum et al., 2000; Vredenburg et al., 2002) have shown that to tackle different organizational constraints, lightweight strategies such as requirements gathering, task analysis, and usability testing are more widely used in industry.

When considering organizations like Facebook, however, there are no "requirements" or "tasks" per say—the focus is on designing an innovative and valuable social experience for users from all walks of life. Furthermore, when dealing with a user base of close to a billion users, there is a constant struggle in optimizing the experience for the majority of users. For example, even if 99% of the users are satisfied, the remaining 1% still represent close to 10 million users. Most modern software companies do not even have a user base of 10 million users, so the impact of each design choice at Facebook can be enormous.

4.4.2 Reconsidering User Research for Agile Environments

Over the last two decades, the repertoire of user research methods has been growing and I have even seen the emergence of "discount methods" (Nielsen, 1994a) to tackle issues around organizational constraints and product cycles. However, as indicated by my findings (and experience reports by other practitioners (Chamberlain et al., 2006; Detweiler, 2007; McInerney & Maurer, 2005), innovation and product launches are often tightly coupled in agile environments. Thus, there is need to further adapt the so-called discount methods that take into account not only time and resource constraints, but also cultural factors, such as producing innovative products in a competitive landscape. The dimension of designing for a billion users may currently be a unique challenge for Facebook, but given the pace at which computing is becoming ubiquitous, it is likely that many more software products could face similar challenges of scale. I hope that through more case studies of current practices, we can better

understand and improve the adaptability of user research methods in different settings.

4.4.2 Adapting HCI Pedagogy

Along with rethinking methods, there also are pedagogical implications to consider. As in my study, user experience researchers have reported in previous studies that they have to constantly adapt their skills when working in agile environments (Chamberlain et al., 2006; Detweiler, 2007; McInerney & Maurer, 2005). They rarely apply *textbook* or prescriptive approaches to their work when faced with tight constraints. In addition, they have to adapt their communication with engineers, designers, and stakeholders and have to understand when to inject UX findings in the process and what *type* of user data to inject. Since many software and web development environments are embracing agility, getting exposed to agile development constraints in the classroom could be valuable for future practitioners. Also, there is need to help future practitioners learn to deal with the challenges of scale and shifts in diversity of the user base. Perhaps exposure to more case studies from practice can help raise awareness of the constraints in different development contexts. Finally, students may benefit from getting hands-on experience with projects that simulate different types of constraints in industry.

4.5 SUMMARY

This chapter investigated the context of user-centered design in a modern software development environment at Facebook Inc. Unlike previous studies of usability practice that have focused on upfront design methods, this case study considered the full spectrum of understanding and supporting end users. Key findings from this study indicate that software designers were much more aware of key design and usability principles than what Gould & Lewis (1985) had found in their earlier studies. However, the study also shows that tensions of working within constrained product

lifecycles still persist and software designers continue to make tradeoffs in near-term vs. long-term design. In addition, most of the study participants indicated that upfront usability was not enough and that help and user education were a necessary part of supporting the overall user experience. This study also shows that there is increased in designing software by using large-scale interaction data, for example through internal instrumentation or logging of specific events. However, a major limitation of such instrumented flows is that it does not always provide insight into *why* users were behaving a certain way or what could be causing particular breakdowns. Also, as I discuss in Chapter 5, it turns out that the majority of usability professionals are rarely involved in the post-deployment phase and do not necessarily help understand the large-scale interaction data for design purposes. In Chapter 6, I explore the role of software support specialists further and how they diagnose and manage user-reported issues after deployment.

CHAPTER 5

Understanding Post-Deployment Usability Activities

Chapter 4 presented an in-depth case study of user-centered design from the perspective of software designers and product stakeholders in a modern software development environment. Key findings from the case study indicated that due to constrained product launch cycles, software designers were increasingly interested in improving software after deployment based on usage and user feedback. However, usability and user experience research activities tended to dominate the pre-launch design process. As pointed out in the literature review in Chapter 2, currently little is known about the role of user feedback and the work of usability professionals after deployment. In this chapter, I address this gap by presenting a survey-based study of usability professionals and their role in post-deployment activities. [7]

5.1 Background and Motivation

While upfront user research and prototyping are crucial to designing user-centered products, learning from users in the deployment phase about their *actual* use of the product is also valuable (Nielsen, 1992; Norman & Draper, 1986). However, prior works (Nichols et al., 2003; Vredenburg et al., 2002) have suggested that this "ideal" UCD process across all four phases rarely happens in practice: most companies invest heavily only in upfront UCD methods. As discussed in the literature review in Chapter 2, research on usability practices has largely centered on upfront UCD methods, with little work substantiating how usability is actually practiced in the *post-deployment* phase. Given that most software companies invest a major portion of their budgets on maintaining software (Lehman & Belady, 1985) and providing technical support *after* a software has been released, understanding the UCD practices in the post-deployment

[7] This chapter has been adapted from my CHI 2011 publication (Chilana et al., 2011b)

phase is critical for identifying opportunities and challenges in supporting a product's overall usability.

To better understand the state of post-deployment usability activities in industry, I surveyed usability professionals in North America and abroad. I received 333 responses from Usability professionals working in large and small corporations representing a variety of industries. My key findings suggest that the role of usability appears to diminish in the post-deployment phase and usability professionals are rarely involved in postulated post-deployment activities (Nielsen, 1992) such as usage logs analysis, customer support logs analysis, benchmarking, and *in situ* usability testing. However, respondents also indicated that when they were involved, they found significant value in interactions with support. This study is the first to provide empirical evidence through a large-scale survey about the current state of post-deployment usability. In this regard, it complements prior surveys of usability practitioners discussed in Chapter 2 (*e.g.,* Gulliksen et al., 2006; Vredenburg et al., 2002) that have focused on upfront usability activities. My results further provide an impetus for the HCI community to better align UCD with software support and software maintenance activities that play a critical role in supporting the overall user experience.

5.2 The Survey Instrument

The survey instrument consisted of 16 multiple choice and open-ended questions. I began by asking respondents about their demographics, such as job title, experience, company size, industry, and location. Next, I asked questions about direct involvement in the 4 different phases of UCD and asked respondents to specify their particular pre-deployment and post-deployment activities. I devised categories of activities based on previous surveys (Usability Professionals Association, 2009; Vredenburg et al., 2002) and recommendations in UCD/usability lifecycle guidelines (Nielsen, 1992). I also asked respondents about their interactions with software support and software

development teams in the post-deployment phase. Lastly, I gave respondents a chance to share their stories from practice about post-deployment usability activities.

I distributed the survey online during the summer of 2010. I advertised on 15 different usability-related mailing lists and discussion and alumni lists of major HCI training programs. I also made use of social networking sites such as *LinkedIn* and *Facebook* to advertise my survey in professional discussion groups. In some cases, I personally contacted team leads in large corporations and consulting groups and asked them to encourage their employees to participate in my survey. The respondents were offered a chance to participate in a $50 gift card drawing.

5.3 Demographics of Respondents

User Experience Designer	18.3%
Interaction Designer	15.6%
User Experience Researcher	8.4%
User Experience Manager/Director	5.1%
User Researcher	4.5%
Information Architect	4.2%
Usability Professional	3.3%
Usability Engineer	3.3%
Interface Designer	3.3%
Product Designer	3.0%
User Experience Architect	2.7%
Usability Specialist	2.4%

Table 5.1: Top 12 job titles identified by survey respondents

The survey was targeted at anyone who identified as usability or a User Experience (UX) professional. My respondents listed a variety of job titles and represented various design and usability roles. Table 5.1 shows the top 12 job titles of my respondents. Most of the respondents (78.8%) were full-time employees, 17.1% were contractors and the remaining were part-time or temporary employees or self-employed.

Figure 5.1 shows the range in years of experience my respondents had in the usability field. About a quarter (25.2%) of the respondents fell in 1-3 years range while another quarter (24.3%) were in the 4-6 years range.

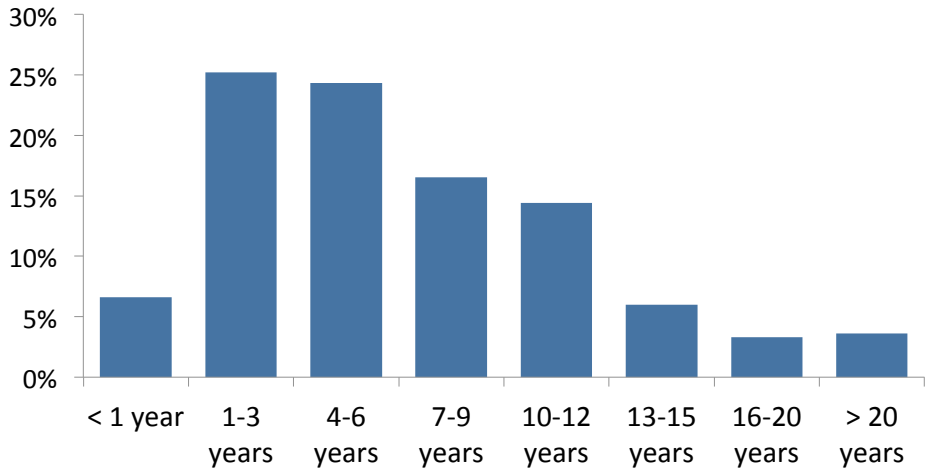


Figure 5.1: Distribution of respondent's experience

The respondents represented organizations of all sizes, with the majority being from large corporations (Table 5.2). These organization ns specialized in a variety of software, hardware, and web applications, including e-commerce sites, operating systems, online search, computer-aided design tools, social networking sites, government sites, among others. I received the largest number of responses (83.8%) from North America, followed by Asia (6.3%) and Europe (6.0%).

Large Corporation (> 1000 employees)	48.7%
Small Corporation (< 100 employees)	16.1%
Medium Corporation (100-1000 employees)	15.2%
Usability Consulting Firm	7.0%
Design Agency	4.4%
Non-profit Organization	1.9%
Educational Institution	1.6%
Self-Employed/Freelance	1.5%
Government or Military	1.3%
Startup	0.9%
Advertising Agency	0.6%

Table 5.2: Respondents' organizations

5.4 Post-Deployment Usability

I now describe responses related to survey questions that explored different facets of post-deployment usability.

One question I asked was *In which phases of development are you directly involved in some capacity?* The responses to this are shown in Figure 5.2. Among my respondents,

76.1% said that they were regularly (always or usually) involved in the user research phase. The majority of respondents (87.7%) said that they were regularly involved in the design phase. The involvement appeared to decrease in the implementation phase, but 69.2% of the respondents reported being regularly involved. Only 50.9% of respondents reported involvement in the post-deployment phase. Overall, the level of involvement of my respondents differed significantly across the 4 phases of development ($\chi^2(3, N=333)=219.9, p<.0001$).

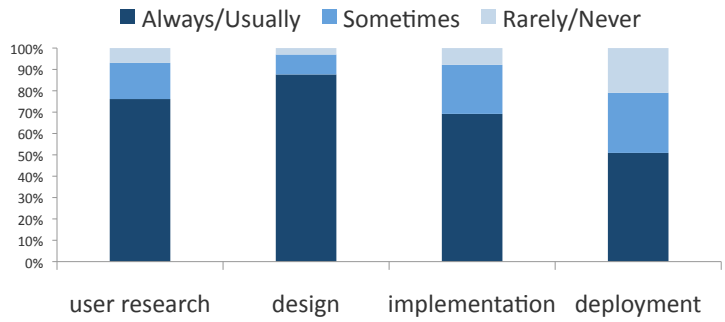


Figure 5.2: Usability involvement in different phases of development.

My next question was, *Please describe your main role after a product that you helped design has been deployed*. Responses are shown in Figure 5.3. I asked respondents to select all options that best described their role after a product had been deployed. It appears that most of the respondents (70.3%) started working on another product and/or the next version of the current product (69.1%). Only 23.1% of respondents said that they were involved in conducting benchmark tests while 33.0% said that they monitored feature data for the deployed product.

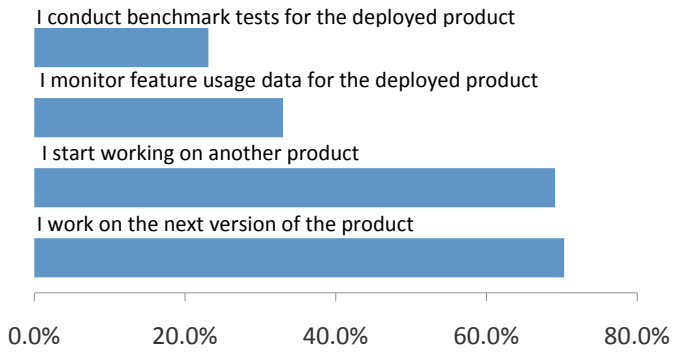


Figure 5.3: Main role of respondents after deployment.

Next, I asked respondents about the usability-specific activities that they engaged in both *before* and *after* a product was deployed. The activities and the comparison of responses are shown in Figure 5.4. Almost all usability activities appeared to drop after a product had been deployed. The only significant increase was in the use of satisfaction surveys ($\chi^2(1, N=333)=95.2$, $p<.01$). Also, note that 12.0% of the respondents selected not applicable (N/A) for the post-deployment phase.

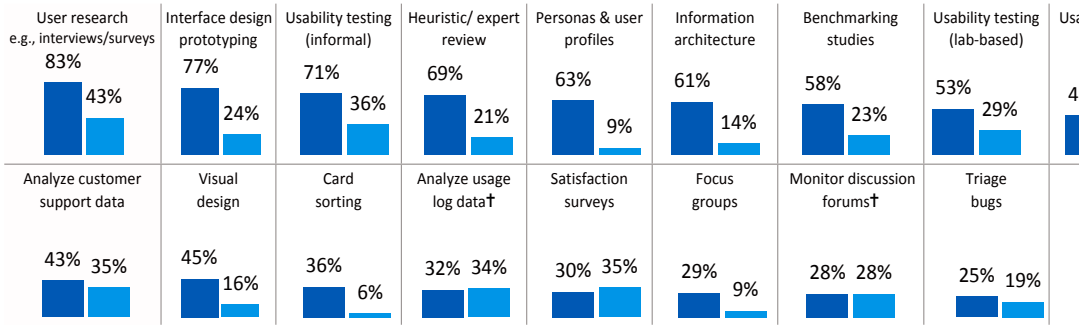


Figure 5.4: Proportion of respondents' indicating involvement across a range of activities. Dark bars represent the pre-deployment phase; lighter bars represent the post-deployment phase. All pre/post differences were significant ($p<.05$), except those indicated by (†).

My next question was, How often do you interact with support specialists (i.e., product support, customer support) after a product has been deployed? (via email, phone, or in-person meetings). Responses are shown in Figure 5.5.

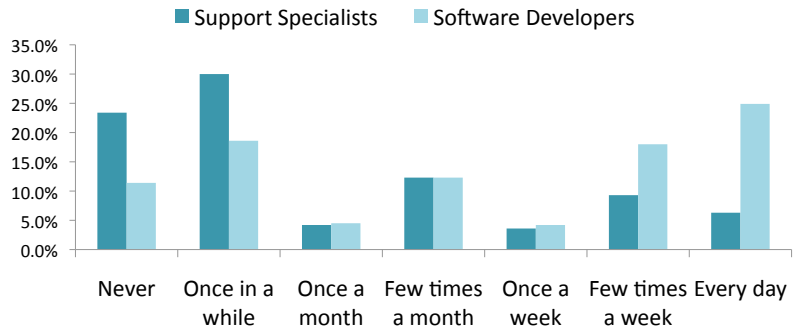


Figure 5.5: Frequency of interaction with support specialists and software developers after a product has been deployed.

Close to a quarter (23.4%) of my respondents reported never interacting with support specialists after deployment, while another 30.2 % only interacted once in a while. Another 10.6% of the respondents said that they were not sure about their level of interaction. Among the 6.3% of respondents who talked to support specialists every day, about a third (33.3%) were from small corporations. I also asked respondents to list where the support specialists were located in their organizations (Table 5.3). About half of the respondents (49.8%) said that support specialists were located on the same floor as the usability professionals or within the same building. A quarter of the respondents (23.1%) were not sure where support specialists were located.

Location	Support Specialists	Software Developers
Same floor as me	25.7%	56.5%
Same building as me	25.7%	39.0%
Another office at a different location	38.4%	34.9%
I'm not sure	23.5%	7.9%
In an office, but I work remotely	7.6%	7.6%
In the office where I consult	5.1%	5.7%

Table 5.3: Location of support specialists and software developers in the organization, relative to the respondents

The next question I asked was, *How often do you interact with software developers after a product has been deployed? (via email, phone, or in-person meetings)*. These results appear in Figure 5.5. Compared to support specialists, usability professionals appeared to have a more regular interaction with software developers after a product had been deployed ($\chi^2(1, N=333)=71.7$, $p<.0001$). Among my respondents, 24.9% said that they interacted with

developers every day and another 18.0% said they interacted a few times a week. Still, about 30.0% of respondents did not appear to have much interaction with developers (never or only once in a while). In terms of location, 56.5% of the usability professionals said that developers were located on the same floor as them and 39.0% said that developers were in the same building (Table 5.3). This difference was significant compared to the relative location of usability professionals and support specialists ($\chi^2(5,N=333)=67.8$, $p<.0001$). It appears that software developers in general were more collocated than support specialists and this could be one reason why usability professionals had more of an interaction with developers overall. However, closer examination reveals that of the respondents who talked to developers every day or few times a week, 35.7% said that developers were located in another office at another location. This finding suggests that there are factors other than location that could underlie the difference in interaction among usability professionals and software developers versus support specialists.

5.5 Perspectives on Post-Deployment Usability Activities

I also asked respondents to reflect on their role in post-deployment usability in free response questions; there was a 51.8% average response rate for the two questions. I used respondents' comments to begin to understand some of the trends that I observed in the quantitative findings. First, I filtered responses based on comments from respondents who said they were always involved in the post-deployment phase to see what type of activities they engaged in. Respondents described activities such as monitoring current usage, benchmark studies, satisfaction surveys, user testing, and having informal contact with users, consistent with Figures 5.3 and 5.4. For example, one respondent explained how she may interact with customer support after deployment:

> *I might seek them out [customer support] to find out how customers are doing with a particular product. Get their "gut feel." or I might ask to see the call reports related*

to that product. Or I might sit in a meeting with a member of product support and just get general "how things are going" info.

Still, the majority of respondents did not have such contact with customer support. Some respondents explicitly stated not being satisfied with their level and type of involvement in the post-deployment phase:

We get good feedback, but we don't work directly with customers, so it's hard to understand their specific pain points. I have a suspicion that there are minor irritations that don't ever get reported because people just don't think it's worth the effort to write to customer service. Without direct customer usability research, I don't exactly know what those are.

In the next part of the analysis, I filtered responses based on comments from respondents who said they were never or rarely involved in the post-deployment phase. One prevalent response was that usability professionals tried to be more heavily involved upfront to prevent post-deployment issues and some firmly believed that was the most important part of their job (echoing similar sentiments as some of the participants in the case study presented in Chapter 4).

Another response was that usability professionals did want to be more involved in post-deployment usability but the hindrance again came from organizational cultures and perceptions about the role of usability:

Involve us! Don't think we're just there to run a test for you-reach out to us and share your concerns for deployment and give us feedback after it's deployed.

Even when there was interest in sustaining usability throughout the lifecycle, it appears that product delivery schedules and resource constraints made it difficult to practice usability after deployment. For example, one respondent described how organizational cultures were not conducive to any substantial post-deployment activity beyond bug fixing:

Would be great to have time and budget for post-launch usability testing and user satisfaction surveys. This has never been approved in my organization. In my experience, the business owners are very eager to "wrap things up" and unwilling to invest any more time beyond fixing selected high-priority bugs.

Another challenge stemmed from organizational structures because in some cases customer support and software maintenance groups operated in silos, unaware of each other's activities. For example, one respondent indicated:

It [customer feedback] has to be channeled through a few layers of managers and generally written up as an opportunity proposal before they can give our team feedback on a launch product.

Another respondent explained the rigidity of the roles that different practitioners play in an organization:

Our support specialists are hourly and not allowed to step away from their duties to discuss with any other teams.

Beyond organizational-level constraints, several responses indicated desire for better tools for understanding and making sense of user feedback:

It works well to get reports of feedback, but it would be great if this could extend beyond the immediate post-launch period. Our support team is often unsure of who needs to see what feedback and we don't have a searchable database where I could seek this out myself.

Another theme in the responses is best summarized in the following comment:

It would be great if there were a unified way to access user responses to a new feature or release

This theme will come up again in Chapter 7 when we see that the aggregate or community impact of a software issue can affect its resolution, but that few existing tools provide a mechanism to provide such feedback about community impact.

5.6 Discussion

The survey findings presented in this chapter have several implications for post-usability research and practice, especially from the perspectives of software support and software maintenance.

5.6.1 The Software Support Perspective

The before/after findings of usability activities showed that only 34.8% of respondents appeared to leverage customer support data in the post-deployment phase, even less than the pre-deployment phase (Figure 5.4). Furthermore, over 50% of respondents never or rarely talked to support specialists. Since support specialists are at the front lines of directly interacting with end-users and helping them troubleshoot or learn about product features, it is possible that usability practitioners are missing several opportunities for learning about user experience from the field. As today's systems are becoming more complex and enabling idiosyncratic customizations, it is likely that the role of support will continue to be integral in supporting and evolving user experience. Thus, my survey findings highlight the need for the usability community to consider ways in which customer support data can be leveraged more effectively to guide iterative design tasks.

5.6.2. The Software Maintenance Perspective

The software engineering community has long recognized that software maintenance and evolution are an inevitable part of the software development life cycle (Lehman & Belady, 1985). Software developers spend most of their time triaging and fixing bugs. Only 18.9% of usability professionals said that they are directly involved in helping triage bugs, and about 30.0% of respondents never or rarely talked to software developers after a product had been deployed. Since a number of bugs that arise in the post-deployment phase are potential design and usability bugs (Nichols et al., 2003), there is opportunity for exploring how usability professionals can play a more

influential role in the bug triaging process.

5.6.3 Are Usability Professionals Really Doing User-Centered Design?

My results show that, as a whole, the role of usability in current practice appears to diminish after a product has been deployed. This finding is somewhat troubling given that iteration and user feedback have been advocated as core components of all phases in UCD and the usability engineering lifecycle (Gould & Lewis, 1985; Nielsen, 1992; Norman & Draper, 1986). Given the increased uptake of usability in industry, it is not a surprise that the value of getting upfront design into organizations has paid off.

However, despite sincere intentions in tackling potential usability problems upfront, I know from current industry practices that a number of issues emerge in the post-deployment phase (hence, the large software maintenance costs (Lehman & Belady, 1985) and software support costs). Thus, post-deployment usability may be the next frontier in translating research into practice.

I propose that within the field of usability, there needs to be focus on "usability maintenance," paralleling software maintenance. The goal of usability maintenance should be to enhance post-deployment user experience based on the actual use of a product and provide ongoing support for usability principles of learnability, efficiency, memorability, recovery from errors, and satisfaction. As discussed above, we can start further studying and inventing opportunities for support and maintenance teams to interact with usability practitioners.

5.6.4 Limitations

This survey provides the basis upon which future work can investigate industry practices in the post-deployment phase in more depth and devise new methods or guidelines. I generalize the survey findings with some caution since the survey was only distributed in English and respondents were largely from North America.

However, since my respondents represented a variety of usability-related positions and a range of organizations of different sizes and specializations, it is possible that these results would hold globally. My study currently provides an aggregate, quantitative view of usability practices, but in future work, I hope to tease out the effect of organizational cultures, usability positions, and differences between large and small corporations. It would also be interesting to complement these survey findings with observations of interactions among usability professionals and support specialists and software developers. Lastly, it is likely that in some cases pre-deployment activities on one version of a product can be considered post-deployment activities on the previous one and in future work I hope to shed light on such nuances. Together, I hope these perspectives will reveal new opportunities for usability maintenance.

5.7 Summary

This chapter presented results from a survey of 333 usability professionals about their post-deployment usability activities. The survey findings suggested that the majority of respondents started working on another product and/or the next version of the current product and had little or no role after deployment. Furthermore, due to time constraints, organizational barriers, and compartmentalized roles, usability professionals appeared to have little formal contact with support specialists after deployment. Chapter 6 sheds light on the work of support specialists and shows how these specialists are at the frontlines of handling user feedback and complaints and there is much opportunity in understanding these user experiences from a usability perspective. In Chapter 10, I will revisit this theme as I discuss findings from my deployment study with teams and discuss the potential of help analytics data captured by my LemonAid system (Chapter 8).

CHAPTER 6

How Software Teams Diagnose User-Reported Issues After Deployment

Chapters 4 and 5 highlight some of the hurdles in understanding users and achieving user-centered design in the context of modern software development. In particular, Chapter 5 shows that the role of usability professionals appears to diminish after deployment even though a number of software problems emerge in the post-deployment phase. In this chapter[8], I investigate the work of software support specialists who are at the frontlines of interacting with end users after deployment. The main focus is on understanding how support specialists diagnose and resolve user-reported issues using modern tools at Autodesk, Inc. While this chapter focuses on the work of support specialists in issue reporting and diagnosis (Figure 6.1), the perspectives of end users and the challenges that they face in describing unwanted software behaviors are discussed in Chapter 7.

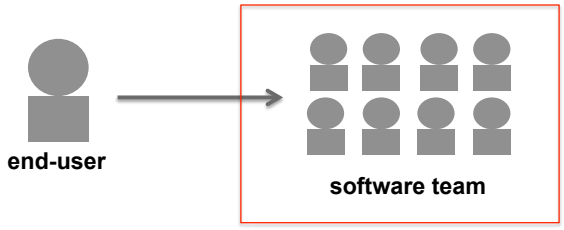


Figure 6.1: Chapter 6 focuses on the role of support specialists in diagnosing user-reported issues

6.1 Background and Motivation

Computer support or product support channels are inundated with support requests from end users every day and cost commercial software companies millions of dollars ("Technical Support Cost Ratios," 2000). Recent surveys show that calls for support, in fact, have been increasing every year[9]. There are currently over half a million support specialists in the United States alone, with a projected 14% increase in employment by

[8] [8] This chapter has been adapted from my CHI 2011 publication (Chilana et al., 2011a)
[9] http://www.supportindustry.com/2009supportmetrics.html

2018[10].

In its earlier days, software support was typically offered over the phone, but the trend in support today is to use the web (Negash et al., 2003). (Please see chapter 2 for a more detailed overview of the current state of support practices and issue reporting). Users and support specialists can now leverage new channels for communicating and troubleshooting, such as using screen sharing tools or attaching images and videos to requests and responses.

What is surprising is that even though support specialists are at the frontlines of interacting with software users, and are an important resource for providing end user assistance, few HCI studies have directly investigated support workflows and the challenges that support specialists face in resolving user issues remotely. Furthermore, although media attachments in modern support systems have the potential to facilitate a type of conversational grounding (Fussell et al., 2000) not possible in phone-based support, little is known about how these formats fit into support workflows and what utility they offer in helping specialists resolve issues.

I address these gaps in existing research by investigating the product support practices at AUTODESK, Inc, a large, globally distributed software company that serves more than 80 products to over 10 million customers worldwide, and has nearly 200 product support specialists. I used a mixed-method approach for my study: an analysis of AUTODESK's internal archive of support requests, a company-wide survey of product support specialists, and follow-up one-on-one interviews with support specialists.

The main contribution of this work is in illustrating how modern web-based support is practiced and the relevance of multimedia formats in resolving support issues. In addition, I identify four main bottlenecks that product support specialists face despite the potential availability of information through visual channels. Based on my findings, I

[10] http://www.bls.gov/oco/ocos268.htm

discuss several opportunities for better facilitating the exchange of support-related information over the web. My analysis will be useful for developers seeking to improve the design of support tools, support personnel and managers interested in learning from support practices in the field, and HCI researchers studying user-reported software issues.

6.2 Research Site and Method

I now describe my research site and the strategies used to collect data for understanding product support activities.

6.2.1 About Autodesk

AUTODESK, Inc. is a leading software company specializing in 2D and 3D design, engineering and entertainment software for the manufacturing, building and construction, and media and entertainment domains. There are over 80 different AUTODESK products currently in the market, serving over 10 million end users in 185 countries.

6.2.2 Method Overview

Since AUTODESK offers a range of products and product support activities at AUTODESK are spread across different channels, I used a mixed-method approach for collecting data. I began with a pilot investigation where I conducted unstructured interviews with two product support specialists. My goal in these initial interviews was to get an overview of the product support process, who the relevant staff members were and where they were located, and learn about the tools used by customers and the support specialists. To learn more about internal support-specific tools, I also interviewed one of the lead developers of AUTODESK's internal customer support database. Through this process, I obtained access to an archival database containing the complete set of customer requests from the last 8 years.

Following this pilot investigation, I designed 3 strategies to systematically gain an in-depth understanding about the product support process: (1) an analysis of existing

support requests, (2) a company-wide survey distributed to 200 product support specialists, and (3) follow-up interviews with a subset of product support specialists.

6.2.2.1 Quantitative Analysis Of Archived Support Requests

The internal customer support database contained over 4 million records from various customer-related service activities, including technical support requests. I obtained access to the internal archival SQL server database, and wrote my own C# programs to run queries to explore these requests in more detail.

Through my analysis I found that there were 1, 665, 970 service requests in the database submitted between January 1, 2002 – May 15, 2010. Out of these, 381,060 were business requests, while 1,284,910 were product-related technical support requests. I first randomly read 20 technical support reports to gain an understanding of the report structure and report-specific activities. Next, to investigate some trends that I observed in this sample, I carried out quantitative analyses over the last one year of data which contained around 75,000 support requests.

6.2.2.2 Company-Wide Survey

The second method that I used was a company-wide survey of product support specialists. My goal was to better understand the demographics of product support specialists and the prevalence of some trends that arose in my initial analysis of support requests. The questionnaire consisted of 30 questions that elicited a mix of multiple-choice and short open-ended responses. I focused my questions on the demographics of support specialists and their use of tools in communicating with customers and diagnosing software issues. Finally, I asked respondents if they wanted to opt-in for a follow-up interview. The survey took about 10-15 minutes to complete.

I distributed the survey online using the company's intranet and by advertising on internal mailing lists. All responses were collected anonymously. I received 72 responses from a pool of 198 product support specialists at AUTODESK, giving us a response rate of

36.4%.

6.2.2.3 Semi-Structured Interviews

Lastly, to confirm trends in my quantitative findings and to learn more about experiences of support specialists, I conducted 16 semi-structured interviews. These interviews were carried out with product support specialists working across different AUTODESK sites. Each interview lasted approximately 45 minutes. I carried out 11 of the interviews in person, 4 over the phone, and 1 using the company's internal telepresence conferencing facility. My 16 interviewees included 7 product support specialists, 3 senior product support specialists, 5 product support team leads, and 1 product support technical lead. They had different specializations across 10 of the major AUTODESK products and 2 of the interviewees provided support for all products. The support-related experience of the interviewees ranged between 2 to 15 years, with an average of 5.9 years.

In the first part of the interviews, I focused on confirming some of the trends that emerged in my analysis of support requests and the company-wide survey. Next, I used the critical incident technique (Flanagan, 1954) to probe into scenarios that were particularly challenging in providing support. Since I conducted the majority of interviewees within the product support's specialist work environment and they had access to the repository of support requests, I found that it noticeably facilitated recall among the interviewees.

I audiotaped and transcribed all of the interviews. All transcripts were organized, coded, and analyzed using the NVivo data analysis software. In the first pass, I coded for data related to multimedia formats mentioned by my interviewees. In the next pass, I examined critical incidents described by interviewees using an inductive analysis approach (Strauss & Corbin, 1998). This approach of was useful in exploring different facets of the incidents described by support specialists and in identifying recurring themes.

6.2.2.4 Presentation of Results

Since my three data collection strategies produced a large amount of quantitative and qualitative data, I have combined my presentation of the results in terms of the major themes that emerged. I begin in the next section with an overview of product support at AUTODESK.

6.3 Overview of Product Support

I first provide a global overview of the process of product support provision and demographics about product support specialists.

6.3.1 Reporting and Tracking Tools

All support requests were logged in a web-based tracking system (even those reported via the phone initially). The web-based system allowed customers to update information and attach files during the course of the request. Customers could also track the progress of all their requests through this system. The required fields when submitting a request were the product name, operating system (OS), a one-sentence summary, and the full problem description.

Product support specialists accessed support requests via another web-based system where all the history and attachments related to individual requests were stored. The specialists could send information to the customers, add updates, and run queries through this interface as well.

Finally, every report submitted since 2002 was archived in a consolidated internal database. This database could be use by support specialists to search for existing solutions or to find similar issues. Developers and QA specialists also archived change requests or bugs in this database.

6.3.2 Reporting Process

When a customer submitted a request, it was handled by tier 1 specialists who had basic knowledge of AUTODESK products, but were not specialized in any specific

support area (Figure 6.2). They could search for known issues in the consolidated database and provide help, or they could escalate the issue to a product support specialist. From that point forward, the product support specialist and the customer communicated with each other directly. The support specialists who I studied were at the second tier and had extensive system-level and application-level expertise. When necessary, the support specialists would collaborate with other team members, developers, or QA specialists to diagnose and resolve complex issues.

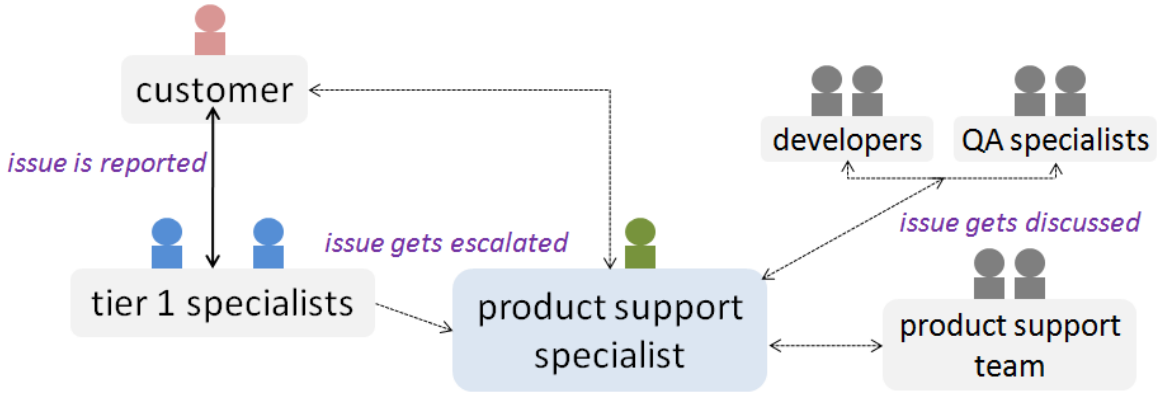


Figure 6.2: The issue reporting and handling process.

6.3.3 Demographics of Product Support Specialists

There are around 200 product support specialists at AUTODESK who specialize in different products and areas of support. Among my 72 survey respondents, 44 had the title product support specialist, 10 were senior product support specialists, 13 were product support team leads, 2 were product support directors, and 4 had other titles such as customer support engineer. The median experience of the respondents in the field of product support was 8.8 years. The majority of my respondents worked on specific AUTODESK product families, and only 11% across all products, mostly providing support for installation issues. In terms of training, 41% of respondents had Bachelor's degrees, 36% had Diplomas or Associates Degrees, 12% had Masters degrees, and 8% had Certificates in a range of fields such as computer science, civil engineering, animation and design, among others. My respondents estimated that they

addressed between 2 and 30 support requests every day, with an average of 5 each.

6.4 Analysis of support requests

I carried out various analyses to understand the structure and contents of support requests and computed statistics about activities surrounding these requests.

6.4.1 Overview of Support Requests

I initially looked at a random sample of 20 existing support requests to see the structure of the information contained in these requests. The main information contained in these records of support requests included:

- the dates when the request was created and modified
- the request status (i.e., whether it was closed or open)
- the issue's summary and description
- the product and feature affected by the issue
- the issue's internal classification

Based on my informal screening of these support requests, I learned that some reports lasted a few hours whereas some lasted a few months. There was also variability in terms of the topics of the reports and the number of iterations in the conversations between the customers and support specialists. Some requests were about system configuration issues, such as installation. Others were related to application feature issues. Support specialists were frequently asking for more information from customers, and there were numerous files formats being sent back and forth.

To investigate these support request trends more systematically, I carried out a quantitative analysis over the last one year of support requests that had been closed.

6.4.2 Trends In Support Requests

The results of my quantitative analysis are summarized in Table 6.1. I found that the

median life-time of support requests, calculated as the time between when they were submitted to the time they were officially closed was about 2.9 days. The median number of activities (messages and/or attachment submissions) per request was 6, while the average was 12, so there appeared to be a high amount of activity in the support requests within their short life-times.

Total number of support requests	74,837
Life-time of support requests (until marked as closed)	2.9 days (median)
Number of activities in support requests (messages and attachments exchanged)	6 (median)

Table 6.1: Summary of one year of support requests.

The distribution of the internal classification of support request areas is illustrated in Table 6.2. It is interesting to note the high proportion of *how-to's* in support requests, suggesting that customers consult support not only when something goes wrong, but also when they want to figure out how to accomplish a particular task. Some of my interviewees who worked on feature issues indicated that the number of how-to's was as high as 75%.

Since the web was used as the main platform for support at AUTODESK, there was opportunity for exchanging information in formats not possible through phone-based support. Next, I look at the file formats exchanged between customers and product support specialists in more detail.

6.4.3 File Formats In Support Requests

In the one year sample of requests, I found that 23,514 requests (31.4%) contained attachments, but only 12370 requests (16.5%) came in with an attachment at the time of submission. I further investigated these attachment types and found that they included a range of file formats, such as images (jpg, png), source files (AUTODESK-specific drawing files), archive files (zip, rar), documents (doc, docx, pdf), text files, log files, and videos (avi, swf). Figure 6.3 shows the combined distribution of attachment types

with initial requests.

Area	% of Support Requests
How-to	26%
Incorrect Result	26%
Error message	15%
Crash/Hang	9%
Configuration	7%
Limitation/Wish	6%
Performance	3%
Documentation	3%

Table 6.2: Distribution of top 8 support areas

To better understand the actual use and utility of these different multimedia formats, I asked format-specific questions in my survey and had follow-up questions in my interviews. I considered most of the formats that were found in on my analysis: textual descriptions, screenshots, source files, and video screen captures. I also considered screen sharing which I noticed was also being used in my random sampling of requests.

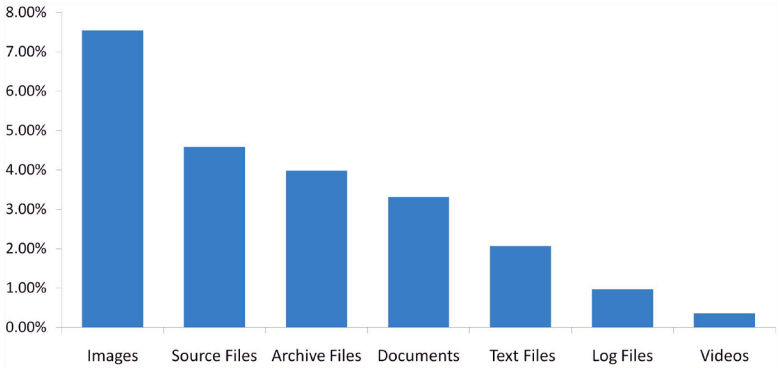


Figure 6.3: Percentage of support requests that initially came in with the various attachment types, in the one-year sample data.

I asked respondents to rank: the usefulness of formats for seeing information related to a customer's support request (Figure 6.4a); how likely it was for customers to

include information in these different formats with their initial support request (Figure 6.4b); and, how likely it was for them to request information in these 5 different formats after seeing a customer's initial request (Figure 6.4c).

The data illustrated in Figures 6.4(a)-(c) points to a perplexing discrepancy in terms of the formats that the product support specialists said were useful for diagnosis, versus what they reported customers included in their request, and what support specialists *actually* asked customers to submit. For example, consider that 89% of respondents said that video screen captures were useful or very useful, but only 27% of specialists were likely to ask customers to submit videos. To understand this discrepancy and why product support specialists made such choices, I asked my interviewees to comment on the merits and challenges of using the different formats.

6.4.3.1 Screenshots

My interviewees explained that screenshots were the most useful format usually because screenshots provided a more accurate visual sense of what was actually happening on the customer's screen, compared to the customer's paraphrased description. One interviewee commented:

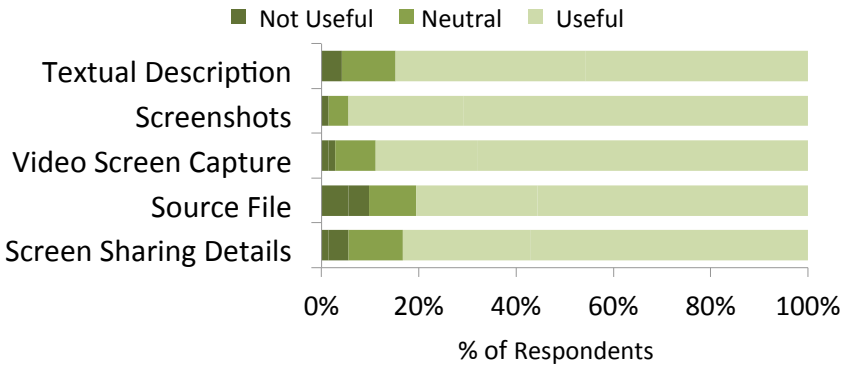


Figure 6.4(a): Usefulness of formats for understanding issues from the perspective of support specialists. In general, any additional information was considered useful.

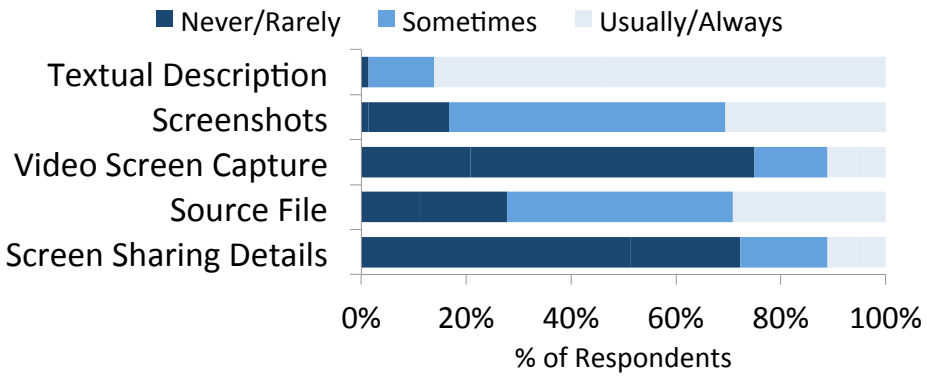


Figure 6.4(b): The likelihood of customers submitting additional information with their initial requests from the perspective of support specialists

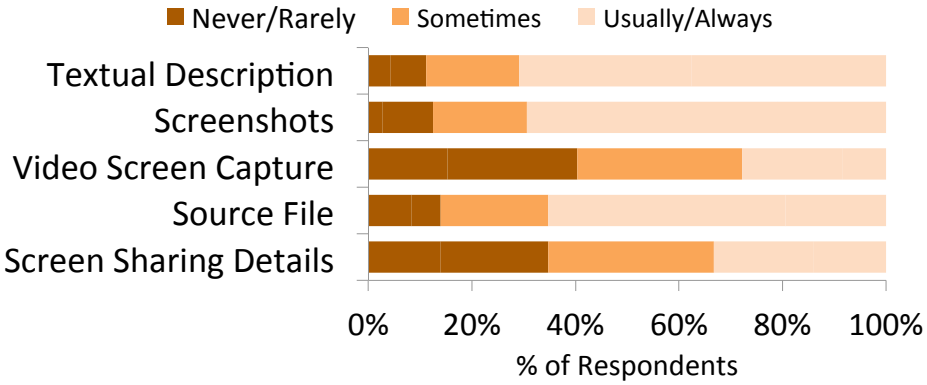


Figure 6.4(c): The likelihood of support specialists requesting information in different formats from customers after the initial requests were submitted.

We ask for screenshots for error messages. Our error messages are cryptic and numerical. If the error has something like 000bc and you type in 000dc, that's a significant difference for us. Instead of the additional havoc we have to deal with, we just ask users to submit us screenshots of the exact error message. (P16)

The interviewees generally agreed that often asking for a screenshot of the problem was easier since most OS today offer a built-in "print screen" function. Still, one surprising finding was that four of my interviewees (25%) described instances where customers did not understand the concept of a screenshot. One interviewee remarked:

We had a customer [who], when we asked for a screenshot, got out the company's digital camera and took a picture of the screen and then emailed that picture to us. (P04)

In addition, interviewees pointed out that although screenshots were a lot more useful than text alone, the screenshots alone did not always provide enough information for troubleshooting complex feature issues:

[The] problem with screen capture images is…[they don't] tell us the workflow...unless they've edited the image or drawn arrows or clouded something, often times, they're not even pointing at the issue itself. I may have a whole screen image of a bunch of stuff, but it may not be obvious where the issue is. (P12)

6.4.3.2 Video Screen Captures

Video screen captures were considered useful for understanding complex or multiple steps needed to reproduce a problem or for explaining things to customers when screenshots were not sufficient. For example, one of my interviewees explained

Videos are useful in specific circumstance. If you can't reproduce the problem or get the customer to describe their problem, ask for a video or get them to demonstrate it live in WebEx. Those circumstances are rare - more specific to workflow issues. (P01)

Video captures were also useful for capturing missing steps during the diagnosis process of a complex issue:

I told him [the customer]: I did this, here are 2 videos, this is the result I get, do you get the same thing? We ended up with 7 videos for each step, trying to exactly isolate where the problem was. He sent one video back just to make one thing clear. (P10)

One specific question I had was why despite being so useful were videos not requested from customers as much as screenshots. Interviewees explained that the main reason was they did not want their customers to struggle with installing 3rd party software for creating videos:

According to my interviewees, another issue with videos was that they consumed more time compared to screenshots:

> *It's also easier for customers to provide screenshots than videos. Videos are more complex. Not all customers have time. I've asked customers for videos before but they just tell us they're too busy to do it. Then what are we going to do. (P01)*

6.4.3.3 Source Files

My interviewees explained that most of the tasks that users accomplished with AUTODESK software products involved some form of drawings consisting of intricate processes. Thus, when customers were trying to explain what they did or what is wrong, the product support specialists benefited from seeing the actual file in question, so they could have the ability to reproduce the issue on their end:

> *We do end up asking for drawing files, so we can try to reproduce them on our system and test to see if the issue is with their[the csotmer's] network and to try to recreate the error with their system. (P01)*

Another benefit of source files was that the specialists could see what the cause of a problem was, for example, for an error or crash situation:

> *At least 50% of the time I need the file or script. We get files because customers says, 'I want to do this' or 'I'm rendering and it's crashing'. At this point, a video won't be helpful because yes it'll show me the problem, but won't help me diagnose it. Then I need the file so I can reproduce the problem and break down the steps to reproduce and [see] the related steps. (P03)*

One issue with sharing files was that they were often very large in size due to the complex drawings that they contained. Special FTP sites had to be set up in many instances for customers to upload large files, which users often had difficulty accessing. In addition, some interviewees felt that files could introduce information overload. Complex files contained "too much" information and the customers mostly failed in pinpointing the

exact location of the problem:

> *I had a guy tell me the other day he couldn't select a line. The drawing he sent me had over 2000 objects in it, so I'm like ok that's great, but which line are you having a problem with? (P04)*

6.4.3.4 Screen Sharing

My interviewees had mixed feelings about screen sharing: some said they live by screen sharing, while others said that they would only resort to screen sharing when other possibilities had been exhausted.

The obvious advantage of sharing screens was that it maximized the potential for "shared understanding" (Fussell et al., 2000) for the situation at hand. Product support specialists who worked on system configuration issues particularly found this shared understanding to be useful because of the number of system variables involved. But, since modern screen sharing software allowed support specialists to even take control of the customer's systems, other benefits could also be gleaned as described in this account:

> *Without the [screen sharing], it would be a nightmare. Our stuff is pretty complicated and even some of our advanced users just don't get it sometimes. We depend on [operating system] components for setting up servers and some people just want us to do it for liability issues if they are not confident, they think it will get messed up. So with [screen sharing] we can share files, we can connect desktops, it's like live meetings, remotely access the system. (P02)*

Other interviewees described the downsides to screen sharing that they had experienced with customers. Some interviewees felt that the screen sharing sessions ended up taking too much of their time in cases where they needed to investigate the underlying issue further. For example, one interviewee pointed out screen sharing put him "on the spot" when he was dealing with complex features:

So, when I get an email case, I can sit down with it, try couple of things. It may take me an hour. But, if I connect right away, I just wasted my hour and the customer's time. I didn't make myself look good because I appear like I'm failing in trying to diagnose. Whereas if I get their file, I can look at it and tell them this is what your problem was and this is how you fix it after an hour. (P11)

In summary, I found that the use of multimedia formats did help support specialists in establishing common ground (Clark & Brennan, 1991) with customers, but customers were not always aware of *how* to make use of these formats and did not always know *which* format would best describe a problem.

6.5 Bottlenecks In Issue Resolution

Despite the use of multimedia attachments, my findings shed light on a number of challenges that product support specialists faced in resolving support issues. These challenges were synthesized from the *critical incidents* that I elicited in the interviews. In this section, I discuss the four major themes that appeared within these critical incidents and group them as *bottlenecks* in issue resolution.

6.5.1 Unclear Problem Descriptions And Steps To Reproduce

One bottleneck that came up repeatedly in the interviews was that customers usually provided descriptions that were either too vague, brief, or general and not specific enough to the context of the issue. Furthermore, the descriptions often lacked clear steps to reproduce.

For example, one issue was the mismatch between the actual problem and what the customer *believed* was the problem in the initial description:

[The customer] said I'm running this script but my filter is disappearing. [He asked] Is there something wrong with my scripting command? So he wrote it up and sends me the script which is 100 lines long. He said this should describe my problem. I spent a lot of days figuring out what he had done, I spoke to QA and we looked at it

together. We weren't sure if the script that he had provided was a clear example of what he was describing...So, I got him on the phone and talked to him and tried to figure it out. And that's when he goes to say, oh actually that was just an example of 'this', but I really want to do 'that'. The 'that' was a lot more complex. (P03)

As discussed in the previous section on formats, even when customers did try to be helpful and sent in additional information such as screenshots or source files, this information was not always helpful in establishing an understanding of the problem if it lacked an accompanying explanation of the relevant context.

I had one case this morning, customer basically had a view where they basically have this kind of nice round hand rail in their program and when they go to put in another view of it, it just completely vanishes for no good reason. So, I asked him for his file. Initially, he just sent me a screenshot and we don't always get all the information we need from that. The problem is that it doesn't show me his problem. It could be happening because of many things, like he could've selected the wrong view. (P13)

This type of problem was also evident in my survey, which showed that 77% of the support specialists would find it somewhat useful or extremely useful to see additional tags or annotations on images or videos sent by customers to better understand the context of the issue.

The support specialists also felt that customers did not see the value of reproducibility information in the descriptions they provided. For the types of complex software at AUTODESK, often leaving out even a little step had a great impact on the diagnosis and the resolution of the issue. One product support team lead explained a case that his team had worked on extended for a month just because the customer kept missing one step in the reproducibility information:

There's one that's been going on for a while that I wasn't able to reproduce …the case has been open for a month..first it was in tier 1 hands, and it finally got escalated to

me because they couldn't reproduce it. I tried to find out if it was reported before. I asked them to uninstall the hotfix and they replied that as soon as they put the hotfix on, nothing worked. I did that too and couldn't reproduce it. So I asked them [customer] for the exact file that they had used..I verified it that I had the same dll file. When I tested it again in a new environment with this file and followed their steps, I was able to reproduce it. I may have missed out a step earlier because yesterday morning they [customers] clarified the steps again, but it took a while to get us there. (P09)

One factor influencing unclear descriptions and incomplete reproducibility steps could be that the current interface which the customers use to submit requests does not explicitly ask them to provide steps to reproduce. But, as other studies have shown, even in interfaces where the requirement to submit the steps to reproduce information is explicitly stated (i.e., in OSS communities (Bettenburg et al., 2008; Singh et al., 2006), reporters rarely follow these instructions. I discuss possible ways of mitigating this problem in my discussion section.

6.5.2 Variability In System Environments

Another bottleneck that affected the resolution of support issues was variability within customers' system environments. Customers were reported to be using a wide range of OS, software, and hardware configurations, with any combination of these potentially being at the root cause of an issue. Thus, even in the rare case when a customer succeeded in providing a clear description of the problem and the steps to reproduce, sometimes support specialists were stalled in diagnosing the customer's issue because they could not replicate the underlying system environment.

For example, one interviewee pointed out that it could be something as simple as an automated OS update that could cause incompatibly with an application's features:

There's another one [case] that I just closed today. This one had a ton of iterations. It's been running for over 3 months. It's not even solved, but the customer's last response was just close it and I'll look at it in the fall. He sent me his files and they worked fine for me. We can see the files ok, using the same OS, and same version of the application. I sent him screenshots, videos, and screenshare, done it all…there are things like an [OS update] that break things. There are 2 different versions of [XY software] and there's nothing in common. Same software. But, routinely just an [OS update] may break it. But you never really know what did it. (P14)

One product support specialist who worked on an application that was dependent on the system environment described an extreme case in diagnosing an error that the customer was experiencing. He had to physically obtain the customer's system in order to do further troubleshooting:

…a lot of times the error comes from customization. Our program allows them [customers] to make things really customized so we end up getting their whole system, restore it, and then try to reproduce it. (P02)

A related problem identified by support specialists was that customers lacked knowledge about their system environment, concurring with other findings (Jiang et al., 2009). For example, customers often did not know the version of the software they were using, details about their OS, graphics card, memory, driver information, among other system configuration settings. Furthermore, there were also application-specific settings which could cause incompatibilities. In my discussion section I discuss possible opportunities that could assist users in reporting their context at the system and application levels.

6.5.3 Privacy Or Security Concerns

Another class of bottlenecks that emerged in the critical incidents described by interviewees had to do with privacy or security concerns in sharing data. Strict

lockdowns in the customer's work environment prevented them from providing relevant information needed for diagnosis:

> *A couple of private firms wouldn't send me the data so the SR [support request] kind of died. What can I do. I've had a few calls, I send them a WebEx link and their firewall is blocking it, so it's kind of frustrating. I can try to supplement it with a video...but we ran into [issues with] that too. A lot of workplaces block Flash, so I send them a link to a swf file (video) and they can't play it. Sometimes customers take that video home with them and then play it from there. (P07)*

Some customers required specialized *non-disclosure agreements* (NDA) in order to share files. But, such cases often ended up entangled in legal issues and stalled the diagnosis of the issue:

> *They [customers] indicated that they were unable to send the model because they needed a non-disclosure agreement (NDA) first. I retrieved an NDA at first but they refused it because they didn't want to sign a two way NDA and said that it had to be a one way [NDA]. So, we had to go through our legal department and they indicated that they didn't do one-way NDA agreements. About a month later, they [legal team] figured out we could do one way. I sent that to the customer—they then sent us their NDA with modifications, then we sent it back. That's basically where it's been. (P05)*

Since privacy issues sometimes blocked the progress in a support request completely, product support specialists had come up with workarounds in helping customers self-diagnose their issues:

> *I've been making suggestions along the way for the customers. I've even pointed out that new release has come out [which could help]. None of them has provided a resolution, so it's a situation that I haven't heard anything from the legal dept or the customer in terms of where things stand…essentially to isolate a problem in our modeling software, it's really hard, even if there was some way to make the model*

private...we need to know the parameters, whole geometry, in order to isolate the problem. If the problem is happening in an area considered private or secret, then we can't really deal with that. (P05)

Some support specialists used the strategy of asking the customer to extract a limited portion of the file that showed their issue or to send in as much as information possible. Other support specialists disagreed because the extent of the privacy or security concerns was so great that an excerpt of a file was not helpful:

Getting the customer to edit out or do something to hide a part just wouldn't help. I mean their issues are either black and white - they can send the data or not. Hiding the name of the building that they are creating doesn't make it ok to send it out. (P04)

Although privacy issues related to files are entangled in larger organizational and social issues, I point out some possibilities in my discussion section about how to enable the exchange of sensitive information more securely.

6.5.4 Variability In User Workflows

The final bottleneck I discuss, that emerged in my data, was related to support specialists struggling to understand the specialized domain or application-specific workflows being used by customers. For instance, many support specialists felt that the root cause of a customer's issue was sometimes in the workflow of what the customer was trying to accomplish, rather than a defect with the application:

It's half workflow issue, like they [customers] are confused...it's a relatively complex thing. It's working as it should for the most part, but the way they've done it is just more complicated. We have to get them started from scratch and tell them to go through these steps and tell them why it's doing it. (P 10)

Other interviewees said that customers often assumed the support specialists would be familiar with their idiosyncratic workflows:

When I ask a customer to tell me how to reproduce the issue, he'll be like just draw a wall. And they don't say click here, here and there. I had a case where we kept going back and forth and I couldn't reproduce the issue. It turned out that the customer kept leaving out a step. They [customers] are architects and engineers and we're the people writing the steps so there's kind of a disconnect. (P13)

Half of my interviewees pointed out that regardless of their knowledge of an application, sometimes the lack of familiarity with a customer's domain also impeded the diagnosis process:

When [YZ software] was first released, it was supposed to be for palette design. Now, we've seen everything from a prison facility with a fence that went all around it, to a fire-escape for sky scrapers, a floating barge that loaded and unloaded oil rigs in sea...and I'm like you know what...this is a little beyond me. (P15)

This interviewee further commented that he could just walk across the hall and talk to someone in such situations. Other support specialists worked around by calling friends in industry who could sometimes help them understand latest industry-related issues relevant to a support request. Collaboration appeared to play a role in dealing with a lack of domain knowledge, as has been illustrated in other studies (Chilana et al., 2009). Most support specialists noted that online discussion forums were becoming popular for discussing workflow-related issues and the specialists even encouraged customers to consult these forums:

Largely because there's more than one way of doing a lot of this stuff. The community can share best practices. Because honestly we're not users of the product, so I don't know the best way to design a cul-de-sac, but someone out there probably does. (P11)

My interviewees felt that they could already see community-based discussions changing the current state and future of one-on-one product support.

6.6 Discussion

My results illustrate that in remote resolution of software issues, common ground (Clark & Brennan, 1991) between a user and a support specialist can be enhanced by the use of multimedia formats. However, users do not appear to always use appropriate mediums for describing the features of a problem or fail to adequately describe the sequence of steps desired by support specialists. Although these findings of unclear or incomplete user descriptions are consistent with previous studies of troubleshooting (Bettenburg et al., 2008; Crabtree et al., 2006; Singh et al., 2006; (Twidale & Ruhleder, 2004), my study shows that commercial web-based support practices are plagued with additional concerns, such as privacy control in exchanging sensitive files. Also, with increased push for end user tailoring and customizability of applications (Harrison, 1995; Mørch, 1997) the disadvantage for support specialists is that they have to deal with unique scenarios of feature usage, more variability in steps used to accomplish tasks, and occurrence of more problems due to different combinations of application features tried by users.

I now discuss the implications of my findings for improving the resolution of software issues through web-based support.

6.6.1 Opportunities For Improving Issue Resolution

I consider three levels of opportunities for improving remote issue resolution: (1) the integration of reporting formats and applications; (2) the design of automatic tools for capturing the user's system and application level context; and (3) the integration of community-based support channels.

6.6.1.1 Integration of reporting formats and applications

Although a vast number of freeware tools online today could be useful for diagnosis purposes (i.e., *Jing* for quick video captures), my results showed that they were rarely used. The support specialists felt that the main problem which hindered users in

making use of external multimedia tools was the lack of integration with the application state. Users were either not aware or were hesitant to download freeware that they did not know how to use only for the purpose of creating a support request. Thus, one way of mitigating this concern would be to allow users to generate screenshots or videos within the reporting interface related to an application. Recent tools such as the *Windows Problem Steps Recorder* could be exploited for this purpose, although at the time of this study, I did not find widespread usage of this tool in the support requests submitted to AUTODESK.

In addition, as discussed in my findings, support specialists often found it difficult to make sense of a user's problem from raw multimedia attachments. For complex issues, they needed to see a pointer to the issue that the user was experiencing whether in a screenshot, a video, or a source file. A potential solution is to provide seamless ways for users to add annotations or highlights to images, videos, and documents that are attached to a request.

6.6.1.2 Automatic Capture Tools

As illustrated in my findings, when support issues are intricate and involve several variables, users struggle in looking up and providing the necessary system or application-state level information. My results point to two opportunities for relieving the burden on the end user to describe these levels of context: (1) automatic capture of information about the user's system environment, and (2) automatic capture of application-level sequence of interactions.

System-level: Automatic capture and analysis of system logs has been discussed in prior work (Bettenburg et al., 2008; Jiang et al., 2009) but my findings suggest that even basic system level information about the OS, version numbers, installed packages, and permissions can reduce iterations between end users and support specialists. Simple ways of capturing the relevant environment information are needed along with facilities for

attaching this information automatically at the time a request is created so that the burden on the user decreases and the support specialist does not have to repeat the same request to all users.

Application-level: The other bottleneck that support specialists faced was dealing with descriptions where users struggled in describing their own actions and history of application-level interactions. I believe there are opportunities in this space to explore how document histories can be captured automatically so that when users submit support requests about feature issues, their steps to reproduce get automatically attached. For example, the Chronicle system (Grossman, Tovi et al., 2010) which captures and visualizes a document's workflow history, could be adapted to be used in the context of issue reporting.

Although efforts in automatic capture of system-level details and application-level details could alleviate some of the burden that users have in describing in their context, there are some inherent challenges in doing automatic capture that are implied in my findings. For example, a major concern that emerged in my findings was preserving the privacy of the user's information. To address this, users could be provided with regulated tools to control what portion of their system logs get included in an automatic capture, or what portion of a user's file gets recorded or submitted. Users could adjust the levels of sensitivity from project-to-project and modify the type and extent of the automatic capture of context. Another potential tension surrounding automatic captures is information overload from the captured data. I believe it is imperative for support specialists to be included in the design of such tools upfront so that they can provide a perspective on support workflows and pinpoint to data that would be most useful for diagnosing different types of issues.

6.6.1.3 Community-Based Support Channels

Finally, my results suggest that given the variability in user workflows and

environments and the high cost of one-on-one support, community-based forums will emerge as the next frontier in modern support. Online discussion boards that facilitate the exchange of troubleshooting experiences have existed within OSS communities for over a decade (Lakhani & Von Hippel, 2003; Singh et al., 2006), but their uptake in commercial contexts is also on the rise (Nambisan & Baron, 2007; Wiertz & de Ruyter, 2007). Such online user communities could be particularly useful for users having domain-specific workflow problems where they could benefit by learning about best practices in the community from other users. The opportunities that I have discussed in the previous sections, such as integration of reporting tools and privacy control in exchanging sensitive information, should be considered for community-based help initiatives as well.

6.6.2 Limitations

Finally, I acknowledge my study has a number of limitations. First, although the support specialists and the support requests that I investigated came from 80 different products at AUTODESK, my results could potentially be biased because of organization or corporate culture. In addition, some of the findings could be a direct result of the specifics of the support tool being used, rather than the overall process itself (although my investigations showed that the particular support management system used at AUTODESK is used by over 6000 companies globally). The theme of workflow variability could be unique to my study, since I were dealing with software for design work, which is known to be creative, with ill-defined problems (Wolf et al., 2006). There are also limitations associated with the methods that I used. For example, an inherent limitation of semi-structured interviews is that no two interviews end up being the same. Thus, I generalize the findings with some caution. Still, I believe that since I took a multi-dimensional approach in studying this topic and the research site was a large company, it is possible that my results reliably illustrate the different facets

of modern product support.

6.7 Summary

In this chapter, I presented results from a multi-dimensional analysis of product support activities at a leading design software company where the web was used as the main platform for interacting with end users. I carried out a quantitative analysis of existing support requests, a survey with product support specialists, and follow-up interviews to understand current practices in support. This study's results illustrate the advantages of different formats that get exchanged in web-based support requests, but also point to larger bottlenecks faced by support specialists in diagnosing and resolving users' issues. The LemonAid help system that I present in Chapter 8 is a potential approach to alleviating the burden that support teams face in diagnosing issues that lack context because users can ask questions and report problems within the context of the application. Also, LemonAid allows users to engage with each other in the support process and allows for the solutions to be retrieved by anyone in the future, potentially reducing the need for repetitive one-on-one support sessions. In Chapter 7, I consider issue reporting from the perspective of end users as they describe unwanted software behaviors in web-based discussions.

CHAPTER 7

How Users Report Software Problems in Web-Based Discussions

As discussed in Chapter 6, a major bottleneck that software support specialists face in diagnosing user-reported software issues is that end users provide vague, brief, or unclear descriptions, often lacking the necessary context for the issue being reported. Studies of bug reporting practices where users directly report software issues to developers online have also revealed the same lack of clarity in issue report descriptions (Bettenburg et al., 2008). In Chapter 7, I explore this problem further and investigate the qualitative aspect of how end users describe unwanted software behaviors in web-based bug report discussions and how users' descriptions impact the final resolution of the bugs by software developers (Figure 7.1). [11]

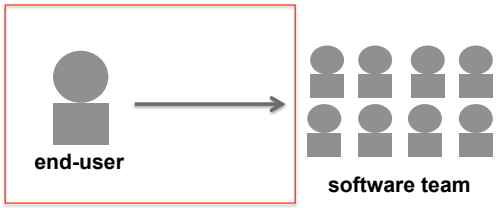


Figure 7.1: Chapter 7 focuses on the role of end users in software issue reporting

7.1 Background and Motivation

Popular Open Source Software (OSS) projects such as Mozilla are inundated with hundreds of bug reports every day from end-users around the world. Although bug reports allow users to inform developers of the problems encountered while using a software. Bug reports typically contain a detailed description of a failure and occasionally hint at the location of the fault in the code (in form of patches or stack traces). However, bug reports vary in their quality of content; they often provide inadequate or incorrect information. It is no surprise that developers are slowed down

[11] This chapter has been adapted from my VL-HCC'10 publication (Chilana et al., 2010)

by poorly written bug reports because identifying the problem from such reports takes more time

One way that the OSS community copes with this daily wave of issues is to separate them roughly into two categories: (1) problems that violate developers' intents, and (2) everything else, including feature requests, help requests, issues out of a team's control, among others (Antoniol et al., 2008; Weiss et al., 2007).

Of course, most end-users know little about developers' intents: they simply know that an application did something unwanted and often users report that unwanted behavior as a bug. But what is an "unwanted" behavior from the user's perspective? And how are users notions of "unwanted" software behaviors related to developers' intents? And how does this clash between users' expectations and developers' intents affect which reported bugs are addressed? I investigated these questions, complementing recent studies of bug reporting from developers' perspectives (Anvik et al., 2006; Bettenburg et al., 2008; Glerum et al., 2009).

My approach was to randomly sample bug reports from Mozilla's Bugzilla repository, analyzing unwanted behaviors described in the report titles and descriptions. I found that in describing unwanted behaviors, reporters implicitly referred to one or more common classes of expectations that had been violated, including the reporter's personal experiences or the practices of the user community at large. From this initial analysis, I extracted a classification scheme of seven common expectation violations and applied it to 1,000 Mozilla bug reports. Using these classifications, I analyzed the relationship between the expectations identified in each report and whether the report was eventually resolved as fixed. My key findings reveal that, at least in the Mozilla project, whether a bug is fixed depends largely on whether the reporter explicitly refers to the developers' intents or to the expectations of the Mozilla user community. All other forms of expectations receive little attention.

This work contributes: (1) a conceptualization of unwanted behaviors described in bug reports as a violation of expectations, (2) a classification scheme for capturing the different types of expectation violations, (3) an analysis of expectation sources from a large sample of Mozilla reports, and (4) empirical findings that show the relationship between the expectation source identified in a bug report and the report's final resolution. I conclude by discussing the implications of my classification scheme on open bug reporting and bug reporting tools, highlighting some ways that OSS communities can better leverage contributions from end-users.

7.2 Method

To study and classify unwanted behaviors described in bug reports, I gathered data from the Bugzilla repository of the Mozilla project. I chose to study the Mozilla project because of its large user base and its reputation as a user-centered open source project. I downloaded all available Mozilla bug reports, 496,766 in total, on August 14, 2009 using standard HTTP queries. Since I was interested in whether or not a bug was eventually fixed, I did not include any bug reports that were still open. I focused on bugs that had been reproduced and decided upon by selecting bugs marked as CLOSED, RESOLVED, or VERIFIED in Bugzilla. This filtering criteria resulted in 420,005 reports. I wrote Perl scripts for my initial exploration of the bug report data and for computing some variables of interest.

I next describe the method that I used to classify unwanted behaviors in bug descriptions and my application of the resulting classification scheme onto a sample of 1000 reports.

Bug 413312 - Autocomplete dropmarker styling is different if site is secure Last Comment

Status: RESOLVED FIXED
Whiteboard:
Keywords: regression

Product: Firefox (show info)
Component: Location Bar (show info)
Version: Trunk
Platform: x86 Windows XP

Importance: -- normal (vote)
Target Milestone: Firefox 3 beta3
Assigned To: Dão Gottwald [:dao]

Reported: 2008-01-21 00:55 PST by Steve England [:stevee]
Modified: 2008-02-06 20:32 PST (History)
CC List: 5 users (show)

See Also:

Crash Signature:

Attachments		
dropdown styling as I hover over it (on secure site) (2.13 KB, image/png) 2008-01-21 00:55 PST, Steve England [:stevee]	*no flags*	Details
patch (935 bytes, patch) 2008-01-21 11:29 PST, Dão Gottwald [:dao]	gavin.sharp: review+ mtschrep: approval1.9+	Details | Diff | Review
Add an attachment (proposed patch, testcase, etc.)		View All

Steve England [:stevee] 2008-01-21 00:55:41 PST **Description**

Created attachment 298228 [details]
dropdown styling as I hover over it (on secure site)

Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3pre) Gecko/2008012004 Minefield/3.0b3pre ID:2008012004

~~Bug 397331~~ changed the auto-complete drop-down styling to look like attachment 297257 [details]. For me I see this styling on normal sites, but on secure sites when the location bar background colour changes to yellow, hovering the dropdown marker just draws a box around it rather than styles it to look like a normal widget.

I think I would expect it to look as per attachment 297257 [details].

(I don't know if this matters, but I use classic theme in WinXP)

Dão Gottwald [:dao] 2008-01-21 11:29:19 PST **Comment 1**

Created attachment 298309 [details] [diff] [review]
patch

Dão Gottwald [:dao] 2008-01-21 11:31:20 PST **Comment 2**

(In reply to comment #0)
> (I don't know if this matters, but I use classic theme in WinXP)

It does matter, because classic themes don't provide a 'toolbox' appearance.

...

Reed Loden [:reed] 2008-01-23 23:07:48 PST **Comment 5**

Checking in browser/themes/winstripe/browser/browser.css;
/cvsroot/mozilla/browser/themes/winstripe/browser/browser.css,v <-- browser.css
new revision: 1.152; previous revision: 1.151
done

Figure 7.2: A Sample Bug Report from Mozilla's repository

7.2.1 Classification of Unwanted Behaviors

I first selected a sample of 50 bug reports and analyzed unwanted behaviors described in the report's titles and descriptions. Figure 7.2 shows an example of a bug report from Mozilla's repository. Through this analysis, I found that users implicitly referred to different expectations that had been violated as they described unwanted behaviors. I decided that the source of expectation a reporter believed was violated was a potentially interesting and important variable. To operationalize source of expectation, I employed an inductive analysis approach (Glaser, 1992), classifying and reclassifying my descriptions of the different sources of expectations. The first author independently examined 3 sets of randomly selected bug reports (100 reports each), generating descriptions of what was being violated in the bug report titles and descriptions. These descriptions converged on a single coding scheme after numerous iterations and discussions with the other authors.

My classification of the different sources of expectations consisted of seven codes. The first three codes represent more conventional notions of a bug as a violation of developer intent:

1. **Runtime logic.** Explicit violations of some runtime expectation, including errors, warnings, assertion violations, crashes, and hangs (e.g., "…scary deadlock assertions exiting mozilla after referencing nsInstallTrigger…").
2. **Specification**. An agreed upon functional requirement among the developers (e.g., "There's an incorrectly placed PR_MAX in the code for pref width distribution of colspanning cells.").
3. **Standards**. Specifications shared by the industry in which the application is deployed (e.g., "'codebase' attribute of the HTML 4.0 OBJECT element is not supported…").

The remaining four categories refer to other sources of expectations, outside the scope of the implementation, developer community, or industry:

4. **Reporter expectations**. A reporter's personal perspective about what the system should do (e.g., "Every time I Sort By Name by Bookmarks Firefox sorts and closes my Bookmark menu... Why does it do this??").
5. **Community expectations**. A reporter's belief about a "typical" user's expectations, including specific references to user, users, user interface, or usability. (e.g., "The preference to not show the tab bar when only one tab is open could be set to false by default. This would at least alert a new user to the possibility that tabs exist) The old tabbed browsing preferences could be returned.").
6. **Genre conventions**. References to applications with similar functionality; allusions to how a specific feature behaves for the same action. (e.g., "Firefox does not limit the slideshow horizontal size to the window width. The same source works correctly in IE.").
7. **Prior behavior**: References to the prior desirable behavior of the system (e.g., "The latest version of Firefox only imports one certificate from each file. I used to import all certificates previously.").

While these categories may not be exhaustive, they did capture the full range of expectation violations found in my sample.

7.2.2 Sampling and Analysis

To test my classification, I next selected a uniform random sample of 1,000 bug reports from my data set, excluding those used to develop the classification. The first author applied the coding scheme described above to my sample. To assess the reliability of the coding scheme, the second author coded a subset of 100 reports. For this subset,

there was a 78% agreement on issue types between the two coders (κ=0.62). Finally, note that a small portion of bugs in my sample (n=25) did not fit my coding scheme because they described meta-issues about the bug reporting process; these were excluded from my analyses.

Next, I explored the association between source of expectation and bug resolution. Upon my initial analysis, I observed that 30.07% were marked DUPLICATE. I then further analyzed the resolution of these DUPLICATE bugs to determine their final resolution flags. For simplicity, I marked the bugs that were fixed as DUPLICATE_FIXED and grouped all other resolution flags as DUPLICATE_NOTFIXED. Table 7.1 shows the distribution of bug resolution flags in my sample, and a brief description of each resolution status.[12]

FIXED fixed by a check-in	40.00%
DUPLICATE_FIXED duplicate of another bug and was fixed	16.40%
DUPLICATE_NOTFIXED duplicate of another bug and was not fixed	13.70%
WORKSFORME cannot be reproduced	13.40%
INVALID observed behavior is the intended behavior	9.80%
WONTFIX valid but cannot be fixed	3.50%
EXPIRED	1.80%

[12] Source of Mozilla-specific bug resolution definitions: https://developer.mozilla.org/en/What_to_do_and_what_not_to_do_in_Bugzilla

expired after a period of inactivity	
INCOMPLETE steps to reproduce are not complete	1.40%

Table 7.1: Distribution of Resolution Flags in My Sample

7.3 Results

I now report my main findings: (1) the distribution resulting after applying my coding scheme to a sample of 1,000 bugs and (2) the correlation between the *source of expectation* and *bug resolution*.

Figure 7.3: Distribution of sources of expectations

Figure 7.3 shows the distribution of sources of expectations violated in my sample of 1,000 bugs. Clearly, the largest portion of my bugs in my sample were reporter expectations, which referred to violations of reporter's personal expectations (n=337). Violations of runtime logic (n=195) and specification (n=177) were the next largest groups. The remaining groups each accounted for less than 10% of the sample and were distributed as follows: community expectations (n=85), genre conventions (n=71), prior behavior (n=69), and standards (n=41).

To assess how the source of expectation affected bug resolution, I performed

multinomial regression with *source of expectation* as a nominal predictor and bug resolution as a nominal outcome. I found that the *source of expectation* had a significant effect on bug resolution ($\chi^2(7, N=1000) = 35.8, p<.001$). Figure 7.4 shows the relationship between the *source of expectation* and *bug resolution* categories. As shown, over half of the bugs that were about violations of **specification** and **community expectations** were FIXED. Although **reporter expectations** occupied the largest proportion in my sample distribution (Figure 7.3), only about 20% of these reports were first resolved as FIXED—about half of these bugs were initially marked as DUPLICATE and only about 20% of the duplicate bugs were eventually FIXED. Bugs about **standards**, **genre conventions**, and **prior behavior** were more likely to get marked INVALID, meaning that the developers considered the actual behavior to be the intended behavior and not violations.

Furthermore, the distribution of FIXED bugs shows that when users identified unwanted behaviors that were violations of **specification**, **community expectations**, or **runtime logic**, they were more successful in getting their bugs resolved as FIXED. On the other hand, when users cited personal experiences only, or conventions in competing systems, they achieved little success.

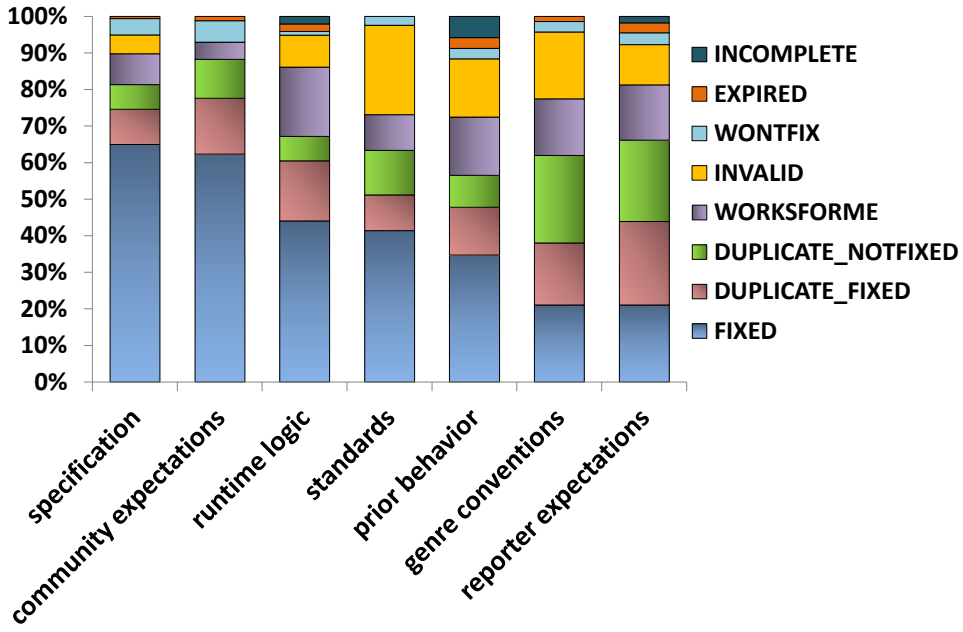


Figure 7.4: Distribution of resolution categories for sources of expectations.

7.4 Discussion

This study contributes a detailed articulation of the unwanted behaviors that users describe in bug reports. My analysis shows that there is a correlation between a user's expression of whose expectation is being violated and whether or not the bug will be fixed. My current analysis did not take into account possible confounds that could affect bug resolution, which I plan to include in future work. It would be particularly interesting to examine other factors that influence reporters to describe bugs as violations of their personal expectations. For example, reporters who have not yet diagnosed their problems may just tend to report non-issues and tend to explain things in personal terms instead of community terms. Although, my results are limited to Mozilla, below I discuss implications of my classification scheme on understanding end-user bug reporting behavior and augmenting the design of bug reporting tools.

7.4.1 Expanding the Notion of a Bug

First, my findings reveal the limitations of simple divisions between unintended behavior and feature requests, expanding the notion of a bug to a wide variety of

sources of user expectations. If I look at the bugs in my sample from the perspective of binary classifications (i.e., Antoniol et al., 2008), real "bugs" (the specification and runtime logic violations in my scheme) only accounted for about 37.2% of my sample. By this measure, over 60% of the other bug reports were simply "non-bugs." But through my classification, I learned that these "non-bugs" encompassed a range of unwanted behaviors that violated the users' idiosyncratic personal expectations, exposure to previous versions of a system or use of other similar systems. The developers were in fact responsive to fixing many of these "non-bugs" provided that they were expressed as violations of the user community's expectations. (These findings also contrast the extant belief (cf. Dalle et al., 2008; Lakhani & Wolf, 2005) that OSS developers tend to focus only on issues relevant to errors in code or functionality problems.)

My results open an intriguing question: can users "game" open bug reporting by articulating a problem in community terms? Or do developers eventually uncover the reality of an issue? It appears that what users write and what the real issue is are two dimensions of a bug report. Future studies should investigate how an issue's phrasing really affects the outcome of a report. My results suggest that the answer to this question will depend on the source of expectation. For example, it appeared that many users had a difficult time accurately interpreting the meaning of HTML specifications, which led to several invalid reports. However, in the case of reporter expectations, there may be many common, critical usability issues behind individual issue descriptions that are never discovered, simply because of how they are phrased.

7.4.2 Implications for Issue Reporting Tools

With the current design of open bug reporting tools, it is likely that end-users will continue to submit a large number of isolated idiosyncratic descriptions of unwanted behaviors, most of which will not get fixed. But if 10,000 such idiosyncratic

descriptions were to point to the same issue, how could we redesign bug reporting tools so that the community impact of such an issue is more obvious and the chances of that issue being resolved are increased?

Current focus on enhancing bug tracking tools has been on improving the quality of the bug report (Bettenburg et al., 2008; Just et al., 2008), and the information exchange between end-users and developers (Breu et al., 2010). But these improvements largely benefit developers. To better leverage user participation in open bug reporting, my results suggest that bug reporting tools should provide the user with: (1) more concrete ways to express a range of unwanted behaviors, and (2) some form of feedback about the extent to which the reported issue also affects the larger user community. For example, if tools could automatically identify violations of personal expectations in bug report descriptions, users could learn up front that their bugs are not likely to get fixed. This feedback would perhaps encourage users to refine their reports or investigate other ways of resolving their individual issue.

Also, if end-users have more concrete ways of expressing unwanted behaviors, and bug reporting tools can be designed to aggregate these in a meaningful way, OSS developers could have a rich view of community impact and be able to make more informed software evolution decisions.

7.5 Summary

Open bug reports serve as a forum for users to communicate with developers and express a range of unwanted software behaviors, as seen by my classification scheme. My results illustrate how articulation of community impact can allow users to have more success in getting problems resolved. Although these findings suggest that software teams consider community impact to be an important consideration for resolving an issue, few issue-reporting tools facilitate the capture of such data. In Chapter 10, I discuss an extension that I built to my LemonAid system that allows end

users to indicate "me too" on questions or problems reported by other users (preventing additional idiosyncratic descriptions of the same issue). These question-specific votes and views on questions are aggregated and presented in an analytics dashboard to provide data on the top issues affecting users.

CHAPTER 8

LemonAid: Selection-Based Crowdsourced Contextual Help Retrieval

In the preceding two chapters, I discussed software issue reporting and diagnosis from the perspectives of software teams (Chapter 6) and end users (Chapter 7). These studies highlight a key problem in web-based software issue reporting: users' descriptions are often incomplete and imprecise and lack the context necessary for fully understanding the issue. Even if a user is reporting an issue that has already been previously reported, the user's idiosyncratic description can make it difficult to locate the existing issue and its resolution. This chapter introduces LemonAid, a new approach to software help and issue reporting that allows end users to retrieve questions and answers provided by other users or software teams by simply making selections within an application's interface.[13]

8.1 Background and Motivation

In Chapter 3, I derived four main design principles for modern software help based on theories and empirical studies of software help seeking. These principles included: 1) minimize the burden of switching between tasks and help; 2) minimize the vocabulary burden in searching for help; 3) support the social aspects of help; and 4) foster minimalism in help content.

In chapter 2, I summarized the idea of *crowdsourced contextual help* that has recently been explored by some researchers and practitioners where the goal has been to integrate social forms of help within the context within the application's user interface (UI). For example, the *CHIC* framework (Stevens & Wiedenhöfer, 2006) for Eclipse adds links from each UI control to a wiki where users can author help. *TurboTax* help ("TurboTax

[13] This chapter has been adapted from my CHI 2012 publication (Chilana et al., 2012b)

Support," 2013) and *IP-QAT* (Matejka et al., 2011) display help discussions relevant to the current view in a sidebar within the application. In terms of the design principles stated above, I argue that these forms of crowdsourced contextual help systems work well in supporting principle #1 in that the help is displayed within the application's interface; they also support principle #3 in that they support social aspects of help; and also support principle #4 in that the help content is presented concisely in the form of questions and answers.

Despite the power offered by these crowdsourced contextual help systems, the key problem with retrieving help from forums and user-generated discussions where end users are expected to search for relevant questions and answers is two-fold: 1) users' queries tend to be incomplete and imprecise (Belkin, 2000); and 2) queries are plagued with the vocabulary problem (Furnas et al., 1987), where different users provide different words to describe the same goal (i.e., "add a photo" *vs.* "insert an image"). Therefore, these existing forms of crowdsourced contextual help systems do not fully address design principle #2 focused on minimizing the vocabulary burden.

This dissertation introduces *LemonAid*, a new approach to technical help retrieval that allows users to ask for help by selecting a label, widget, link, image or other user interface (UI) element, rather than choosing keywords. With LemonAid, help is integrated directly into the UI (as in 8.1) and users can ask questions, provide answers, and search for help without ever leaving their application. The key insight that makes LemonAid work, one supported by my formative studies, is that users tend to make similar selections in the interface for similar help needs and different selections for different help needs. This tight coupling of user needs to UI elements is central to LemonAid's effectiveness, reducing unnecessary variation in users' queries (as will be discussed in the retrieval evaluation in Chapter 9).

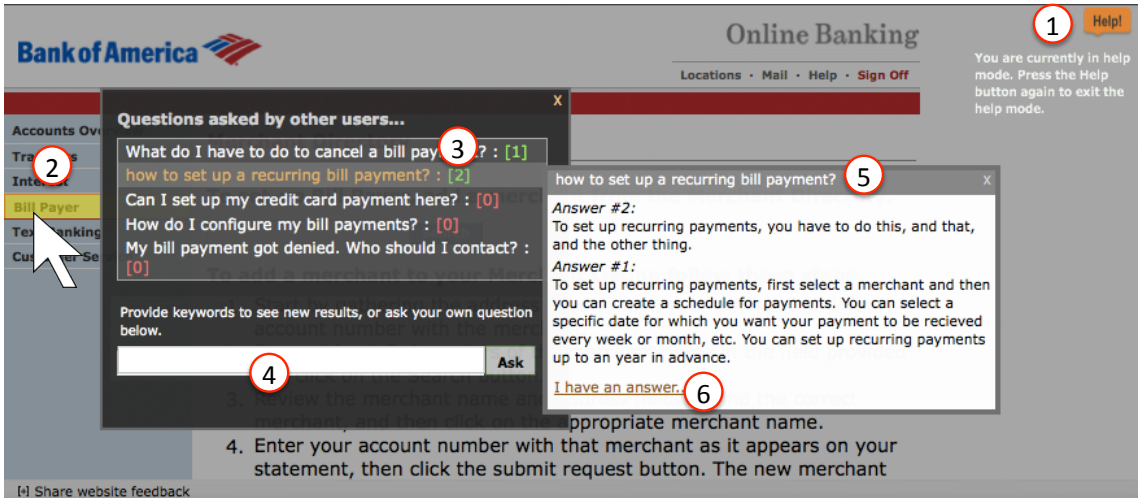


Figure 8.1. The LemonAid user interface: (1) the help button, which invokes the help mode; (2) a user's selection; and (3) questions relevant to the user's selection (the brackets contain the number of answers available for each question). (4) A search box where users can provide keywords to filter the results or ask a new question if they do not see a relevant question. (5) Answers linked to the selected question and (6) a link that allows a user to submit an answer for the selected question.

LemonAid works exclusively with standard DOMs, makes no assumptions about how an application's back or front end is implemented, does not require the modification of application source code, and does not require the use of a specific UI toolkit. It simply operates as a layer above an application's UI. The only work that developers must do to integrate LemonAid into their site is extract text labels appearing in the UI from a web application's code and include the framework in their client-side deployment. I have tested the LemonAid framework with a few web applications and found that the integration work is minimal.

While LemonAid's approach is influenced by recent work on helping programmers debug code in-context (*e.g.*, (Brandt et al., 2010; Hartmann et al., 2010; Ko & Myers, 2004)), the novelty lies in providing a contextual help framework for *end-users* to resolve issues through crowdsourcing. My larger vision is that software teams will be able to use this repository of contextually reported issues for better understanding potential usability problems and user impact. This chapter contributes:

- A new selection-based querying interface and a question and answer authoring

interface allows users to generate help content and find help without leaving the application.

- A new help retrieval algorithm that leverages contextual information and user interface selections to retrieve relevant help content.
- A framework that allows web developers to easily integrate LemonAid within a web application.

8.2 The Design of LemonAid

LemonAid allows users to find application help in-context by selecting user interface (UI) elements, including labels, widgets, links, and images that they believe are relevant to their help request, question, or problem. For example, consider 8.1, which shows an example scenario in which Bob, a teacher who occasionally does online banking, wants to pay his electricity bill by setting up a monthly online payment through his bank. Bob has checked his account balance a few times using the bank's online application, but he is not familiar with all of the other application features. He clicks on the "Bill Payer" option and it opens up a new page with more information, but now Bob does not know what to do. Normally, Bob would call his tech-savvy friend for help, but since it is late, Bob needs to find a solution on his own.

Bob clicks on the LemonAid "Help" button at the upper right of the page (8.1.1) and the screen dims, indicating that the application has entered help mode. LemonAid fades the user interface, indicating to Bob that the meaning of a "click" has changed. As Bob moves his mouse cursor over the page, he notices that words and objects under his cursor are highlighted in yellow, indicating that they are clickable. Bob selects the "Bill Payer" label, as he thinks it is most relevant to his problem (8.1.2). LemonAid displays five questions that it believes are relevant to his selection (8.1.3), all which have been asked by other users who had previously selected the same or similar labels. Bob immediately notices the 2nd question, "how to set up recurring payments," and sees it has 2 answers

(indicated by the number in brackets). Bob clicks on the question and sees that the first answer is what he needs (8.1.5).

While he is logged in, Bob also wants to update his phone number in the e-banking system. He again goes into LemonAid's help mode and this time, he clicks on a tab labeled "Account Profile." The help system now shows a much longer list of relevant questions (not shown), as there are many features relevant to this label. Bob does not want to read all of them, so he starts typing into the search box (8.1.4) and notices that the question list updates. Bob now only sees two questions (not shown), the first of which is explicitly about adding phone numbers.

8.3 LemonAid design and architecture

The main component of LemonAid is a retrieval engine that produces a ranked list of relevant questions in response to a user's selection on a page. Selections are captured as a DOM element and other context (described next) and then matched against the existing set of questions stored in LemonAid's repository of application-specific questions and answers. In this section, I (1) describe the contextual data that LemonAid captures, (2) explain how LemonAid represents questions, answers, and user selections, and (3) explain LemonAid's retrieval algorithm.

8.3.1 Capture of Contextual Data

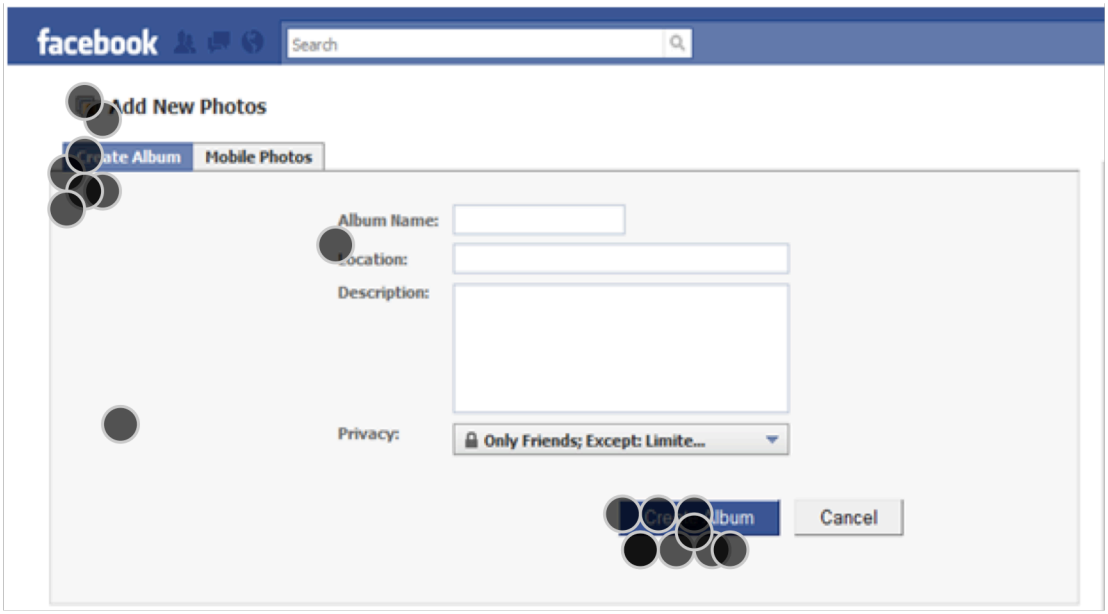
When a user makes a selection, LemonAid captures information about the target DOM object and underlying HTML, which I will call *contextual data*. There was a variety of contextual information that LemonAid could gather, so I first performed a formative study to identify what context was useful in discriminating between different help problems in an application.

8.3.1.1 Formative Study Setup and Design

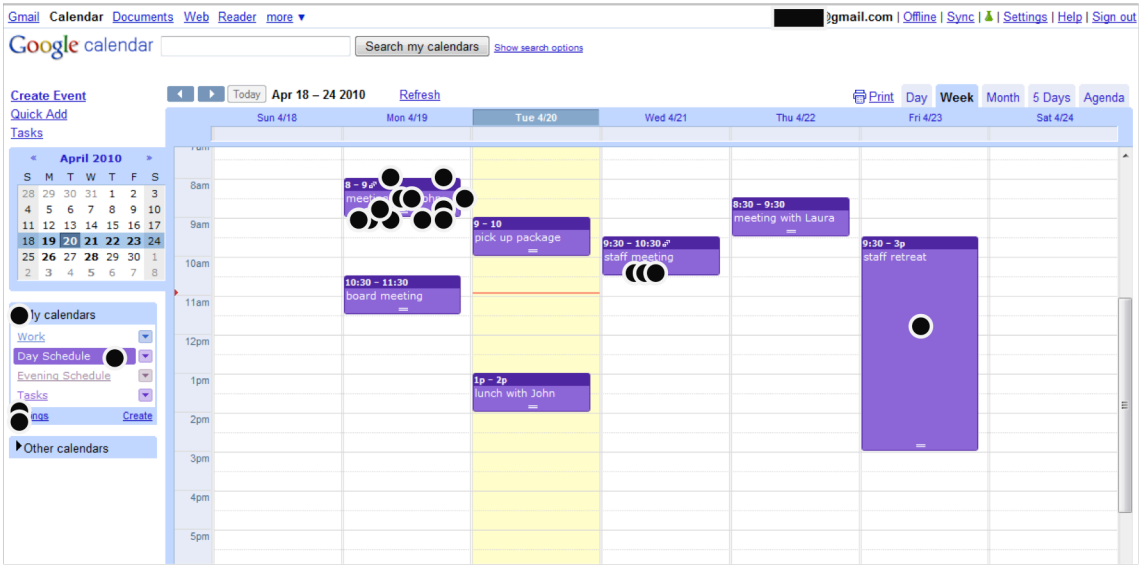
My study presented 20 participants with a series of 12 screen shots from popular web applications. I recruited participants from different age groups and with varying levels

of technical expertise. There were 11 females, and 9 males, aged between 19-60. There were 14 participants who described themselves as "non-technical" and 7 of the participants spoke English as a second language.

Each of the 12 screen shots conveyed a problem scenario consisting of a description and a printout of where a problem was encountered, as shown in Figure 8.2. All of the problem scenarios were taken from real questions from help forums for these web applications. I asked participants to pretend that they had a "magic wand" that they could use to point anywhere on the interface to get help and to indicate their selection with a physical sticker. I provided all participants with three stickers (labeled 1, 2, and X) for each scenario that could be used to indicate where the magic wand is being pointed according to this scheme: sticker #1 was used to indicate the first choice in pointing the magic wand; sticker #2 was used to indicate the second choice, and sticker X was used to indicate that there was nothing relevant on the current screen.



Scenario. You are trying to add photos from a recent trip to your Facebook page for the first time. You heard from a friend that it's easy to upload multiple photos from your hard drive, but when you arrived at this page, you did not see any such option. You are wondering if you came to the right place and what you should be doing next to upload your photos.



Scenario. You are currently looking at your day schedule and a little confused with all the events happening this week that are regular versus those that will only occur this week. So, you think it would be useful to change the background color of your repeating events to make them stand out from the rest.

Figure 8.2: Aggregate results from a task in our formative study and its corresponding scenario .

8.3.1.2 Formative Study Findings

There were two major findings from the study. First, participants tended to select labels in the UI that they believed were *conceptually relevant* to the help problem. Most of these keywords were application-specific labels or headings (*e.g.*, 15 of 20 participants selected the "*Create Album*" keywords in the scenario in Figure 8.2). Second, when no label appeared relevant, participants selected UI elements that were similar in terms of their visual appearance and location on the screen, with a bias towards the top-left. These findings suggested that LemonAid could determine similarity between selections largely based on the text on UI labels, and leverage additional attributes of the selected DOM object such its layout position and appearance.

8.3.1.3 Contextual data in HTML

Based on the findings from the formative study, I designed LemonAid to capture the three contextual details listed in Table 8.1. When a user clicks on a region of a page, LemonAid first determines the topmost DOM node (based on *z-order*) under the user's cursor. From this, it extracts the tag name of the selected node (`nodeType`). It also extracts the XPath string representing the sequence of tag names and child indices that indicate the path from the root of the DOM tree to the selected node (`nodeXPath`). Finally, it extracts all of the text node descendants of the selected node, concatenated into one string, using the standard *innerText* or *textContent* property of the selected DOM node, depending on the browser (`nodeText`). If the selected node is an image and includes an *alt* attribute, this text is also concatenated to `nodeText`. While I also considered using HTML *id*s or *class*es, since they are also related to appearance and layout, they are often dynamically generated between user sessions and thus not useful for representing questions shared by users.

Since the text labels on DOM elements could potentially be user-generated and privacy-sensitive, LemonAid only stores the `nodeText` if it is a known *UI literal*. UI literals include any string that is explicitly defined in the application source code or any whitespace-delimited string that appears in application resource files, such as localization files. A UI literal may represent labels on UI widgets, headings, or error messages, among other application-specific strings. Every time a user makes a selection, LemonAid compares the `nodeText` of the selected node against a whitelist of known application UI literals to determine whether or not to store the captured `nodeText`. For example, for a heading element where the text is "Account Settings," the `nodeText` would only be stored as part of the contextual data if "Account Settings" was found in the list of application UI literals. In contrast, if the selected text were a container that included a user's username, this would likely *not* be in the whitelist and would therefore not be included in `nodeText`. (I describe LemonAid's built-in functionality for extracting UI literals in a later section discussing LemonAid's integration steps.)

Attribute	Description	Example
`nodeText`	Visible text on the selected DOM node	"Bill Payer"
`nodeXPath`	The XPath uniquely identifying the selected DOM node in the page	`/HTML/BODY/TABLE/TBODY/TR[5]/TD`
`nodeType`	The HTML tag of the selected DOM node (i.e., DIV, TABLE, BUTTON, etc.)	`TD`

Table 8.1: Contextual data captured in a user selection with example from the bill payer scenario in Figure 8.1.

8.3.2 Questions, Answers, and Users' Selections

Once the user makes a selection and LemonAid captures the data in Table 8.1 to represent it, the data is used as a query to LemonAid's repository of questions. To explain this retrieval, I first define how questions and answers are stored. Each *Question* submitted by a user through LemonAid's question box (Figure 8.1.4) includes a unique id for each question in the set (*id*); the `nodeXPath`, `nodeType`, and `nodeText` described in Table 8.1 (*contextualData*), and the optional question text provided by the user in the text field in Figure 8.1.4 (*questionString*). Since the focus of this chapter is on

help retrieval, and not help authoring, LemonAid's *Answers* are basic, mimicking the kind of answers found in discussion forums: each *Answer* has a unique *id*, an *answerString*, which stores the text provided in the field in Figure 8.1.6, and a *questionID*, linking it to the question for which the answer was provided. *Questions* may have multiple *Answers*.

A *UserSelection* in LemonAid consists of the contextual data captured from the user's selection (as described in Table 8.1) and optional `searchTerms`, which store the text provided by the user in the search input field (Figure 8.1.4). A *UserSelection* initially generates a query consisting only of the contextual data. Each keystroke in the search input field generates a new *UserSelection* and updates the retrieved results using an *auto-suggest* interaction.

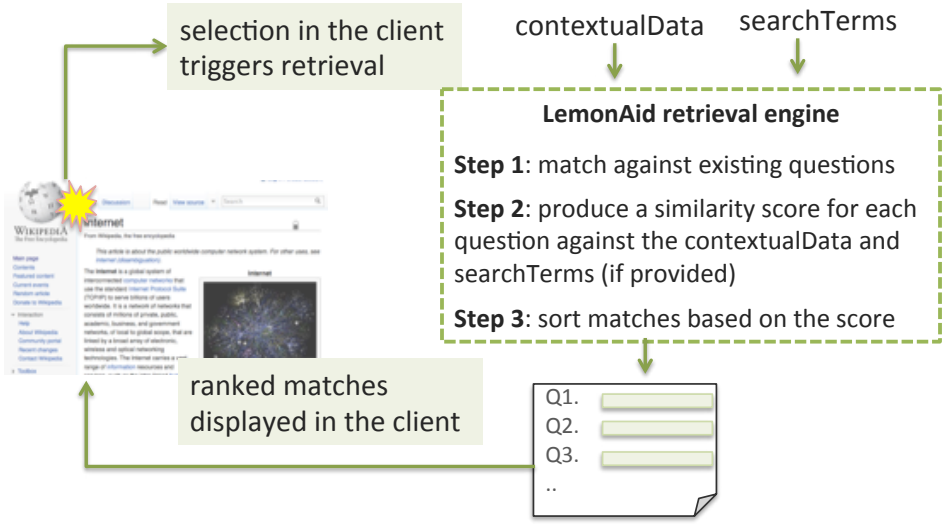


Figure 8.3. The retrieval engine uses contextual data from the user's selection and any search terms provided by the user to produce a ranked list of relevant questions

8.3.3 Ranked Retrieval of Matching Questions

To retrieve help, LemonAid utilizes a relevance ranking approach leveraging *contextualData* and optional *searchTerms* using the process shown in Figure 8.3. The retrieval algorithm takes a *UserSelection* and compares it to each previously asked *Question* in the application's repository, producing a *similarity score* between 0 and 1 for

each *Question*, with 1 being the best match. LemonAid then presents the matching questions in descending order of score. The score, which is computed as in Figure 8.4, is a combination of factors, including a `contextScore` based on a weighted sum of the three scores in Table 8.2 and a `textSimilarityScore` if the user provided `searchTerms`. The next two sections describe these scores in detail.

```
findMatchingResults(UILiteral, nodeType, nodeXPath,
searchTerms)
returnValue: a sorted resultSet containing matching questions
from set of existing questions.
-----------------------------------------------------------------
for each question Qi in the set of existing questions Q
    nodeTextScore = Qi .nodeText contains UILiteral ? 1 : 0
    xPathScore = a percentage computed by comparing the
overlap in
    Qi.XPath and nodeXPath of the current selection;
    typeScore = Qi .nodeType string equals nodeType ? 1 : 0
    contextScore = .7 nodeTextScore + .2 xPathScore + .1
typeScore
    if contextScore > 0.25
       add Qi to the resultSet
    if searchTerms is non-empty
        compute textSimilarityScore with full text tf–idf weighting
        if textSimilarityScore > 0
            add Qi to resultSet
if searchTerms is non-empty
   sort resultSet by textSimilarity, then contextScore
else
   sort resultSet by contextScore
return resultSet
-----------------------------------------------------------------
```

Figure 8.4. Pseudocode for LemonAid's retrieval algorithm.

8.3.4 Similarity based on context

As discussed above, my formative study suggested that similarity between selections could largely be based on the text on UI literals. Therefore, the primary factor in the `contextScore` is the `nodeTextScore`, which is 1 if the `nodeText` of a selection contains (ignoring case) the `nodeText` of the *Question* being compared and 0 otherwise. With this approach, LemonAid is able retrieve a match related to a specific item in the container

(*i.e.*, a navigation menu item) even if the user's selection was the container itself. This factor is given a weight of 0.7, since it was the most important factor in my formative studies.

The 2nd factor in the `contextScore` is the `XPathScore`, which captures similarity in layout and position identified in my formative study. Although XPaths can change as user interfaces layouts evolve over time (Adar et al., 2009; Bolin et al., 2005), many menus and header items on a page stay relatively the same or have only slight layout differences over time. Therefore, this score is a measure of the percent node overlap between the `nodeXPath` of the query and a *Question*'s `nodeXPath`. I compute this by starting from the root and doing a node-by-node string comparison from root to leaf, incrementing the score by 1 every time there is a match, and stopping when there is no match or the last node of the shorter path is reached. I divide the final sum by the length of the longer XPath to get a percentage. (For example, the overlap between `HTML/BODY/DIV/` and `HTML/BODY/DIV/DIV/P/` is 3/5 or 60%). Because location and position were only a factor in a minority of the scenarios in my formative study, the `nodeXpath` score has a weight of only 0.2.

The 3rd and final factor in the `contextScore` compares the `nodeType` of the selected node to that of the potentially matching *Question*. This factor accounts for both appearance similarities, while also helping with situations where multiple UI elements share the same text label, such as a button and a heading with the same words. The `nodeTypeScore` is 1 if the labels of the selection and the Question are equivalent and 0 if not. Because ties were rare, and appearance was only rarely a factor in my formative study, I only give `nodeTypeScore` a small weight of 0.1.

After `contextScore` is computed, the algorithm in Figure 8.4 includes a *Question* in the results set if its score is above 0.25. This threshold was selected because it implies that there is no `nodeText` match, but there is a strong match between the `nodeXPath` and `nodeType`. Even though this type of match is weaker than one based on `nodeText`, it is

useful for cases where a question may be attached to a container or non-UI literal text (*e.g.*, user-generated content). Since `nodeText` similarity is not relevant in such cases, the `nodeXPath` and `nodeType` similarity can still be used as (weak) indicators of relevant questions.

8.3.5 Similarity based on search keywords

If the user provides `searchTerms` in the search field in Figure 8.1.4, the algorithm in Figure 8.4 also computes a `textSimilarityScore`. It does this by comparing a query's `searchTerms` with the whitespace-delimited words in each existing *Question*'s *questionString*. To compare the similarity between these keywords, I created a search index on each *questionString* and used a standard full-text search feature based on the vector space model (Salton et al., 1975). The similarity score is computed using the classic term frequency–inverse document frequency (*tf–idf)* weighting approach in information retrieval. The main idea of *tf-idf* is that terms that occur frequently in the target document (in my case, a question in the repository of previously asked questions), but less frequently in the whole document collection are the useful terms. The weight of these terms is a combination of term frequency within the target document and its frequency across all documents. Each *Question* that matches the user's *searchTerms* is included in the result and sorted in descending order based on the `textSimilarityScore`. This result set is then sorted by `contextScore` of each question against the *UserSelection*.

Constituent	Similarity Score	Weight
`nodeTextScore`	string contains (1 or 0)	0.7
`XPathScore`	% node overlap between XPaths [0,1]	0.2
`nodeTypeScore`	string equals (1 or 0)	0.1

Table 8.2. Weights for contextual data.

8.4 Integrating LemonAid Into Web Applications

One of the strengths of LemonAid's simplicity is that it can be easily integrated into an

existing website with minimal modification to the site itself.

First, site administrators choose an ID to uniquely identify their application-specific help information in the third party server. Next, administrators can either provide a URL to their main source directory or run a script provided by LemonAid to extract UI literals from a site's code and localization files. From this, LemonAid generates a CSV file containing the list of literals and stores it alongside the question repository. LemonAid uses a simple algorithm for finding string literals in commonly used web programming languages, looking for sequences of characters delimited by single ('') and double ASCII quotes (""). While this approach does not account for UI literals that may be dynamically generated, it covers a large range of UI literals defined at design time. While this extraction approach may generate false positives (extracting strings that do not appear in the user interface), these non-UI literals are not visible to the user and hence not selectable anyway. Furthermore, site administrators have full control in editing the CSV file containing string literals from their source code.

Finally, site administrators include a few lines of JavaScript on all of their web application's pages, just as with analytics services such as *Google Analytics*. Doing so links the LemonAid source code to the web application, and makes LemonAid functional on that page (example of literals and JavaScript code is shown in Figure 8.5). The interface shown in Figure 8.1 is an example implementation of LemonAid on the static version of *Bank of America*'s Bill Payer site. The UI literals were obtained by manual screen scraping since I did not have access to Bank of America's source.

The current implementation of LemonAid sets up a basic infrastructure through which anyone can anchor questions and answers on the underlying application's UI literals. Site administrators may have different needs in terms of managing the Q&A and the related community of users (these issues will be discussed in more detail in Chapter 10 where I discuss field deployments of LemonAid). For example, some web applications are

already controlled by user authentication and it may be just a matter of integrating LemonAid with the existing user accounts on the site. Another approach may be the use of social networking plugins such as *Facebook Connect* to facilitate communication among users within their social network. In other cases, administrators may want to restrict answer authoring to the company's support personnel and may want to store the help data locally.

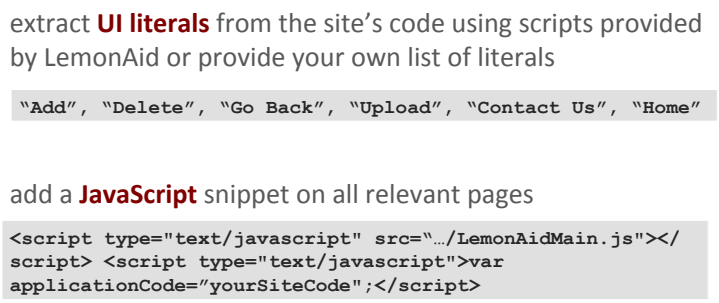


Figure 8.5: Integrating LemonAid into Web Applications

8.5 Summary

This chapter introduced the concept of selection-based crowdsourced contextual help retrieval and the design of LemonAid system. I first presented a formative study that was used to determine what type of contextual data should be captured to enable this selection-based retrieval in web applications. The key insight from this formative study was that users made similar UI selections for similar help needs and different selections for different help needs. This tight coupling of user needs to UI elements has been central to providing an alternative to users' textual queries that are plagued with unnecessary variation. Chapter 8 also introduced LemonAid's flexible architecture for embedding this new help approach in any web application and in Chapter 10, I will show how this architecture was extended and used to carry multiple live deployments of LemonAid. In the next chapter, I present LemonAid's retrieval evaluation that assessed to what extent LemonAid could find relevant results using UI selections alone in an interactive application. In Chapter 11, I reflect on the larger vision underlying

LemonAid's approach to crowdsourced contextual help and discuss issues around scope, scalability, robustness, and privacy.

CHAPTER 9

Retrieval Evaluation of LemonAid's Selection-Based Crowdsourced Contextual Help Approach

As discussed in Chapter 8, at the core of LemonAid is a retrieval engine that produces a ranked list of questions relevant to a user's selection and optional search terms. Although users' natural language descriptions of the same problem may differ, my formative study in Chapter 8 showed that users tend to make the similar selections in the UI for a given problem. But, does this finding hold in the context of an interactive web application? And, how effective is LemonAid's underlying retrieval algorithm at retrieving relevant questions and answers using only users' selections? These are the questions addressed in this chapter that assesses the effectiveness of LemonAid's retrieval algorithm using a crowdsourced corpus of UI selections.[14]

I focused the evaluation of the algorithm on answering the following question: across a corpus of help problem scenarios, *how effective is LemonAid at retrieving a relevant question asked by another user using only the current user's selection*? To operationalize this, I measured the rank of the first relevant question (Figure 9.1.2) for a given help problem, using only the contextual data from the user's selection (Figure 9.1.1).

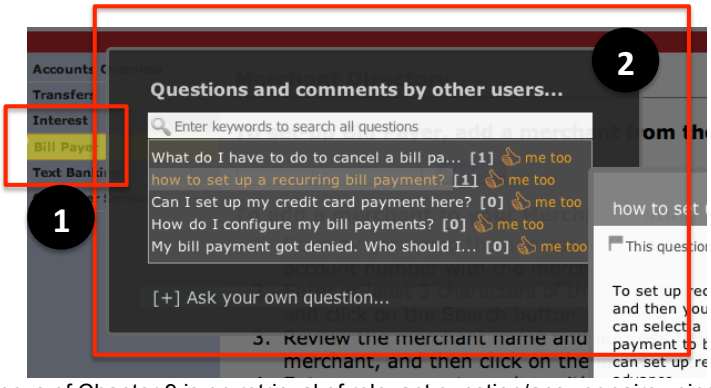


Figure 9.1: The focus of Chapter 9 is on retrieval of relevant question/answer pairs using only the user's selection [going from (1) to (2)]

[14] This chapter has been adapted from my CHI 2012 publication (Chilana et al., 2012b)

9.1 Developing a Crowdsourced Corpus

To perform this assessment, I first needed a large corpus of LemonAid help selections. Since I did not have a site with a large number of users to which LemonAid could be deployed (as website owners viewed the adoption of the tool without evidence of its efficacy as risky), I developed a corpus using a simulated community of users through Amazon's Mechanical Turk (mTurk) platform.[15]

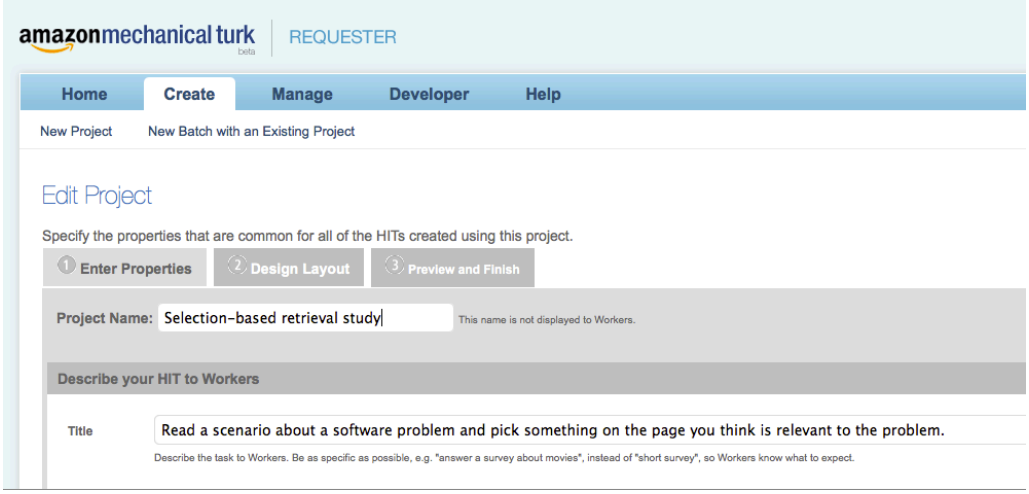


Figure 9.2. Requester Interface Used to Set up mTurk tasks

9.1.1 Using Mechanical Turk in Behavioral Studies

mTurk is an online marketplace where workers receive micro payments for performing small tasks on the web, termed Human Intelligence Tasks (HITs). Recently, mTurk has become a popular way for researchers to recruit a large number of participants for small tasks in behavioral studies in HCI, economics, political science, among others (Kittur et al., 2008; Paolacci et al., 2010; Snow et al., 2008). Three unique benefits of using mTurk and crowdsourcing platforms for running online studies have been recognized as: (1) subject pool access, (2) subject pool diversity, and (3) low cost (Mason & Suri, 2012).

I used the mTurk platform to design a study similar to the magic wand study

[15] https://www.mturk.com/mturk/ (Retrieved May 29, 2013)

discussed in Chapter 8 to capture users' selections in the context of application screens where users were most likely to seek help. The main approach of this study was to have hundreds of web users read a detailed help scenario (described next) and perform a LemonAid help request by selecting a UI element and providing a question relevant to the scenario. I set up a requester account on mTurk (Figure 9.2) and used the administrative interface to devise and publish these tasks.

9.1.2 Help Scenarios

To ensure that my corpus of help scenarios was realistic, I began by selecting the first 100 questions tagged as popular or recently asked in the *How Do I* category of Google Calendar's help forum ("Google Calendar Forums," 2011). I chose Google Calendar because it is a popular application used by millions of people and offers not only basic functionality, but also a range of advanced functions that people have trouble finding and using. From my sample of 100 popular or recently asked questions, I eliminated questions that appeared to be duplicates and created a random sample of 50 questions that I could use in my evaluation. Although there were many more than 50 questions in Google's help forums, by analyzing the 10 "related discussions" that Google lists alongside each thread, I found that many of these discussions concerned the same issue and believe that 50 questions represented a substantial proportion of the common problems. This is reinforced by previous studies that have shown that there often are a large number of duplicate discussions on forums (Singh et al., 2009) and other forms of issue reports (Bettenburg et al., 2008).

To convert the help discussions into scenarios, I identified the expected or desired behavior identified by the help requester and wrote a textual scenario to represent it (*e.g.* Figure 9.3.a). I also included a motivation for the task in the scenario and details about Google Calendar functionality to help a user unfamiliar with the application understand the specified goal.

After the users' had a chance to read the scenario, they were asked to answer two comprehension questions (*e.g.* Figure 9.3.b and 9.3.c). The goal in including these questions was gain some confidence that participants understood the scenario and were not selecting user interface elements randomly (a common problem in some mTurk studies (Downs et al., 2010; Kittur et al., 2008). Each comprehension question had 5 items; the scenarios and questions were carefully edited by another researcher for clarity. If users answered one of the questions incorrectly, they were given another explanation of the scenario to help them understand the scenario better.

(a) Scenario:	*You just added a new event to your calendar and noticed that by default your status "Show me as" has been set to "available" instead of "busy". You think this is rather odd since previously when you added a new event, your status was set to "busy" by default. You would like to change your default settings so you always appear as "busy" when you add an event.*
(b) Comprehension question #1	This task is about changing default settings for: a. Not sure. b. Event location. c. Availability status. d. Event time. e. Privacy setting.
(c) Comprehension question #2	After this change, your next event will have which of these default settings: a. The event will be public. b. The event will be private. c. The status will be available. d. The status will be busy. e. Not sure.

Figure 9.3: Example scenario for a task given on mTurk.

9.1.3 Interactive mTurk HIT creation

Based on the help scenarios that I extracted, I created a Google Calendar page representing an application state in which a user might encounter those problems. I created the HTML pages for each scenario by manually recreating the chosen Google Calendar state and scraping the application's corresponding HTML for that state. I then augmented each scenario page LemonAid's question-asking functionality. Since

LemonAid requires a list of UI literals corresponding to the application and I did not have access to Google Calendar's source code, I manually extracted a set of UI literals by scraping each visible text label (and ALT text of images) from all user interface elements for each scenario page. Finally, because the focus of my study was retrieval performance on users' first selections and not on expert use of LemonAid, I disabled LemonAid's help retrieval and answer authoring functions for the study, so that after a participant selected a user interface element and wrote a query, their task was complete. This reduced the possibility that participants would change the type of selections they made after completing multiple HITs based on the type of results returned by LemonAid's retrieval algorithm. Figure 9.4 shows an example of an interactive web page augmented with LemonAid help functionality and instrumented to collect data for the scenario listed above.

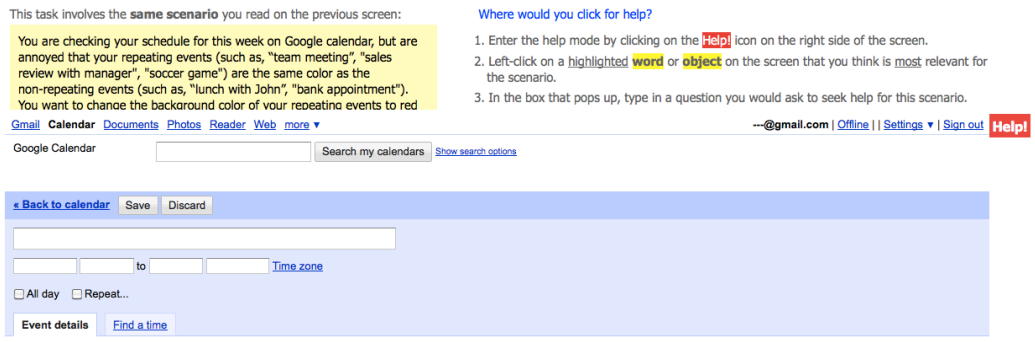


Figure 9.4: Example scenario interface for a task given on mTurk.

Of the 50 help problems, 8 were likely to be encountered in different contexts (for example on the main calendar view, or in a configuration dialog); for these, I created two scenarios, each with a different application state, resulting in a total of 58 scenarios overall.

My mTurk HIT presented participants with one of these 58 help-seeking scenarios, including the scenario text and the static HTML page with interactive LemonAid features. Users were asked to (1) read the scenario, (2) answer two multiple choice comprehension questions (described above), (3) enter the help mode, (4) select one of the highlighted

words or elements on the screen that they felt were most relevant to the problem (Figure 9.4), and (5) provide a question in their own words that they would ask to get help in the given scenario. A summary of this task as an mTurk HIT is illustrated Figure 9.5.

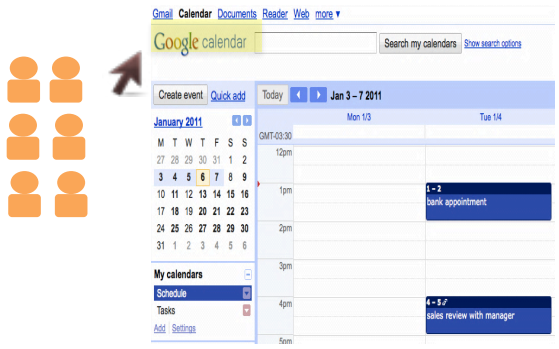
An mTurk *HIT* consisted of:

1) Reading a problem scenario related to Google Calendar.

> You share your calendar with all team members at your company. Recently, one of the members left the company and now you would like to stop sharing the calendar with her, but cannot remember how to change the sharing details.

2) Answering 2 comprehension questions related to the scenario.

3) Making a selection relevant to the problem on a corresponding Google Calendar page instrumented with LemonAid.



4) Formulating a help-seeking question based on the scenario.

```
How to remove a person from a shared calendar?
```

Figure 9.**5**: Illustration of mTurk use in developing a corpus

Each mTurk HIT was launched with 55 assignments per HIT, with the intent of gathering 50 selections per scenario. I used 5 of the 58 HITs to pilot the mTurk protocol and my data collection strategy, resulting in a final data set of 53 unique HITs. I paid users $0.15 per HIT. (Each HIT took an average of 3.5 minutes to complete.) The study (including the pilot tests) ran for about 5 weeks. To prevent duplicate responses and other mischief, I asked mTurk users to provide a unique 6-digit passcode that was generated

after they made an on-screen selection for a particular HIT. I also asked mTurk users had to write a brief explanation for why they made a particular selection.

After obtaining the data, I computed the time that an mTurk user spent on the task and compared it to the average completion time (3.5 minutes). If this time was below the 20% of the average (i.e., less than 45 seconds), I automatically eliminated the response. For responses that fell between 45 seconds and 3.5 minutes, I manually checked the written explanation of why a particular selection was made. If the explanation was not intelligible, I excluded that response. Finally, I also checked the passcode that mTurk users provided against the passcodes generated by my system and eliminated responses that had incorrect passcodes. These three data points together allowed us to detect user interface selections that appeared to be hastily selected with no apparent comprehension of the scenario. I was able to use between 47-52 selections for each HIT (about 10% of the data contained invalid selections as per the above criteria). My final corpus included 2,748 help selections from 533 different mTurk accounts across 53 scenarios.

9.2 Results

Because LemonAid uses a ranked retrieval approach where the retrieved Questions are presented in an ordered list, traditional metrics of recall and precision designed for set-based retrieval were not appropriate. Instead, I focused on computing the rank of the 1st relevant Question for a given selection of all retrieved results.

9.2.1 Rank-based retrieval evaluation

I defined a relevant Question as follows: for a given help selection corresponding to one of my 53 help scenarios, any retrieved Question that concerned the same help scenario was deemed relevant. This meant that there could be multiple relevant results for a selection, since there were multiple other selections corresponding to the same scenario.

Ranks were computed for all 2,748 selections in the corpus, retrieving relevant results from all other selections in the corpus using only the contextual data in the selections (excluding participants' query text). LemonAid retrieved 1 or more results for 90.3% of the selections. Figure 6 shows the proportion of queries resulting in median ranks of 1 through 10. The median rank of the results across the whole corpus was 2, meaning that the relevant result was likely to be in the top 2 results.

$$\mathrm{MRR} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{1}{rank_i}$$

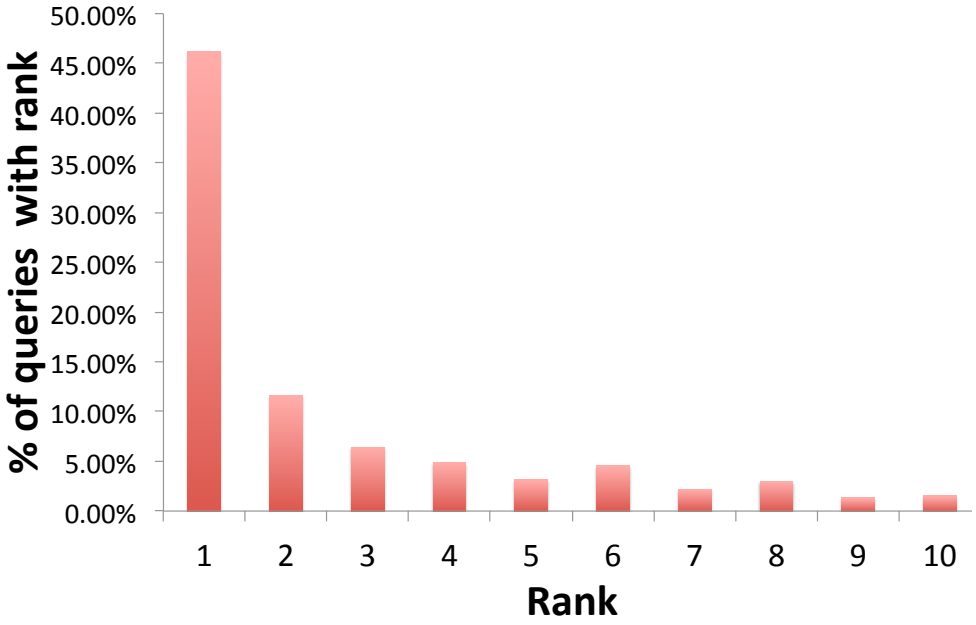


Figure 9.6. Distribution of ranks in the corpus.

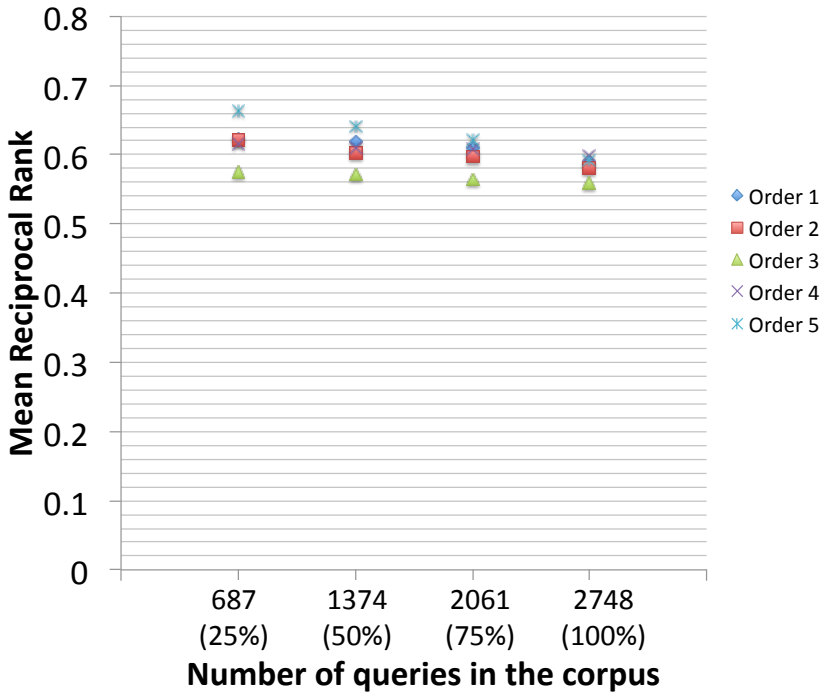


Figure 9.7. MRR for different corpora sizes and orderings.

To assess performance across the whole corpus more systematically, I computed the Mean Reciprocal Rank (MRR). MRR values are bounded between 0 and 1 and are sensitive to the rank position (e.g., from rank 1 to 2, MRR falls from 1.0 to 0.5). The

reciprocal rank of a result is equivalent to the multiplicative inverse of the rank of the first relevant result. The MRR is computed as the average of the reciprocal ranks of results for a set of queries in corpus C where $1/rank_i$ is the inverse rank of the ith query in C, and $|C|$ is the size of the corpus. The resulting MRR was 0.5844, meaning that the average rank of the result across the repository (taking into account all the best and worst-case ranks) was between 1 and 2.

9.2.1 Performance over time

While the overall performance of the retrieval algorithm is important, its performance over time, as users ask more questions, is also important. To investigate the effect of corpus size, I randomly selected 5 subsets of queries of four different corpus sizes (25%, 50%, 75%, and 100 % of the 2,748 queries). Figure 7 displays the MRR for these 20 corpus subsets, showing that while MRR degrades as the number of selections increase, it degrades quite slowly. A live deployment of LemonAid would obviously introduce other factors that would affect these outcomes; for example, there might be many more help problems. However, these results show that users would also be more likely to find an existing question about a problem rather than ask a new one.

9.3 Limitations

The retrieval evaluation has some limitations that that should be considered when interpreting the results. For example, these results might only hold for the type of users represented by mTurk workers (Kittur et al., 2008). Also, this study's corpus does not represent users in actual help scenarios; the selections that users make when actually needing help could differ. Moreover, my evaluation did not explore the effect of LemonAid users *interactively* exploring LemonAid search results, which may also affect the overall utility of LemonAid.

9.4 Summary

This chapter presented an evaluation of LemonAid's retrieval using a crowdsourced

corpus of selections generated by a simulated community of users on mTurk. The overall results of LemonAid's retrieval were promising: based on a corpus of over 2,700 UI selections from over 500 users on mTurk, LemonAid could retrieve a relevant result for 90% of help requests based on UI selections and, of those, over half had relevant matches in the top 2 results. The main implication of this finding is that in most cases users would only have to make a UI selection in the interface that they think is relevant and they would see a relevant question (and answer, if available). This is a dramatic improvement over traditional natural language queries for help on the broader web, which require substantially more effort and are inconsistent among different users. The key phenomenon that facilitates the retrieval of high-ranking relevant results is that users' queries are restricted to a smaller and more focused set of user interface selections instead of natural language text and that users tend to select similar labels for similar problems and different labels for different problems (as discussed in Chapter 8). Still, many questions remain about the utility of using this help approach in real applications in the context of real tasks. Chapter 10 discusses multiple live deployments of LemonAid that were used to assess the helpfulness and reuse value of the selection-based contextual help approach from the perspective of end users and software teams.

CHAPTER 10

Field Evaluation of LemonAid's Selection-Based Crowdsourced Contextual Help: Perspectives of End Users and Software Teams

Chapter 9 focused on the evaluation of LemonAid's selection-based contextual help retrieval approach introduced in Chapter 8. Although results indicated that LemonAid could retrieve relevant help in the top 2 results for over half of help seeking scenarios, this finding was based only a simulated community of users and pre-defined tasks. Many questions remained about the actual utility and helpfulness of this approach in practice. In this chapter, I present a multi-site field deployment study of LemonAid that investigates the ecological validity of this new help approach. While a large focus of this study is on capturing end users' perspectives, this study also considers the perspectives of software teams that integrated LemonAid into their applications. I tie back these findings to the larger context of software issue reporting (discussed in Chapters 6 and 7) and to the overall process of user-centered design (Chapters 4 and 5). [16]

10.1 Background and Motivation

As discussed earlier, recently we have seen the emergence of *crowdsourced contextual help* systems that integrate social forms of help within the application's user interface (UI). For example, *TurboTax* help ("TurboTax Support," 2013) and *IP-QAT* (Matejka et al., 2011) display help discussions relevant to the current view in a sidebar within the application. LemonAid (introduced in Chapter 8) is also a crowdsourced contextual help system that lets users retrieve Q&A at an even finer granularity by selecting a label, widget, link, or another UI element.

Although crowdsourced contextual help systems hold a lot of promise in improving users' interactions with crowdsourced help, many questions remain about their

[16] This chapter has been adapted from my CHI 2013 publication (Chilana et al., 2013)

effectiveness at providing help during real tasks. For example, while IP-QAT's initial evaluation (Matejka et al., 2011) showed that it was more useful and easier to use than a basic discussion board, the study involved only one instrumented application and was carried out with paid volunteers who were *required* to contribute 3 items per day. LemonAid's initial evaluation discussed in Chapter 9 showed that LemonAid could retrieve relevant help in the top 2 results for over half of help seeking scenarios, but this finding was based only a simulated community of users and pre-defined tasks. As social systems often interact with the social and organizational contexts in which they are implemented (Grudin, 1988), it is difficult to know if and when these lab study results apply to real users and their real situations and tasks (Klein & Myers, 1999; Dan R. Olsen, 2007).

To understand how crowdsourced contextual help is perceived in real settings and to increase the ecological validity of LemonAid's design (Chapter 8), I partnered with multiple software teams to deploy and evaluate LemonAid in the field. After extending the original LemonAid prototype with community Q&A, answer notification, and back-end analytics features, I deployed it on a set of four diverse sites for periods ranging from 7 to 15 weeks, reaching user populations of 150 to over 40,000 users in size. From over 1,200 logs, 168 exit surveys, and 36 interviews with end users, I report on end users' perspectives on LemonAid's helpfulness, usability, and desirability for reuse, and how it compared to alternative forms of help seeking. I also present software teams' perspectives on issues related to LemonAid's adoption and integration and the utility of LemonAid's analytics data.

The result of these deployments is the first holistic picture of the adoption and use of a crowdsourced contextual help system, contributing: 1) Extensions to LemonAid's crowdsourced contextual help (Chapter 8) that support participation by end users and software teams, moderation of help activity, and improved discoverability of help content;

2) Empirical findings that highlight the unique strengths of crowdsourced contextual help and selection-based help in particular, but also tradeoffs in integrating help into an application user interface; 3) Results that illustrate the factors affecting software teams' adoption of a crowdsourced contextual help tool and teams' perceptions of the value of crowdsourced contextual help analytics data; 4) Insights into the process of developing and integrating help systems into existing applications, and how help systems fit into larger social and organizational contexts.

10.2 Field Study Approach for Evaluating Help Systems

Although "in-the-wild" field studies are increasingly common in HCI (Brown et al., 2011), field studies of software help have been rare. Although not directly in the domain of software help, Ackerman's (Ackerman, 1998) evaluation of *AnswerGarden*, a question-answering tool, comes closest to my study design. The study used multiple data collection procedures including questionnaires, usage data, and interviews and the findings provided rich insights into whether augmenting organizational memory was possible, and also contributed a number of lessons for designing organizational memory systems. Brandt et al. (Brandt et al., 2010) evaluated their *Blueprint* tool for programmers and followed-up with a 3-month deployment at a large company, collecting usage data and interviews. The field study component helped them gain valuable insights into how participants integrated Blueprint into their everyday programming workflow. Although not a field evaluation, the study of IPQAT is one of the first to evaluate crowdsourced contextual help with users (Matejka et al., 2011, p. -)]. While the study ran for 2 weeks and there were 36 participants, all of these participants were using the same application and most of them were already users of product discussion forums. Furthermore, these participants were required to contribute 3 items per day and received additional monetary incentives to contribute more. In contrast to these works, my field study investigates the use of crowdsourced

contextual help by hundreds of users for their real tasks across multiple sites.

Figure 10.1. Main components of the LemonAid interface in the help mode: (1) a user selects an on-screen label or image highlighted in yellow; (2) the user's selection triggers the retrieval of relevant questions; (3) the user can click on a question to see the answer(s) for that question and indicate whether the answer was helpful or not helpful.

10.3 Extending LemonAid For Field Deployment

The key idea behind LemonAid is that users enter a semi-transparent help mode overlaid onto the application's user interface and find help by selecting a label or image they believe is relevant to their problem (as seen in Figure 10.1.1). Upon a selection, LemonAid retrieves questions and answers related to the selection (Figure 10.1.2) based on an algorithm that retrieves help based on the text, tag type, and location of the selection within the user interface (described in detail in Chapter 8). Users can then select a retrieved question to view its answers and indicate their helpfulness, or if they do not find a relevant question, they can submit one. Support staff and other users can then answer them. Users can also use keywords to search through existing questions, allowing LemonAid to behave like a site-specific Q&A search.

The prototype described in Chapter 8 included only a retrieval algorithm and a selection user interface. For deployment, I added several critical features:

Improving discovery of existing help content. One of my design goals was to facilitate users' navigation of the existing help content upon entry into the help mode. To facilitate this, LemonAid adds yellow question marks next to elements that have at least one question attached (Figure 10.1.1). These questions marks are dynamically generated as soon as a user enters the help mode based on the selections of previously asked questions.

To help users remember which UI elements they have already viewed, the system modifies the color of the visited question marks.

Encouraging user participation. Another goal was to encourage users to contribute questions and answers. When users add a new question in LemonAid, they can provide their email address and be notified of new answers without having to return to the site. LemonAid also allows users to report potential spam and offer "me too" votes on questions (Figure 10.1.2). When users view answers, they also contribute data on whether the answer was helpful or not helpful in order to dismiss the answer window (Figure 10.1.3).

Moderating help content. Many of the software teams that I approached were concerned about users' ability to deface their site with unwanted content. Therefore, I added basic a moderation feature, allowing support staff to receive e-mail notifications of new questions and answers so that they can both approve content, answer new questions, and improve user-contributed answers. The moderators also get notified when a user flags a post as inappropriate.

Analytics dashboard for monitoring help activity. I added a web-based administrator "dashboard" that shows an aggregate view of users' activities in LemonAid, specific to the host deployment site. Moderators can get a real-time snapshot of the elements where users are looking for help. They can also see rankings of popular questions based on views and votes, and the helpfulness of answers.

Implementation and setup: To set up LemonAid for each site, I needed: (1) a whitelist of the UI literals where Q&A would be attached in the interface, to prevent users from selecting privacy-sensitive content; and, (2) access to the team's source to include the LemonAid functionality on the site. I hosted all of the Q&A data, logs, and scripts related to LemonAid on my own servers. For (1), the previous version of LemonAid in Chapter 8 offered a mechanism for extracting UI literals from the source code; teams desired more

control over what part of the interface would actually become selectable on the screen, so I created a plug-in that would help us interactively choose selectable labels and images on specific pages. For (2), software developers only had to embed a link to the main LemonAid JavaScript file in their source code (as is required to use tools such as *Google Analytics*, for example). One team decided to embed this link at the root level so that LemonAid would appear on all pages in the site; the other three teams placed LemonAid on their home page and selectively on other frequently accessed pages.

I also created a client-side browser plug-in that allowed us to test LemonAid on any site. Except for mobile devices, I was able to support LemonAid on most major browsers and platforms (although I did have problems with LemonAid on Internet Explorer versions 8.0 and lower for two of the deployments). I also had to resolve some minor JavaScript namespace conflicts on sites that used different versions of common JavaScript libraries, such as *jQuery*. The plug-in allowed us to sort out most of the compatibility issues independently of the host software teams, before deployment.

10.4 Method

10.4.1 Field Deployment Sites

The context for all of my deployment sites was a large university and its various software teams developing web applications to support teaching and research. To recruit field deployment sites, I met with software developers and site administrators across campus. Prior to making a commitment, the teams had a chance to see LemonAid demos, learn details about the field study, and discuss time and effort that they would be willing to commit. Six teams were enthusiastic about participating; the first site served as a pilot deployment and one of the remaining five had to leave the study due to conflicting schedules. The pilot site was a local intranet used by about 20 staff and students for internal project management. The pilot site allowed us to assess the usability of the interface, uncover implementation issues that had to be addressed

for larger cross-platform deployments, and collect pilot data for improving my set-up and logging. I describe the four web sites augmented with LemonAid in Table 10.1.

Before making LemonAid live, I asked each software team to seed the help database with FAQs or other help content relevant to each page. In two cases, additional staff and the first author were also involved in seeding the help content.

10.4.2 Mixed-Method Data Collection

I treated each deployment as an individual case (Yin, 1992), but used the same data collection procedure for each site. I used a mixed-method approach to better capture the plurality of experiences and perspectives of users (Greene & Caracelli, 1997; Yin, 1992). I collected data from three sources: (1) usage logs, (2) surveys, and, (3) interviews. The data collection occurred during the spring and summer of 2012.

Site	Description	User Population
LIBRARY	Provided access to a wide range of print and electronic resources available within the university library system and on web. One of the largest academic library systems in the US with millions of cataloged books, journals, and other materials.	Over 40,000 university students, faculty, and staff, visiting scholars, and the general public.
DEPT	A large academic department site that offered information about courses, upcoming events, admission, faculty, collaboration opportunities, and research facilities.	Over 900 undergraduate and graduate students, faculty, staff, prospective students, and the public.
EDC	An electronic data capture (EDC) system used for collecting data from clinical trials in an electronic format. Offered features for data entry, creating reports, analyzing data, exporting data, and sharing data with multiple users.	Over 500 staff personnel within the medical school and affiliated research center (restricted access).
RDB	A records database management system for storing and organizing data needed to maintain internal records of staff researchers. Used periodically by staff to review and update personal information, publications, grants, and to collaborate.	Over 150 staff and researchers in a unit within School of Medicine (restricted access).

Table 10.1. Summary of the four deployment sites.

10.4.2.1 Usage Log Data

I instrumented LemonAid to gather a time-stamped log including: entry into help mode, exit from help mode (by clicking on the exit help button or leaving the current page), selections of elements on the screen, selections of questions, helpfulness of answers viewed, votes on questions, flags used to mark questions as spam, content of submitted questions and/or answers, content of search keywords, and lists of users subscribed to receive answers to questions. LemonAid gathered all of the foregoing data anonymously, meaning that I could not ascribe activity in the help mode to individual users, only users in aggregate.

10.4.2.2 Exit Survey

When users exited the help mode for the first time (detected by LemonAid's browser cookie), they were presented with a brief exit survey. Users had the option of being included in a $50 gift card drawing as an incentive to participate and could win additional prizes if they wanted to participate in a follow-up interview. Survey responses and identities of the respondents were not linked.

The survey asked users three questions about their experience, each scored on a 7-point Likert scale, ranging from "strongly disagree" to "strongly agree." These questions were: (1) "I found something helpful;" (2) "I would use this help feature again," and (3) "the help feature was intuitive." I also asked users closed demographic questions about their role (e.g., staff, student, faculty, etc.), their frequency of website use (ranging from daily, few times a week, few times during the quarter, or few times a year), and preferred method of getting help for web applications ("using the site's help", "searching online", "asking someone", "calling a support line", or "trying on my own"). I required users to provide a response for each of these questions to maintain consistency in the responses.

10.4.2.3 One-on-One Interviews with Users and Software Teams

Using the list of survey respondents who wished to be interviewed (~56% of all

respondents), I recruited interviewees from each deployment site. I conducted all interviews on campus, except four were on the phone. The distribution of the interviewees were: graduate students (31.4%), staff researchers (i.e., post-docs, lab techinicians, 25.7%), undergraduate studnets (22.9%), administrative staff (17.1%), and faculty (2.9%).

The interviews were semi-structured and lasted around 30 minutes on average. My goal was to understand users' help seeking behavior in the context of modern help tools and how users saw LemonAid fit in with these tools. Interviewees had access to LemonAid during the interview to facilitate recall. I also probed into interviewees' perceptions of LemonAid's usability and usefulness, particularly compared to other help formats. I also probed into aspects of LemonAid that the interviewees believed were confusing or could be improved.

Near the end of each deployment period, I also interviewed the software team members involved in the deployments. I began the interviews by probing into teams' perspectives on the experience of integrating LemonAid compared to other forms of help. I also asked interviewees to describe how they currently managed user feedback and whether the feedback was used to inform design decisions. I also showed them usage data captured by LemonAid's Administrative Dashboard and asked them to explore and comment on the utility of the aggregated data, if any. These interviews lasted about 45 minutes.

10.4.2.4 Analysis

To assess how LemonAid was used in practice and whether LemonAid was helpful, usable, and desirable for reuse, I used the concept of data triangulation (Jick, 1979) and looked for convergence in the logs, survey data, and interview data. I began the analysis by parsing the usage logs to get an overview of usage patterns and help activity in LemonAid (the main variables are shown in Table 10.2). For the survey

responses, I investigated associations between my key outcome variables and user demographics.

To analyze the qualitative data, I audiotaped and transcribed each interview, then organizing, coding, and analyzing with qualitative data analysis software. In the first pass, I coded for data related to questions about LemonAid's usage, and end users' perceptions of utility and usability. In the next pass, I examined interviewees' descriptions of help seeking strategies using an inductive analysis approach (Miles & Huberman, 1994; Strauss & Corbin, 1998). This inductive analysis approach was useful for seeing patterns in the data and in identifying recurring themes across the different deployments. A similar inductive approach was also used for analyzing interviews with software team members.

10.5 Overview of LemonAid usage and users

The first set of results concern who used LemonAid and what they did with it, based on the usage data and demographic survey responses.

10.5.1 LemonAid Usage Activity

To describe the usage activity in LemonAid, I use an individual *help session* as the unit of analysis since I was unable to track individual users. I define a help session as an episode consisting of all the activities that occurred when a user entered the help mode and pressed exit or left the page. In my analysis, I ignored all of the help sessions where entry into the help mode was immediately followed by an exit out of the help mode or the current page. About 20% of the logs followed this pattern.

LemonAid's deployments across the four different sites resulted in over 1,200 help sessions. Table 10.2 summarizes this activity for each deployment site. Since the LIBRARY site was the longest deployment and had the largest user base, it yielded the highest usage of LemonAid (972 help sessions). The LIBRARY logs showed that 16 new questions were added during the deployment, constituting about 1.6% of the total help sessions. I also found that no end users answered a question; library staff answered all new questions. I

did find that the 16 new questions asked by users received 121 views, accounting for about 21.5% of all question views and 74.3% of the corresponding answers were marked as helpful. Prior work has shown that this level of activity is typical of what occurs in technical forums (Ko & Chilana, 2010; Lakhani & Von Hippel, 2003; Matejka et al., 2011)] and more broadly on the Internet (e.g., "*the 1% rule*" (Arthur, 2006)) where most users are consumers of online content rather than contributors.

	LIBRARY	DEPT	EDC	RDB
Length of deployment	15 weeks	6 weeks	8 weeks	8 weeks
# of help sessions	972	185	65	40
# of element selections	2057	509	141	91
# of answer views	562	194	61	36
% of viewed answers marked as helpful	77.7%	71.0%	75.1%	69.2%
# of new questions posted	16	7	3	5
# of answer views on new questions	121	27	30	13
% of answers to new questions marked helpful	74.3%	55.6%	67.7%	57.1%
# of search queries	43	23	4	4

Table 10.2. Summary of usage activity across the 4 LemonAid deployments.

10.5.2 Demographics of Survey and Interview Participants

I received 168 responses to exit surveys across the different deployments (with the highest number of responses from the LIBRARY site (n=121, response rate=12%), followed by the DEPT (n=33, response rate=18%), EDC (n=9, response rate=12%), and RDB (n=5, response rate=13%). A breakdown of the survey respondents based on their role is shown in Table 10.3. Note that the EDC and RDB sites required authentication and had respondents who were exclusively staff members at the university or an affiliated institution. The LIBRARY and the DEPT sites were largely public sites (with only some features requiring authentication) and garnered responses more broadly from the university's graduate and undergraduate students, staff, faculty, and external visitors.

Role	LIBRARY (n=121)	DEPT (n=33)	EDC (n=9)	RDB (n=5)
Grad Student	40.0%	29.0%	--	--
Undergrad	39.1%	35.5%	--	--

Staff	8.2%	16.1%	100%	100%
Faculty	6.6%	3.2%		
Other	8.0%	12.9%	--	--

Table 10.3. Distribution of survey respondents' roles. EDC and RDB were restricted access sites accessed only by staff.

I interviewed 36 users across the four deployments, with the majority of interviewees being users of the LIBRARY site (18), followed by DEPT (8), EDC (6), and RDB (4). My interviewees represented a similar proportion of roles as the survey respondents, with graduate students and staff constituting the majority. shows the breakdown of the interviewees based on their role. The graduate and undergraduate students came from a range of departments including Psychology, Drama, Communications, Biostatistics, Political Science, Computer Science, Information Science, among others. The staff were primarily from the School of Medicine and affiliated clinical research centers.

The survey data for the LIBRARY site showed that the majority of respondents were regular site users, with 70% reporting visiting site the daily or few times a week (Figure 10.2.a). To get a sense of how users normally found help on web sites, I asked respondents to indicate their preferred method of finding help. For the LIBRARY deployment, users' preferred form of help (Figure 10.2.b) differed significantly (Pearson $\chi^2_{(4,N=121)}=36.15$, $p<.0001$), with "trying on my own" and "using the site help" accounting for most responses (35.5% and 28.9%, respectively).

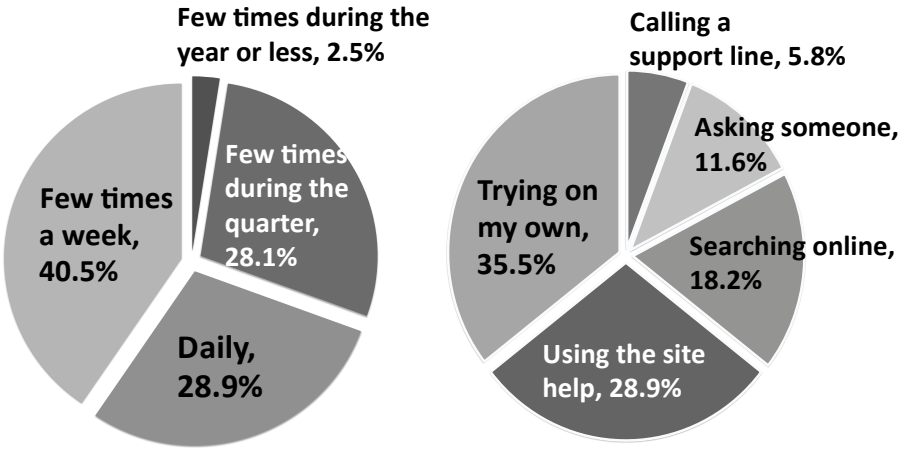


Figure 10.2. (a) Frequency of site use among LIBRARY users; (b) Preferred method of help seeking among LIBRARY users.

10.6 End users' perspectives on LemonAid

I present my main findings about end users' perceptions of LemonAid by combining quantitative analyses from logs and surveys and qualitative findings from interviews.

10.6.1 Helpfulness of LemonAid

To assess the helpfulness of LemonAid, I sought convergence in my three sources of data: (1) helpfulness of Q&A selections captured during actual use (Figure 10.1.3); (2) data collected from an exit survey which explicitly asked whether users found something helpful; and, (3) interview data that shed light on why or why not something was helpful.

Across all four deployment logs, I found that users selected "helpful" for over 73.2% of the viewed answers (LIBRARY=77.7%, DEPT=71.0%, EDC=75.1%, RDB=69.2%, Table 10.2). There is some noise in this log data—some of my interviewees indicated that they were simply browsing questions and not looking for help on a particular topic, but were forced to assess the helpfulness of the answer set anyway. Still, my exit survey data largely corroborates the log data: 70.6% of respondents on

average indicated having found something helpful during the use of LemonAid

(LIBRARY=71.1%, DEPT=65.6%, EDC=77.8%, RDB= 80.0%). The distribution of the survey responses from the LIBRARY site is shown in Figure 10.3.a. A majority of users felt LemonAid was helpful (Pearson $\chi^2_{(6,N=121)}=60.25$, $p<.0001$). Only 14.9% of respondents disagreed at some level that LemonAid was helpful, whereas 71.1% agreed at some level.

I also analyzed whether certain types of users were more likely to find LemonAid helpful. For the LIBRARY data, I found that respondents' frequency of site use was correlated significantly and negatively with whether they felt LemonAid was helpful (Spearman $\rho=-.23$, $N=121$, $p<.01$), meaning users who indicated visiting the site more frequently were less likely to find something helpful through LemonAid. My interview data corroborates this finding as the more frequent users indicated that they were already familiar with the site's main features and many of the existing help questions were "too basic" for them. In contrast, the interviews consistently revealed that new or less frequent site users found LemonAid to be helpful for learning about unfamiliar parts of the interface:

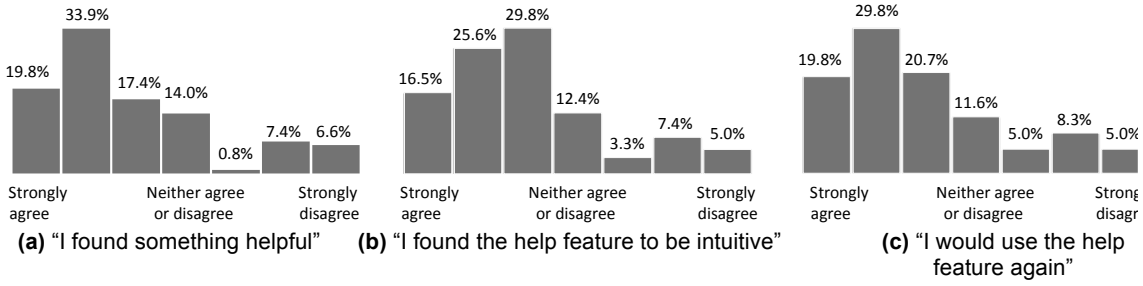


Figure 10.3. Distribution of LIBRARY users' survey responses.

> *Well, what I liked about it is that…you might find a question that you didn't even know you had, so I like that…I was just hovering and looking at stuff and somebody asked a question about audio books, and it reminded me, "oh yeah, I need to get an audio book." So I like that about it, so I think it's good in that when you don't have a*

specific question or you are just looking at what other people are asking, then it's really helpful. (p5)

	Agree	Neither Agree/Disagree	Disagree
Asking someone	64.3%	14.3%	21.4%
Calling a support line	85.7%	0.0%	14.3%
Searching online	77.3%	4.5%	18.2%
Trying on my own	58.1%	23.3%	18.6%
Using the site help	82.9%	11.4%	5.7%

Table 10.4. Cross tabulation of LIBRARY users' responses to "preferred method of help" vs. "*I found something helpful*"

I analyzed the relationship between users' preferred method for finding help and helpfulness of LemonAid for the LIBRARY data (shown in Table 10.4). Although this association was not significant (Spearman ρ=.06, N=121, p=.54), I did learn from the interviews that users who normally preferred to find help using the site's existing help through FAQs or documentation found LemonAid's contextual Q&A approach to be more helpful. Although Table 10.4 shows that users who normally preferred to find help by asking someone were less likely to find LemonAid helpful, this accounted for only about 12% of users overall who asked someone for help (Figure 10.2.b). Many of my interviewees felt that there was often a social cost to asking someone for help. LemonAid, in contrast, allowed users to learn from other people's experiences without having to bother anyone or lose face:

I think that students nowadays like to help themselves a lot more. In terms of figuring out a website, it can be kind of embarrassing to not know how to get around a website. So I think the nice thing about this [LemonAid] is that it's 24/7 and it doesn't require that stepping out of what maybe your comfort zone, you know. If English is your second language, or you're really new…and you're nervous walking up to the librarian at the library, or someone's just not available when you want a question answered, I think it [LemonAid] can be a real positive experience…(p16)

Still, many of my interviewees noted that in-person explanations were richer and allowed opportunity for instant back-and-forth than textual, asynchronous Q&A. Some

interviewees also revealed that even if they did not ask others for help, they were sometimes on the *other* side answering technical questions for less tech-savvy family members or friends. In these cases, the interviewees felt that hands-on help was the most preferred form of help (regardless of the available alternatives that improved help retrieval):

> *They really just want somebody to do it for them. They really don't want to learn, it they just want it done so they'd rather just call someone and have it talked through or done for them even though there's a help function that'll explain it step by step... [p19]*

Perhaps more visual, interactive, and synchronous forms of Q&A in future versions of LemonAid could make the content appear more "hands on" and be more helpful for such users.

10.6.2 Usability of LemonAid

Survey responses across *all* deployments indicated that on average 72.5% of users agreed to some extent that the LemonAid help feature was intuitive (LIBRARY=71.9%, DEPT=68.8%, EDC=75.0%, RDB=100.0%). The distribution of the responses from the library deployment is shown in Figure 10.3.b. Users expressed significant agreement as to LemonAid's intuitiveness (Pearson $\chi^2_{(6,N=121)}=44.74$, $p<.0001$), with a majority of respondents indicating that they felt LemonAid was intuitive (71.9%).

The interviews also revealed that most users found the help interface to be easy to understand even though it was their first time using this crowdsourced contextual help tool. One consistent theme was that users appreciated having direct access to help content without having to visit a separate interface, unlike most other help tools:

> *I liked the visual overlay. I think for a second I thought, "Whoa!" I'm not searching through a separate site to get to the help screen like [default help] where a different window pops up and you scroll through topics…I think that [LemonAid] is extremely*

intuitive. Having the help topics spread out by the areas in which I would need them as opposed to having a separate window open... (p6)

This finding is consistent with other studies that show that switching modes between an application and help often causes users to forget why help was requested in the first place and it takes them longer to return to their task (Clark, 1981). Users also liked that they could simply turn off the help mode when finished with their help-seeking task and could return back to the application. But, some users mentioned that if they were working on a complex, multi-step task, going back and forth in the help mode could be distracting and having keyboard short-cuts perhaps would be useful.

10.6.3 Potential Reuse of LemonAid

As my survey and interviews primarily probed the first use of LemonAid, I also wanted to know if users were likely to use it again. My survey results (Figure 10.3.c) showed that 72.3% of respondents indicated they would use LemonAid again (Pearson $\chi^2_{(6,N=121)}=44.74$, $p<.0001$): (LIBRARY=70.3%, DEPT=81.3%, EDC=66.7%, RDB=100.0%). Whether users found LemonAid helpful correlated significantly and positively with whether they felt they would reuse LemonAid again (Spearman $\rho=.63$, $N=121$, $p<.001$).

These results are a strong contrast to prior work that has shown that users are frustrated with help systems and fear clicking on help (Furnas et al., 1987; Rettig, 1991). In fact, about two-thirds of my interviewees said that they would like to see LemonAid even on other sites and desktop applications. For example, one interviewee recounted frustration with a site and its internal help tools she used recently, wishing that she could instead access help via LemonAid on that site:

You search [on the site] and it gives you three very different things. The only two hits have nothing to do with what I want. If there was a LemonAid type help for each section it would help… I could type in the keywords into LemonAid and see how

many other administrators across campus have the same question or other questions…that would be helpful! (p21)

10.6.4 Utility of LemonAid Compared to Other Forms of Help

In addition to getting insights about LemonAid's usage, my interviews also focused on understanding how LemonAid compared with modern help tools such as built-in help, FAQs, online searching, and discussion forums.

10.6.4.1 LemonAid Provides More Relevant Help Content

The majority of interviewees expressed frustration in using current forms of software help. Although a third of the users said they consulted built-in help, the vast majority of users completely avoided consulting help because they feared seeing "long pages of text." Although this behavior has been known for years (Rettig, 1991), countless applications still offer only textual built-in help. Users also felt that the built-in help content was often outdated, even though more updated content and tips were available through online channels:

Sometimes it feels that they [tech support] don't go through and update…even if they do set up 5 standard categories, it feels like they're not looking at other message boards to see the huge need for this other thing…they're not adapting and updating that (p23)

In contrast, users felt that the help content in LemonAid was more current and relevant since it could be added and edited at any point and could represent a wide range of actual problems rather than those anticipated by software designers. Furthermore, several users mentioned that the questions and answers in LemonAid were easier to understand than official built-in FAQs that often used a lot of system-specific jargon and were presented in a long list:

FAQ's are a pain because, first of all, they don't use the same language that a normal user would most of the time but rather an internal jargon. I tried LemonAid and I

could relate to the questions if I was a student or whoever. This pushes the popular questions out. An FAQ is more like an artificial taxonomy giving a lot of useless information. (p29)

10.6.4.2 LemonAid Reduces the Burden on Choosing Keywords

Although about 30% of my interviewees said they prefer to search when seeking help, they identified a number of problems with keyword-based searching. One common reason that users were not able to find relevant content was due to the mismatch between how users described their questions versus how other users or system designers may have answered the same questions (the classic "vocabulary problem" (Furnas et al., 1987)). The issue of choosing relevant keywords was particularly acute for non-Native English speakers.

As seen in Table 10.2, only a small percentage of users ended up using the built-in LemonAid search feature (e.g., there were 43 search queries in the LIBRARY deployment or about 4.4% of all help sessions). Although it is possible that some users may not have noticed this feature, many interviewees pointed out that LemonAid reduced the need to search altogether because they could find relevant questions and answers by clicking around in the interface:

When you're looking at something in the forums, you have to know what it's called...people can call it different things. With the kind of thing you're talking about, you have to know the terms then you may not find what you're looking for…With this [LemonAid], you're like, 'Oh, I don't know what's happening here!', click the Help button, click on what's confusing you and you don't have to worry about inputting any terms, or what it's called; I like that.. (p9)

This supports the key premise behind LemonAid's selection-based retrieval of relevant results, that users will be more able to choose relevant keywords and terms in the UI than to come up with keywords themselves to find help (as discussed in Chapter 8).

10.6.4.3 LemonAid Facilitates In-Context Social Learning

Another recurring theme in my interviews was that users appreciated the social aspects of retrieving help through LemonAid—for example, being able to browse through other users' questions, answers, and votes:

> *It's sometimes nice to hear what other people's experience is. Sometimes it can be kind of validating. If I'm confused about something, it's nice to know that other people have been similarly confused. (p6)*

Users also felt that the discovery of new information or tips through LemonAid was valuable because the content came from other users or staff and not a predefined document. For example, one user of the LIBRARY site explained:

> *I think [LemonAid] may save time if you are doing research or something like that so you have the ability to get input from other sources very quickly at your fingertips. To me, it's almost like a gold mine in that sense because it leads me into a direction that I may have not been able to get just going through traditional methods. (p14)*

Despite the advantages of social learning, some users also raised concerns about the authority and quality of answers posted on forums (and in LemonAid in the future):

> *You never know the technical level of people in the forum. So sometimes, there's great help…other times, there are people who barely figure out how to turn the thing on, and they're looking for very basic help on the topic, and it's very difficult to differentiate until you just read sometimes pages of content… (p3)*

Another user described the frustration of wading through repetitive questions and comments on forums:

> *…you have to sort through a lot of stuff to get there [answer in a fourm]…and there's repetition, so part of that depends on how well the forum is moderated and how well it's cleaned up so that repetitive answers or answers that are just slightly different but still the same aren't just clogging up the numbers of response... [p11]*

Although few users contributed questions and answers in my deployments, it is possible that problems that impede social learning on forums could affect LemonAid as well. However, as some users pointed out, because users invoked LemonAid from within the application rather than from a separate forum, people were perhaps more cautious about posting content because the help seemed "official," as it was overlaid on the application itself. LemonAid perhaps also succeeds in limiting the number of repetitive "I have the same question" comments because users tended to find relevant questions and could express the same sentiment by clicking on *me-too*, rather than posting a new redundant question. In fact, none of the new 16 questions posted on the LIBRARY site were duplicate questions and users voted "me-too" 63 times.

10.7 Software teams' perspectives on LemonAid

In addition to understanding end users' perspectives on LemonAid, I also wanted to gauge the reactions of the teams who deployed LemonAid. For each of the deployments, I interviewed the lead developer who handled the integration of LemonAid, and for the LIBRARY and DEPT deployments, I also interviewed one site administrator and two support staff (7 people in total).

10.7.1 Motivation for Integrating LemonAid

Although every software team that I worked with had different motivations for integrating LemonAid, a common theme was that with small teams, it was difficult to offer one-on-one support. Although the sites already offered some forms of built-in help, their developers felt that LemonAid could improve the discoverability of existing help, especially at locations in the interface where users were more likely to need it. Another motivation was curiosity about what kind of questions users would *want* to know answers to.

A more practical motivation was that the developers did not have to change the underlying code of their website:

What was appealing [about LemonAid] was that it was an overlay on top of it [the site] and since you're not requiring me to write any code, I think that's what sold it to me, I didn't have to write any code. It's really easy to do…(T02)

10.7.2 Deploying and Moderating LemonAid

Two teams were initially concerned about the possibility of spam, especially since the tool did not require a login. Despite the initial concerns, I found that spam was not a major issue during the deployments—only 5 questions were marked as spam across the four deployments over several weeks. Upon inspection, I found that 4 of these 5 questions were originally put in by staff and were not actually spam; 1 other question was an incomplete sentence that was marked by a staff member to flag it for removal.

Similar to the concerns of end users, administrators were also concerned about maintaining the quality and accuracy of answers. Furthermore, administrators also had concerns about the "public defacing" that was possible with LemonAid's content overlaid on the site:

[Help through LemonAid] is more on the site…and I really like that. The only thing is that…the information needs to be 100% correct and the people who are able to put in answers to the questions need to really make [sure] that it is [correct], you know, because of liability and lawsuits. (T05)

Another administrator mentioned that he once had to intervene in a discussion on a user mailing list when he sensed that users' answers were projecting incorrect information. He would consider his job in administering LemonAid to be no different:

It definitely takes the load off of the administrators to have users helping each other, right? But you do have to monitor and make sure that they are giving the correct information. I've already seen a few occasions where I had to clarify something on an email [in a mailing list] that some user sent out. (T03)

10.7.3 Utility of LemonAid Analytics Data

Teams had few formal methods for learning about users after deployment. The common theme was gathering basic usage analytics data through services such as Google Analytics. Although teams collected support questions through emails or phone calls, there was no systematic process that teams used to inform designs based on this data. This is consistent with studies discussed in Chapters 4 and 5 that show the disconnect between software support and usability engineering.

After asking teams about these existing practices, I showed them the LemonAid data for their site in the form of a basic dashboard (Figure 10.4), like the one found in web-based analytics tools. Most of the interviewees felt that they were not able to obtain this type of data from anywhere else and LemonAid would be a useful way to augment existing usage-based analytics services. For example, one software developer

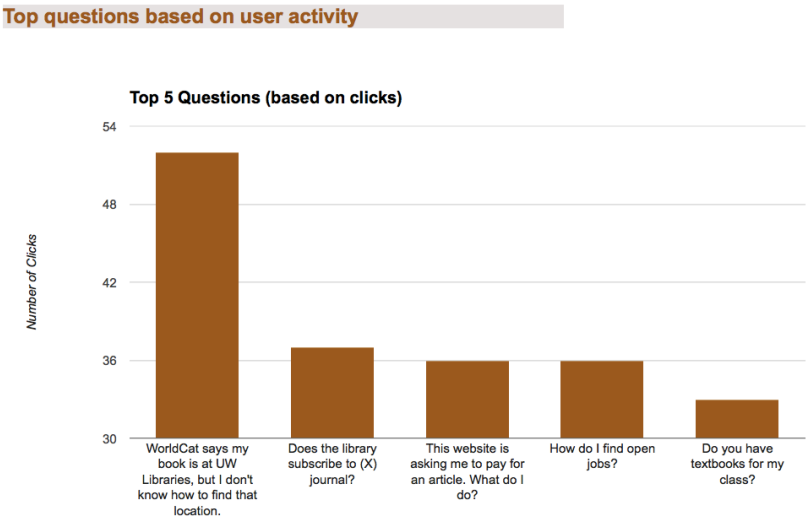


Figure 10.4: Partial view of analytics dashboard showing live deployment activity

pointed out that unlike LemonAid, data from tools such as Google Analytics did not reveal users' intentions in terms of why users were clicking on certain areas or spending more time on a particular page:

I think a harder thing to get at in regular analytics is what people actually struggle with…you can get the stuff that they like, you know the stuff that's popular or works well…but stuff that people are confused about, you don't get through regular analytics…and this LemonAid is useful because it's not just about numbers…it's showing the actual questions that people have…and phrased in a way that they [users] would post these questions, in their language…(T04)

Another administrator was particularly excited to see data about what users considered to be relevant questions based on where they clicked and what they voted on:

I think from the useful perspective, just seeing what questions people have on what elements and what are being asked. Because that kind of informs us where we might have some problems, lack of understanding, or where we can do some PR. And that's a lot of where we need some help…how do we make [the site] more accessible? (T07)

Although some team members were enthusiastic about using this data to argue for design and content changes, they also felt that it would be most useful over a longer period of deployment and with greater user participation.

10.8 Limitations

The primary goal of this field study was to investigate how users perceive the usability, helpfulness, and reuse value of crowdsourced contextual help in real tasks. There are several limitations that should be taken into account when interpreting the findings from this study. First, the field study method inherently lacks the control and precision that could be attained in a controlled setting (Yin, 1992). There is also a sampling bias given that all of my participants had some level of university education and my deployment sites were all hosted within one U.S. university. Some of the initial usage at each site was possibly influenced by a novelty effect as announcements were sent out to users to advertise the feature. Administrators for each site did point out, however, that the number of LemonAid-based help sessions overall were similar to the

number of help requests they receive on average. The way the interview data was collected and coded affects its interpretation and there may be alternative explanations.

10.9 Summary

This chapter presented results of a multi-site field deployment study of the LemonAid help system introduced in Chapter 8. The four deployment sites included the university's main library site, a departmental site, a clinical data capture application, and a personnel and grant management site. The deployment periods ranged from 7 to 15 weeks and over 1,200 logs, 168 exit surveys, and 36 interviews with end users were collected. This mixed-method study and analysis showed that LemonAid was helpful, intuitive, and desirable for reuse for over 70% of users across all the deployment sites and that users found LemonAid to be a refreshing approach compared to other modern forms of help.

While end users were overall positive about LemonAid, I found that most of the Q&A content came from the software teams rather than other users. For example, the logs of my largest deployment with a library site showed that only 16 new questions were added during the deployment, constituting about 1.6% of the total help sessions (972) over 15 weeks. Despite the seemingly low end user participation in the Q&A, the promising finding in the study of LemonAid was that users still derived benefit from using this crowdsourced contextual help approach. Even though software teams had to answer all the questions, team members felt that it was still saving them time compared to tackling users' one-on-one idiosyncratic issue reports (as discussed in Chapter 6 and 7). Also, as pointed out in Chapters 4 and 5, although teams are increasingly interested in using data to understand user behavior and usability issues, few tools allow them to get in-depth insights into users' intentions at a large scale. In this view, the teams in our study indicated that the initial analytics data aggregated by LemonAid was a novel way of understanding frequently asked questions and users' intentions in the context of the application.

CHAPTER 11

Conclusions and Future Work

The main goal of this dissertation has been to investigate how we can better understand and support users after software deployment. By using a design-based inquiry approach, this dissertation demonstrates that a selection-based crowdsourced contextual help system (Chapters 8,9) that allows users to find questions and answers from other users and support staff can be helpful, intuitive, and desirable for reuse for end users and can provide new insights to software teams about frequently asked questions (Chapter 10). In addition, this dissertation draws upon interpretivist approaches to characterize the modern context of software development and tradeoffs in design (Chapters 4), the role of usability after deployment (Chapter 5), and software support and issue reporting practices from the perspective of software teams (Chapter 6) and end users (Chapter 7).

At the pace of innovation in computing systems, we can expect software support to continue to play a key role in facilitating post-deployment user experiences. Although there may not be a single way to alleviate the bottlenecks in finding relevant help and resolving software issues, this dissertation lays the foundation upon which designers of software support tools, support professionals, and HCI researchers can begin to address some ways of improving interactions in software support. In addition, this work opens up new opportunities for learning about users after software deployment and integrating this knowledge with ongoing usability and design activities.

When possible, I have discussed the key implications and limitations of findings within each chapter of this dissertation. I now turn to reflect on some of the main ideas and findings that have emerged from this research. I also discuss some ideas for future research directions that can address some of the limitations of this work or can build upon

this dissertation's insights to explore new problems. Lastly, I restate the major contributions that this dissertation makes to the field of HCI and close with final remarks.

11.1 Reflections and Insights

In this section, I reflect on several insights that I have gained through this dissertation about the design and implementation of LemonAid and the larger context of crowdsourcing questions and answers. I also reflect on the methodological approaches adopted in this work.

11.1.1 Design and Implementation of LemonAid

The results presented in this dissertation, and the larger vision underlying LemonAid's approach to crowdsourced contextual help, raises some issues around scope, scalability, robustness, and privacy.

Problem Scope: For a given application, users may have a range of feature-related or account-specific troubleshooting help needs. Since LemonAid is integrated within the UI of the application, its primary strength is likely to be in providing user interface related help. For other types of issues that reach beyond the user interface, such as a problem with a blocked account or an issue with a credit card transaction, LemonAid would be able to inform a user that it is necessary to contact support, but it will not be able to help the user address their problem directly. Help needs that require the intervention of support personnel are less a limitation of LemonAid and more a limitation of crowdsourced help approaches in general.

Scalability: As shown in Chapter 8 (Figure 8.7), there is some initial indication that the retrieval algorithm is relatively stable as a help corpus increases in size. However, another important question is how LemonAid's retrieval scales for applications that vary from a narrow to a wide range of features and corresponding UI literals. For instance, in my study I observed was different users consistently made the same selection in the UI for the same problem, but made different selections for different types of problems. Thus, for

an application that has a large number of features (and more possibilities for selections), the spread of questions could be sparse. For the case of an application with only a few features, there will likely be similarly few possible selections. We can predict that LemonAid's performance will still degrade slowly as there would possibly be fewer questions about applications that have more limited functionality.

Another case I observed in my evaluation was the same label being used as an anchor for many different problems. For example, the "settings" label of *Google Calendar* was a particularly common choice when users perceived no better label in some of the scenarios. The retrieval algorithm was not able to retrieve a relevant answer in the top few results based on the selection alone. In this situation, the user would need to provide keywords to pare down the results. Still, in the worst case, LemonAid only degrades to the performance of a full-text search, but within the limited scope of questions generated through the LemonAid interface, rather than everything on the web.

Robustness: One concern about the utility of LemonAid in practice might be that web applications are constantly changing; anchoring help to rapidly changing labels and layouts may not be robust to such change. The user interface labels that LemonAid relies on, however, are likely to change less often than the average website content, since changing functionality labels often requires costly user re-education. Moreover, when functionality labels and locations do change, it would actually make sense for the help associated with those UI literals to be deprecated. With LemonAid, this would be automatic, since questions attached to labels that have been removed would no longer be matched. The only process required to keep help content current would be to refresh LemonAid's list of application-specific UI literals, which is a simple matter of re-extracting string literals from their source)

Privacy: By using text on web pages, much of which may be privacy-sensitive, LemonAid also raises some privacy concerns. However, since I am only extracting UI

literals from source code, and users can only select labels that match these static labels, user-generated content is never captured as part of a help request. There is a possibility that there may be some overlap between a UI literal and user-generated text. Future versions of LemonAid could allow users to redact details from their selections before submission.

11.1.2 Challenges and Opportunities for Crowdsourcing Q&A

A major component of this thesis was the field study presented in Chapter 10. I use the study's findings to discuss a number of implications and points of reflection both from the perspective of end users and software teams.

11.1.2.1 Users' Perceptions of the Social Aspects of Help

My field study indicated that users overall appreciated the social aspects of retrieving help through LemonAid. For example, as users browsed through other users' questions and saw other votes on questions, it was a validating experience for them to know that they were not alone in experiencing particular issues. Users also felt that the serendipitous discovery of new information about the application or tips through the contextual Q&A was also useful when they were not looking for any particular answers. The help content in LemonAid was also noted as being valuable because the Q&A came from other people using the application and not "jargon" from a predefined help document (a limitation of other forms of contextual help approaches).

11.1.2.2 Community Participation in the Larger Context

While end users were overall positive about LemonAid, I found that most of the Q&A content came from the software teams rather than other users. For example, the logs of my largest deployment with a library site showed that only 16 new questions were added during the deployment, constituting about 1.6% of the total help sessions (972) over 15 weeks. I also found that no end users answered a question; library staff answered all new questions. (I did find that the 16 new questions asked by users

received 121 views, accounting for about 21.5% of all question views and 74.3% of the corresponding answers were marked as helpful.)

Although it seems that few users contributed questions and answers in my deployments, prior work has shown that this level of activity is typical of what occurs in technical forums (Ko & Chilana, 2010; Lakhani & Von Hippel, 2003; Matejka et al., 2011). Similar low end user participation has also been observed in other community-based systems, such as *AnswerGarden* (Ackerman, 1998). Also, this level of participation is characteristic of communities that exist more broadly on the Internet (e.g., "*the 1% rule*" (Arthur, 2006)) where most users are consumers of online content rather than contributors. Still, my interviews revealed that users were even more cautious about posting content in LemonAid because the Q&A overlaid on the application's interface seemed "more official" (versus a separate forum or social networking site). This challenge is perhaps unique to crowdsourced contextual help systems given that the Q&A are accessed from within the application.

In future work, it would be interesting to compare the level of participation that in the current set of deployments to: 1) sites that have perhaps millions of active users; and 2) sites that use a closed familiar social network to crowdsource Q&A. There is an opportunity in teasing out whether there are differences in a small community-based Q&A forum (e.g., my deployment sites that had a few hundred or thousands of users) vs. a larger crowdsourced Q&A forum (e.g., sites such as *YouTube* that draw millions of daily users) vs. a closed social-network Q&A forum (i.e., sites that connect to only Facebook or Twitter friends). My hunch is that we would see some differences in the level of activity, but in any of these cases, there would still be need to incentivize users to participate on a regular basis. For example, one could consider incentives such as badges, awards and leaderboards that help make forums such as *Stack Overflow* successful (Mamykina et al., 2011). In fact, how to incentivize users in social computing systems is a broader question

being pursued by a number of researchers (see summary in Scekic et al., 2013).

11.1.2.3 The Role of Software Teams in Moderating Q&A

Despite the seemingly low end user participation in the Q&A, the promising finding in the study of LemonAid was that users still derived benefit from using this crowdsourced contextual help approach. Users could still find the content valuable perhaps because the host software teams were actively involved in maintaining the Q&A. For example, the host teams were able to devote time and resources to seed the initial database with existing FAQs, monitor the questions as they were entered, and provide answers. It may be that to sustain the same level of quality in answers, a long-term commitment from the host teams would be necessary. Since many modern organizations have already opted to create peer-to-peer support forums, perhaps engaging with users through crowdsourced contextual help is a natural extension. Also, users noted that when a staff member provided an answer, they were more likely to trust the authority and quality of that answer.

Unlike other forms of social Q&A, I feel that moderation by software teams can be integral to crowdsourced contextual help rather than seen as an extra feature. For example, the team members who participated in my study felt that investing in one-to-many support is more efficient and provides greater cost-savings in the long run compared to supporting users one-on-one. So, if team members can be alerted of new questions posted by users and teams can provide timely answers, it can be mutually beneficial for teams and a large number of users. Also, since LemonAid appeared to be "a part of" the application, maintaining the quality and accuracy of the help content was more critical for the host teams and team members appreciated having the additional control over the Q&A.

11.1.3 Methodological Insights

As mentioned earlier, I adopted a design-based inquiry approach in this research to

develop LemonAid, going through iterative rounds of design and evaluation. Although design-based inquiry approaches are common in HCI literature, many systems researchers in HCI face challenges in using these approaches for designing large-scale systems. Researchers are realizing that making novel technical contributions in the space of social computing is not enough—a sound understanding of how novel systems are actually used or perceived in real settings is just as important, but full evaluations are currently lacking in the literature (Bernstein et al., 2011). There has been a lot of interest in discussing what constitutes as an appropriate evaluation given the scale and nature of social computing systems where controlled conditions are difficult to devise.

I argue that the field evaluation approach (Chapter 10) that I used to assess the ecological validity of LemonAid's design in this dissertation offers many advantages. For example, since this type of evaluation occurs in the user's natural environment where the context is realistic, researchers can collect long-term usage data (Krathwohl, 1993; McGrath, 1995) and obtain a rich long-term perspective on how a system behaves and keeps up in actual use. For the domain of software help, it is particularly important to consider this approach because it allows researchers to get data from users' natural breakdowns and help-seeking contexts.

While powerful, the disadvantages of field evaluations are their high cost in terms of time, upfront investment, and commitment from field sites (Krathwohl, 1993; McGrath, 1995). I had to personally spend months of engineering effort in getting the LemonAid prototype to be robust enough to be deployed on different web applications and to be used by end users using different browsers and platforms. Tackling the interoperability issues among different platforms was particularly challenging and time-consuming. In addition to the technical hurdles, I also had to use social capital to make contact with prospective software teams who would be willing to deploy LemonAid on their sites.

Most of the time, these teams were already overburdened with deadlines and milestones and did not want to add yet another to-do item on their busy development schedules. Still, I feel fortunate that I was able to get the support of four software teams who could see the potential value of this selection-based crowdsourced help retrieval approach and were willing to give it a try.

While the high upfront costs may appear as bottlenecks and may deter many researchers from doing field evaluations, this experience of doing live deployments has given me a new perspective on how to better understand and design for end users. In the context of HCI research, it has also reinforced the importance of taking into account social and organizational factors of design when we engage in design-based inquiry to propose novels solutions. I hope that other social computing researchers will use this study as one model for going into the field and assessing the ecological validity of their systems.

11.2 Future Work

Although there are many opportunities for expanding LemonAid to make it functional on mobile devices and desktop applications, these are primarily engineering endeavors. There also are other opportunities for improving incentives for community participation in systems like LemonAid, but there already is a body of work tackling the question of incentives in social computing systems more broadly (see Scekic et al., 2013).

I focus here on potential fruitful research directions that build upon findings from studies and design ideas explored in this dissertation. Key themes that I discuss are: 1) facilitating software learning through crowdsourced contextual help retrieval; 2) bridging the disconnect between usability practice and software support; 3) designing software analytics tools for understanding users' intentions; 4) expanding crowdsourced contextual help retrieval beyond software troubleshooting.

11.2.1 Facilitating Software Learning through Crowdsourced

Contextual Help Retrieval

As discussed in Chapter 10, in addition to helping users find help content, LemonAid may also provide learning benefits. For example, it allows for the serendipitous discovery of application features, shortcuts, and customizations through the contextual Q&A discussions. An interesting finding from my field study of LemonAid was that users could use LemonAid as a learning tool as they browsed through the goals of other users who have used the site by simply clicking on different labels in the UI.

Even though there is initial indication that LemonAid can provide potential learning benefits, there is a large space of software learnability (Grossman et al., 2009) and learning theories (summarized in Chapter 3) that this dissertation does not explore in depth. How do we actually know that viewing a relevant answer necessarily resulted in learning? I believe that this is the next big open research question that should be explored in the design of help systems so that long-term learning can be fostered as part of the natural help-seeking process.

To have an orientation towards learning, several improvements are potentially needed to the selection-based crowdsourced contextual help approach introduced in this dissertation:

- **Improving capture of context**: Currently, LemonAid is primarily utilizing attributes of the HTML DOM to capture context. Future research can investigate more sophisticated ways of capturing program execution information and operations history to augment contextual data. For example, it would be interesting to explore ways of capturing the path that a user takes to arrive at a particular selection in the UI to seek help and to see how this information can be used to improve the relevance of retrieved Q&A.
- **Allowing for visual help retrieval:** Although the current version of LemonAid

can be integrated in any web application by scraping the keywords in the interface, there is an opportunity to expand this concept to support more visual information retrieval. For example, many participants in my study indicated that they would like to see LemonAid on more complex applications such as *Photoshop* as those involve more complex interaction scenarios. In such visual applications, it is likely that the vocabulary problem is more acute since highly specialized terms are used to name commands and controls. An interesting research direction in this space could perhaps be exploring the space of visual contextual retrieval where the users' selections that trigger implicit queries are visual elements instead of keywords. Some initial work by Yeh et al. (2011) explores the use of screenshots for retrieval, but there is merit in expanding this space and exploring designs for context-rich visual selections.

- **Accommodating version changes:** Given the pace at which software applications (particularly web-based) evolve today, developing help that adapts to different versions of applications is critical. This is particularly important in the context of learning because users often perceive version changes as a setback and are forced to relearn how to navigate a familiar application (Nielsen, 1993). It would be interesting to explore automated approaches for adapting an existing help repository to a new version not just for the purpose of keeping the help content relevant, but also for reducing the learning burden on users in adapting to a new version.
- **Help for highly personalized applications**: Prior work suggests that increasingly applications are offering opportunities to tailor or personalize applications (Mackay, 1991; Mørch, 1997). However, end users do not always make use of these advanced opportunities because they do not even know what is possible or where

to even begin. Systems that explore the use of community-based command recommendations (*i.e.*, Matejka et al., 2009) are one step in this direction, allowing users to learn from other users' application usage. However, beyond commands, it could be worth exploring how knowledge about customized workflows could be crowdsourced and exchanged in the context of the application.

11.2.2 Bridging the disconnect between usability practice and software support

Beyond improving software support, this dissertation calls for organizations to have increased focus on usability throughout the development lifecycle. My post-deployment usability studies suggest that currently the role of usability appears to diminish after software is deployed and usability professionals are rarely involved in postulated post-deployment activities such as usage logs analysis, support log analysis, and in situ usability testing (Chapters 4 and 5). In future work, there is much opportunity in helping organizations be more engaged in post-deployment usability activities. Parallel to the subfield of software maintenance where the focus is on how to maintain systems after they are deployed, I have proposed the idea of "*usability maintenance*" (Chilana et al., 2011b) within the field of human-computer interaction. The goal of usability maintenance should be to enhance post-deployment user experience based on information about the actual use of a product and provide ongoing support for usability principles of learnability, efficiency, memorability, recovery from errors, and satisfaction. My research suggests that there is a large open space to study and invent opportunities for support teams to interact with usability practitioners and share information about user behavior. Advanced analytics tools that can provide insights into users interactions at a large scale after deployment may be one step in this direction.

11.2.3 Designing software analytics tools for understanding user intent

While tools such as *Google Analytics* give a rich overview of users' activity within an application, software teams often have a hard time understanding users' intentions and "why" they did something in a particular way. As discussed above, one component of my LemonAid system was a "dashboard" that aggregated users' activities in the help mode and revealed users' top questions. In future work, there is potential in expanding on these data analytics features and working closely with software teams to investigate which aspect of this aggregated "big data" is actually useful. I believe that there is merit in using more mixed-method approaches for finding meaning in different facets of data, including usage logs and issue reports, for uncovering users' intentions and problems with using software. Apart from using quantitative and visualization techniques to explore data on users' intentions, new methods are needed to discover hidden relationships, such as the content analysis of software problems I presented in Chapter 7.

11.2.4 Expanding Selection-Based Crowdsourced Retrieval to Other Domains

Although the idea of selection-based crowdsourced help retrieval is a novel contribution of this dissertation, several other works have also recently been exploring the broader concept of combining crowdsourcing and information retrieval (Lease & Yilmaz, 2013). Others have also explored the space of *constrained natural language queries* (Lamberti et al., 1994) to tackle the variation in users' natural language queries. While many possibilities exist for expanding LemonAid's idea of selection-based crowdsourced retrieval to other domains, I believe this approach can be most beneficial in complex vocabulary-rich domains. For example, one area that I think can benefit the most would be crowdsourced medical information retrieval. A recent Pew Internet

study[17] shows that as many as 72% of internet users search for health information online, often ending up in crowdsourced health forums (Zeng et al., 2002) that suffer from a mismatch in vocabularies used by patients and health care providers. What would a selection-based crowdsourced retrieval approach look like in this domain? Since the medical context lacks the kinds of program information that can be captured from software applications, how do we even begin to define contextual data? These are some of the challenges that must be tackled to realize the potential of selection-based crowdsourced retrieval beyond the domain of software help.

11.3 Summary of Contributions

The different contributions made by this dissertation are broken down by themes:

11.3.1 Concepts

- Synthesis and conceptualization of theoretical perspectives on software help-seeking. (Chapter 3)
- A new selection-based contextual help approach that allows users to find help, and ask and answer questions within the user interface of an application. (Chapter 8)

11.3.2 Study Results

- Empirical findings that shed light on user-centered design practices in a modern software development context. (Chapter 4)
- Empirical findings that establish the current state of post-deployment usability practices. (Chapter 5)
- Empirical findings that illustrate how web-based support is practiced and how support issues are diagnosed and resolved using modern formats. (Chapter 6)
- Empirical findings that provide a conceptualization for understanding unwanted software behaviors that are reported online. (Chapter 7)

[17] http://www.pewinternet.org/Commentary/2011/November/Pew-Internet-Health.aspx

- A retrieval evaluation that demonstrates the feasibility of the selection-based contextual help approach for a crowdsourced corpus of selections. (Chapter 9)
- Empirical findings that demonstrate that the selection-based contextual help system can be helpful, intuitive, and desirable for reuse for end users. (Chapter 10)
- Empirical findings that show that the analytics data gathered through the selection-based contextual help approach can provide new insights to software teams about frequently asked questions. (Chapter 10)

11.3.3 Technology

- An interactive system that embodies the selection-based crowdsourced contextual help approach. (Chapter 8, 10)
- A new help retrieval algorithm that leverages contextual information and user interface selections to retrieve relevant help content. (Chapter 8)
- A flexible architecture for embedding the new selection-based contextual help system in any web application. (Chapter 8)

11.4 Final Remarks

We are now creating and capturing more digital information than ever before through computers, mobile devices, sensors and other emerging technologies. However, in this era of "information explosion," millions of users battle with software user interfaces and navigation structures to find the information that they actually need. This dissertation combines methods and perspectives from human-computer interaction (HCI) and information science to present a new selection-based crowdsourced contextual help retrieval approach that allows end users to find questions and answers from other users and support staff within an application's interface and provides new analytics insights to software teams about users' questions. While the focus of this dissertation has been to design and evaluate a new help system, this research also

reveals the social and organizational aspects of how modern software systems are designed and deployed, and how organizations can better achieve user-centered design. In particular, this dissertation make an argument for adopting an orientation towards *usability maintenance* in organizations and bridging the disconnect (Figure 11.1) between help and support and usability activities.

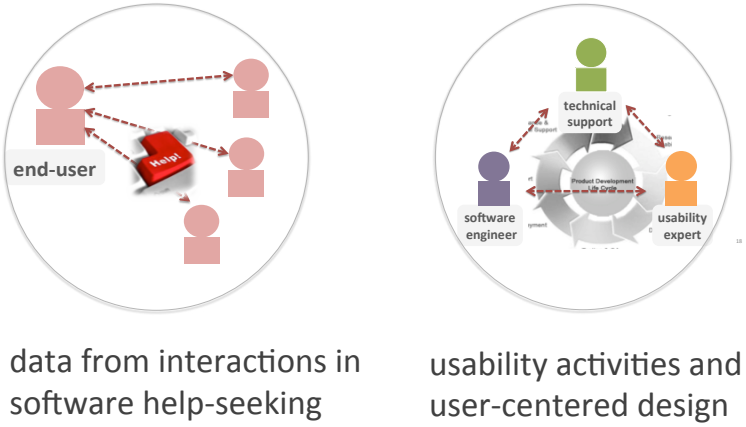


Figure 11.1: Bridging the disconnect between data from interactions in software help-seeking and usability and UCD activities

REFERENCES

Ackerman, M. S. (1998). Augmenting organizational memory: a field study of answer garden. *ACM Trans. Inf. Syst.*, *16*(3), 203–224.

Ackerman, M. S., & Malone, T. W. (1990). Answer Garden: a tool for growing organizational memory. *Proceedings of the ACM SIGOIS and IEEE CS TC-OA conference on Office information systems* (pp. 31–39). Cambridge, Massachusetts, United States: ACM.

Adar, E., Teevan, J., Dumais, S. T., & Elsas, J. L. (2009). The Web changes everything: Understanding the dynamics of Web content. *Proceedings of the Second ACM International Conference on Web Search and Data Mining* (pp. 282–291). ACM.

Ames, A. L. (2001). Just what they need, just when they need it: an introduction to embedded assistance. *Proceedings of the 19th annual international conference on Computer documentation* (pp. 111–115). ACM New York, NY, USA.

Anderson, J. R., Boyle, C. F., Farrell, R., Reiser, B. J., & PSYCHOLOGY, C.-M. U. P. P. D. O. (1984). *Cognitive principles in the design of computer tutors*. Dept. of Psychology, Carnegie-Mellon University.

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., & Guéhéneuc, Y. (2008). Is it a bug or an enhancement?: a text-based approach to classify change requests. ACM New York, NY, USA.

Anvik, J., Hiew, L., & Murphy, G. (2006). Who should fix this bug? (pp. 361–370). ACM New York, NY, USA.

Aranda, J., & Venolia, G. (2009). The secret life of bugs: Going past the errors and omissions in software repositories. *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering* (pp. 298–308). IEEE Computer Society.

Arthur, C. (2006, July 20). What is the 1% rule? *The Guardian*.

Association of Support Professionals. (2012). *The Year's Ten Best Web Support Sites*.

Bates, M. J. (1993). The design of browsing and berrypicking techniques for the online search interface. *Online Information Review*, *13*(5), 407–424.

Belkin, N. J. (2000). Helping people find what they don't know. *Commun. ACM*, *43*(8), 58–61.

Bernstein, M. S., Ackerman, M. S., Chi, E. H., & Miller, R. C. (2011). The trouble with social computing systems research. *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems* (pp. 389–398). ACM.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008). What makes a good bug report? (pp. 308–318). ACM New York, NY, USA.

Bias, R. G., & Mayhew, D. J. (Eds.). (1994). *Cost Justifying Usability*. Boston, MA: Academic Press, Inc.

Bias, R. G., & Mayhew, D. J. (2005). *Cost-Justifying Usability: An Update for the Internet Age*. Morgan Kaufmann Publishers Inc.

Bolin, M., Webber, M., Rha, P., Wilson, T., & Miller, R. C. (2005). Automation and customization of rendered web pages. *Proceedings of the 18th annual ACM symposium on User interface software and technology* (pp. 163–172). ACM.

Brandt, J., Dontcheva, M., Weskamp, M., & Klemmer, S. R. (2010). Example-centric programming: integrating web search into the development environment. *Proceedings of the 28th international conference on Human factors in computing systems* (pp. 513–522). ACM.

Breu, S., Premraj, R., Sillito, J., & Zimmermann, T. (2010). Information needs in bug reports: improving cooperation between developers and users. *Proceedings of the 2010 ACM conference on Computer supported cooperative work* (pp. 301–310). Savannah, Georgia, USA: ACM.

Brown, B., Reeves, S., & Sherwood, S. (2011). Into the wild: challenges and opportunities for field trial methods. *Proceedings of the 2011 annual conference on Human factors in computing systems* (pp. 1657–1666). Vancouver, BC, Canada: ACM.

Carroll, J. M. (1998). *Minimalism beyond the Nurnberg funnel*. The MIT Press.

Carroll, J. M., & Rosson, M. B. (1987). *Paradox of the active user.* The MIT Press.

Carroll, J. M., Smith-Kerker, P. L., Ford, J. R., & Mazur-Rimetz, S. A. (1987). The minimal manual. *Human-Computer Interaction*, *3*(2), 123–153.

Chamberlain, S., Sharp, H., & Maiden, N. (2006). Towards a framework for integrating agile development and user-centred design. *Extreme Programming and Agile Processes in Software Engineering*, 143–153.

Chamberland, L. (1999). Componentization of HTML-based online help. *Proceedings of the 17th annual international conference on Computer documentation* (pp. 165–168). ACM.

Chilana, P. K., Grossman, T., & Fitzmaurice, G. (2011a). Modern software product support processes and the usage of multimedia formats. *Proceedings of the 2011 annual conference on Human factors in computing systems* (pp. 3093–3102). Vancouver, BC, Canada: ACM.

Chilana, P. K., Holsberry, C., Oliveira, F., & Ko, A. J. (2012a). Designing for a Bilion Users: A Case Study of Facebook. *CHI 2012 Case Studies Track (To appear)*.

Chilana, P. K., Ko, A. J., & Wobbrock, J. O. (2010). Understanding Expressions of Unwanted Behaviors in Open Bug Reporting. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '10)* (pp. 203–206). Washington, D.C.: IEEE Computer Society.

Chilana, P. K., Ko, A. J., Wobbrock, J. O., & Grossman, T. (2013). A multi-site field study of crowdsourced contextual help: usage and perspectives of end users and software teams. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 217–226). Paris, France: ACM.

Chilana, P. K., Ko, A. J., Wobbrock, J. O., Grossman, T., & Fitzmaurice, G. (2011b). Post-deployment usability: a survey of current practices. *Proceedings of the*

2011 annual conference on Human factors in computing systems (pp. 2243–2246). Vancouver, BC, Canada: ACM.

Chilana, P., Ko, A. J., & Wobbrock, J. O. (2012b). LemonAid: selection-based crowdsourced contextual help for web applications. *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems* (pp. 1549–1558). ACM.

Chilana, P., Wobbrock, J., & Ko, A. (2009). Usability Practices in Complex Domains: Implications for Training the Next Generation of Usability Professionals. *Submitted for Review.*

Choo, C. W., Detlor, B., & Turnbull, D. (1999). Information Seeking on the Web--An Integrated Model of Browsing and Searching. *Proceedings of the ASIS Annual Meeting* (Vol. 36, pp. 3–16).

Clark, H. H., & Brennan, S. E. (1991). Grounding in communication. *Perspectives on socially shared cognition*, *13*, 127–149.

Clark, I. A. (1981). Software simulation as a tool for usable product design. *IBM Systems Journal*, *20*(3), 272–293.

Cockburn, A. (2006). *Agile software development: the cooperative game (agile software development series)*. Addison-Wesley Professional.

Comeau, J. M., & Milton, P. R. (1993). A window-based help, tutorial and documentation system. *Proceedings of the 11th annual international conference on Systems documentation* (pp. 71–81). ACM.

Covi, L. M., & Ackerman, M. S. (1995). Such easy-to-use systems!: How organizations shape the design and use of online help systems. *Proceedings of conference on Organizational computing systems* (pp. 280–288). ACM New York, NY, USA.

Crabtree, A., O'Neill, J., Tolmie, P., Castellani, S., Colombino, T., & Grasso, A. (2006). The practical indispensability of articulation work to immediate and remote help-giving. *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (p. 228). ACM.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Commun. ACM*, *31*(11), 1268–1287.

Dalle, J., den Besten, M., & Masmoudi, H. (2008). Channeling Firefox Developers: Mom and Dad Aren't Happy Yet. *Open Source Development, Communities and Quality* (pp. 265–271). Boston: Springer.

Dan R. Olsen, J. (2007). Evaluating user interface systems research. Proceedings of the 20th annual ACM symposium on User interface software and technology. doi:http://doi.acm.org/10.1145/1294211.1294256

Das, A. (2003). Knowledge and productivity in technical support work. *Management Science*, *49*(4), 416–431.

Delisle, S., & Moulin, B. (2002). User interfaces and help systems: from helplessness to intelligent assistance. *Artificial Intelligence Review*, *18*(2), 117–157.

Detweiler, M. (2007). Managing UCD within agile projects. *interactions*, *14*(3), 40–42.

Downs, J. S., Holbrook, M. B., Sheng, S., & Cranor, L. F. (2010). Are your participants gaming the system?: screening mechanical turk workers. *Proceedings of the 28th international conference on Human factors in computing systems* (pp. 2399–2402). Atlanta, Georgia, USA: ACM.

Eales, R. T., & Welsh, J. (1995). Design for collaborative learnability. *The first international conference on Computer support for collaborative learning* (pp. 99–106). L. Erlbaum Associates Inc.

Farkas, D. K. (1993). The role of balloon help. *ACM SIGDOC Asterisk Journal of Computer Documentation*, *17*(2), 3–19.

Finin, T. W. (1983). Providing help and advice in task oriented systems. *Proceedings of the Eighth international joint conference on Artificial intelligence - Volume 1* (pp. 176–178). Karlsruhe, West Germany: Morgan Kaufmann Publishers Inc.

Flanagan, J. (1954). The critical incident technique. *Psychological bulletin*, *51*(4), 327–358.

Frayling, C. (1993). *Research in art and design*. Royal College of Art London.

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

Fussell, S. R., Kraut, R. E., & Siegel, J. (2000). Coordination of communication: effects of shared visual context on collaborative work. *Proceedings of the*

2000 ACM conference on Computer supported cooperative work (pp. 21–30). Philadelphia, Pennsylvania, United States: ACM.

Glaser, B. G. (1992). *Basics of grounded theory analysis: emergence vs forcing*. Sociology Press Mill Valley, CA.

Glerum, K., Kinshumann, K., Greenberg, S., Aul, G., Orgovan, V., Nichols, G., Grant, D., et al. (2009). Debugging in the (Very) Large: Ten Years of Implementation and Experience. SOSP.

Goodall, S. D. (1992). Online help: a part of documentation. *Proceedings of the 10th annual international conference on Systems documentation* (pp. 169–174). ACM.

Google Calendar Forums. (n.d.). Retrieved from http://www.google.com/support/forum/p/Calendar/

Gould, J., & Lewis, C. (1985). Designing for usability: key principles and what designers think. *Communications of the ACM*, *28*(3), 300–311.

Grayling, T. (1998). Fear and Loathing of the Help Menu: A Usability Test of Online Help. *Technical Communication: Journal of the Society for Technical Communication*, *45*(2), 168–79.

Greene, J. C., & Caracelli, V. J. (1997). Defining and describing the paradigm issue in mixed-method evaluation. *New Directions for Evaluation*, *1997*(74), 5–17. doi:10.1002/ev.1068

Grossman, T., & Fitzmaurice, G. (2010). Toolclips: An investigation of contextual video assistance for functionality understanding. *Proceedings of the 28th*

international conference on Human factors in computing systems (pp. 1515–1524). ACM.

Grossman, T., Fitzmaurice, G., & Attar, R. (2009). A survey of software learnability: metrics, methodologies and guidelines. *Proceedings of the 27th international conference on Human factors in computing systems* (pp. 649–658). ACM.

Grossman, Tovi, Matejka, Justin, & Fitzmaurice, George. (2010). Chronicle: Capture, Exploration, and Playback of Document Workflow Histories. *ACM Symposium on User Interface Software & Technology*.

Grudin, J. (1988). Why CSCW applications fail: problems in the design and evaluationof organizational interfaces. *Proceedings of the 1988 ACM conference on Computer-supported cooperative work* (pp. 85–93). Portland, Oregon, United States: ACM.

Gulliksen, J., Boivie, I., & Göransson, B. (2006). Usability professionals—current practices and future development. *Interacting with Computers*, *18*(4), 568–600.

Gulliksen, J., Boivie, I., Persson, J., Hektor, A., & Herulf, L. (2004). Making a difference: a survey of the usability profession in Sweden. *Proceedings of the third Nordic conference on Human-computer interaction* (pp. 207–215). Tampere, Finland: ACM.

Gunther, R., Janis, J., & Butler, S. (2001). The UCD Decision Matrix: How, When, and Where to Sell User-Centered Design into the Development Cycle. *Retrieved*, *9*(6), 2006.

Halverson, C. A., Erickson, T., & Ackerman, M. S. (2004). Behind the help desk: evolution of a knowledge management system in a large organization. *Proceedings of the 2004 ACM conference on Computer supported cooperative work* (pp. 304–313). Chicago, Illinois, USA: ACM.

Hammond, N., J\&\#248;rgensen, A., MacLean, A., Barnard, P., & Long, J. (1983). Design practice and interface usability: Evidence from interviews with designers. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (pp. 40–44). Boston, Massachusetts, United States: ACM.

Harper, F. M., Raban, D., Rafaeli, S., & Konstan, J. A. (2008). Predictors of answer quality in online Q\&A sites. *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems* (pp. 865–874). Florence, Italy: ACM.

Harrison, S. M. (1995). A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 82–89). Denver, Colorado, United States: ACM Press/Addison-Wesley Publishing Co.

Hartmann, B., MacDougall, D., Brandt, J., & Klemmer, S. R. (2010). What would other programmers do: suggesting solutions to error messages. *Proceedings of the 28th international conference on Human factors in computing systems* (pp. 1019–1028). ACM.

Hartson, H. R., & Castillo, J. C. (1998). Remote evaluation for post-deployment usability improvement. *Proceedings of the working conference on Advanced visual interfaces* (pp. 22–29). ACM New York, NY, USA.

Hastie, H. W., Johnston, M., & Ehlen, P. (2002). Context-Sensitive Help for Multimodal Dialogue. *Proceedings of the 4th IEEE International Conference on Multimodal Interfaces - Volume 00* (p. 93). IEEE Computer Society.

Heidegger, M. (1978). *Being and time*. Wiley-Blackwell.

Hilbert, D. M., & Redmiles, D. F. (2000). Extracting usability information from user interface events. *ACM Computing Surveys (CSUR)*, *32*(4), 384–421.

Hong, J. (2011). Matters of design. *Commun. ACM*, *54*(2), 10–11.

Horvitz, E. (1999). Principles of mixed-initiative user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit* (pp. 159–166). ACM.

Hussain, Z., Slany, W., & Holzinger, A. (2009). Current state of agile user-centered design: A survey. *HCI and Usability for e-Inclusion* (pp. 416–427). Springer.

Jiang, W., Hu, C., Pasupathy, S., Kanevsky, A., Li, Z., & Zhou, Y. (2009). Understanding customer problem troubleshooting from storage system logs. *Proccedings of the 7th conference on File and storage technologies* (pp. 43–56). San Francisco, California: USENIX Association.

Jick, T. D. (1979). Mixing qualitative and quantitative methods: Triangulation in action. *Administrative science quarterly*, *24*(4), 602–611.

Just, S., Premraj, R., & Zimmermann, T. (2008). Towards the next generation of bug tracking systems. *VL/HCC* (Vol. 8, pp. 82–85).

Kajko-Mattsson, M. (2004). Problems within front-end support. *Journal of Software Maintenance and Evolution: Research and Practice*, *16*(4-5), 309–329.

Kaplan, B., & Maxwell, J. (2005). Qualitative research methods for evaluating computer information systems. *Evaluating the Organizational Impact of Healthcare Information Systems*, 30–55.

Kearsley, G. (1988). *Online help systems: design and implementation*. Ablex Publishing Corp.

Kelleher, C., & Pausch, R. (2005). Stencils-based tutorials: design and evaluation. *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 541–550). ACM.

Kittur, A., Chi, E. H., & Suh, B. (2008). Crowdsourcing user studies with Mechanical Turk. *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems* (pp. 453–456). Florence, Italy: ACM.

Kittur, A., Chi, H., & Suh, B. (n.d.). Crowdsourcing user studies with Mechanical Turk.

Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Q.*, *23*(1), 67–93.

Knabe, K. (1995). Apple guide: a case study in user-aided design of online help. *Conference companion on Human factors in computing systems* (pp. 286–287). Denver, Colorado, United States: ACM.

Ko, A. J., & Chilana, P. K. (2010). How power users help and hinder open bug reporting. *Proceedings of the 28th international conference on Human factors in computing systems* (pp. 1665–1674). Atlanta, Georgia, USA: ACM.

Ko, A. J., & Myers, B. A. (2004). Designing the whyline: a debugging interface for asking questions about program behavior. *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 151–158). ACM.

Krathwohl, D. R. (1993). *Methods of educational and social science research: An integrated approach.* Longman/Addison Wesley Longman.

Lakhani, K. R., & Von Hippel, E. (2003). How open source software works:"free" user-to-user assistance. *Research policy*, *32*(6), 923–943.

Lakhani, K., & Wolf, R. G. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In Feller, J., Fitzgerald, B., Hissam, S., & Lakhani, K (Eds.), *Perspectives on Free and Open Source Software*. Boston, MA: MIT Press.

Lamberti, D. M., Prager, J. M., & Nappari, M. A. (1994). *Constrained natural language interface for a computer that employs a browse function*. Google Patents.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge university press.

Layzell, P. J., & Macaulay, L. (1990). An investigation into software maintenance-perception and practices. *Software Maintenance, 1990, Proceedings., Conference on*, 130–140. doi:10.1109/ICSM.1990.131342

Lease, M., & Yilmaz, E. (2013). Crowdsourcing for information retrieval: introduction to the special issue. *Information Retrieval*, 1–10.

Lee, K. C., & Lee, D. H. (2007). An online help framework for web applications. *Proceedings of the 25th annual ACM international conference on Design of communication* (pp. 176–180). ACM.

Lehman, M. M., & Belady, L. A. (Eds.). (1985). *Program evolution: processes of software change*. Academic Press Professional, Inc.

Mackay, W. E. (1991). Triggers and barriers to customizing software. *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology* (pp. 153–160). New Orleans, Louisiana, United States: ACM.

Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., & Hartmann, B. (2011). Design lessons from the fastest q&a site in the west. *Proceedings of the 2011 annual conference on Human factors in computing systems* (pp. 2857–2866). Vancouver, BC, Canada: ACM.

Marchionini, G. (1997). *Information seeking in electronic environments*. Cambridge Univ Pr.

Mason, W., & Suri, S. (2012). Conducting behavioral research on Amazon's Mechanical Turk. *Behavior research methods*, *44*(1), 1–23.

Matejka, J., Grossman, T., & Fitzmaurice, G. (2011). IP-QAT: in-product questions, answers, & tips. *Proceedings of the 24th annual ACM symposium on User*

interface software and technology (pp. 175–184). Santa Barbara, California, USA: ACM.

Matejka, J., Li, W., Grossman, T., & Fitzmaurice, G. (2009). CommunityCommands: command recommendations for software applications. *Proceedings of the 22nd annual ACM symposium on User interface software and technology* (pp. 193–202). Victoria, BC, Canada: ACM.

McGrath, J. E. (1995). Methodology matters: doing research in the behavioral and social sciences. In R. M. Baecker, J. Grudin, W. A. S. Buxton, & S. Greenberg (Eds.), *Human-computer interaction* (pp. 152–169). Morgan Kaufmann Publishers Inc.

McInerney, P., & Maurer, F. (2005). UCD in agile projects: dream team or odd couple? *interactions*, *12*(6), 19–23.

Meszaros, G., & Aston, J. (2006). Adding usability testing to an agile project. *Agile Conference, 2006* (p. 6 pp.–294). IEEE.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative Data Analysis: An Expanded Sourcebook*. Sage Publications.

Mørch, A. (1997). Three levels of end-user tailoring: Customization, integration, and extension. *Computers and Design in Context*, 51–76.

Morris, M. R., Teevan, J., & Panovich, K. (2010). What do people ask their social networks, and why?: a survey study of status message q&a behavior. *Proceedings of the 28th international conference on Human factors in computing systems* (pp. 1739–1748). ACM.

Myers, B. A., Weitzman, D. A., Ko, A. J., & Chau, D. H. (2006). Answering why and why not questions in user interfaces. *Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 397–406). ACM.

Nambisan, S., & Baron, R. A. (2007). Interactions in virtual customer environments: Implications for product support and customer relationship management. *Journal of Interactive Marketing*, *21*(2), 42–62.

Negash, S., Ryan, T., & Igbaria, M. (2003). Quality and effectiveness in web-based customer support systems. *Information & Management*, *40*(8), 757–768.

Nichols, D. M., McKay, D., & Twidale, M. B. (2003). Participatory Usability: supporting proactive users. *Proceedings of the 4th Annual Conference of the ACM Special Interest Group on Computer Human Interaction-New Zealand Chapter (CHINZ'03)* (pp. 63–68). Citeseer.

Nielsen, J. (1992). The Usability Engineering Life-Cycle. *Computer*, *25*(3), 12–22.

Nielsen, J. (1993). *Usability Engineering*. Morgan Kaufmann.

Nielsen, J. (1994a). Guerrilla HCI: Using discount usability engineering to penetrate the intimidation barrier. *Cost-Justifying usability*, 245–272.

Nielsen, J. (1994b). Guerrilla HCI: using discount usability engineering to penetrate the intimidation barrier.

Norman, D. (1990). *The design of everyday things*. New York: Doubleday.

Norman, D. A., & Draper, S. W. (1986). *User Centered System Design; New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates Inc. Mahwah NJ USA.

Novick, D. G., Elizalde, E., & Bean, N. (2007). Toward a more accurate view of when and how people seek help with computer applications. *Proceedings of the 25th annual ACM international conference on Design of communication* (pp. 95–102). El Paso, Texas, USA: ACM.

Orr, J. E. (1996). *Talking about machines: An ethnography of a modern job*. Cornell University Press.

Palanque, P., Bastide, R., & Dourte, L. (1993). Contextual help for free with formal dialogue design. *ADVANCES IN HUMAN FACTORS ERGONOMICS*, *19*, 615–615.

Palmiter, S., Elkerton, J., & Baggett, P. (1991). Animated demonstrations vs written instructions for learning procedural tasks: a preliminary investigation. *International Journal of Man-Machine Studies*, *34*(5), 687–701.

Pangoli, S., & Paterno, F. (1995). Automatic generation of task-oriented help. *Proceedings of the 8th annual ACM symposium on User interface and software technology* (pp. 181–187). ACM New York, NY, USA.

Paolacci, G., Chandler, J., & Ipeirotis, P. G. (2010). Running experiments on amazon mechanical turk. *Judgment and Decision Making*, *5*(5), 411–419.

Patrick, A., & McGurgan, A. (1993). One proven methodology for designing robust online help systems. *Proceedings of the 11th annual international conference on Systems documentation* (pp. 223–232). ACM.

Pirolli, P., & Card, S. (1995). Information foraging in information access environments. *Proceedings of the SIGCHI conference on Human factors in*

computing systems (pp. 51–58). Denver, Colorado, United States: ACM Press/Addison-Wesley Publishing Co.

Poltrock, S. E., & Grudin, J. (1994). Organizational obstacles to interface design and development: two participant-observer studies. *ACM Trans. Comput.-Hum. Interact.*, *1*(1), 52–80.

Poole, E. S., Edwards, W. K., & Jarvis, L. (2009). The Home Network as a Socio-Technical System: Understanding the Challenges of Remote Home Network Problem Diagnosis. *Comput. Supported Coop. Work*, *18*(2-3), 277–299.

Ramachandran, A., & Young, R. M. (2005). Providing intelligent help across applications in dynamic user and environment contexts. *Proceedings of the 10th international conference on Intelligent user interfaces* (pp. 269–271). ACM.

Rettig, M. (1991). Nobody reads documentation. *Communications of the ACM*, *34*(7), 19–24.

Rosenbaum, S., Rohn, J. A., & Humburg, J. (2000). A toolkit for strategic usability: results from workshops, panels, and surveys. *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 337–344). The Hague, The Netherlands: ACM.

Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, *18*(11), 613–620.

Scekic, O., Truong, H.-L., & Dustdar, S. (2013). Incentives and rewarding in social computing. *Commun. ACM*, *56*(6), 72–82.

Schon, D. A. (1992). Designing as reflective conversation with the materials of a design situation. *Research in Engineering Design*, *3*(3), 131.

Sellen, A., & Nicol, A. (1995). Building user-centered on-line help. *Human-computer interaction* (pp. 718–723). Morgan Kaufmann Publishers Inc.

Singer, J. (1998). Practices of software maintenance. *Software Maintenance, 1998. Proceedings., International Conference on*, 139–145. doi:10.1109/ICSM.1998.738502

Singh, V., & Twidale, M. B. (2008). The confusion of crowds: non-dyadic help interactions. *Proceedings of the ACM 2008 conference on Computer supported cooperative work* (pp. 699–702). San Diego, CA, USA: ACM.

Singh, V., Twidale, M. B., & Nichols, D. M. (2009). Users of Open Source Software - How Do They Get Help? *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on* (pp. 1–10). Presented at the System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on.

Singh, V., Twidale, M. B., & Rathi, D. (2006). Open Source Technical Support: A Look at Peer Help-Giving. *Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 06* (p. 118.3). IEEE Computer Society.

Snow, R., O'Connor, B., Jurafsky, D., & Ng, A. Y. (2008). Cheap and fast---but is it good?: evaluating non-expert annotations for natural language tasks. *Proceedings of the Conference on Empirical Methods in Natural Language*

Processing (pp. 254–263). Honolulu, Hawaii: Association for Computational Linguistics.

Spannagel, C., Girwidz, R., Löthe, H., Zendler, A., & Schroeder, U. (2008). Animated demonstrations and training wheels interfaces in a complex learning environment. *Interacting with Computers*, *20*(1), 97–111.

Steehouder, M. F. (2002). Beyond technical documentation: users helping each other. *Professional Communication Conference, 2002. IPCC 2002. Proceedings. IEEE International* (pp. 489– 499). Presented at the Professional Communication Conference, 2002. IPCC 2002. Proceedings. IEEE International. doi:10.1109/IPCC.2002.1049133

Stevens, G., & Wiedenhöfer, T. (2006). CHIC - a pluggable solution for community help in context. *Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles* (pp. 212–221). Oslo, Norway: ACM.

Strauss, A., & Corbin, J. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications Inc.

Suchman, L. A. (1994). *Plans and situated actions: The problem of human-machine communication*. Cambridge University Press.

Sukaviriya, P., & Foley, J. D. (1990). Coupling a UI framework with automatic generation of context-sensitive animated help. *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology* (pp. 152–166). Snowbird, Utah, United States: ACM.

Sy, D. (2007). Adapting usability investigations for agile user-centered design. *Journal of Usability Studies*. Citeseer.

Technical Support Cost Ratios. (2000). Association of Support Professionals. Retrieved from http://www.asponline.com/tscr.pdf

Tomasi, M. D., & Mehlenbacher, B. (1999). Re-engineering online documentation: Designing examples-based online support systems. *Technical communication*. Citeseer.

Tourniaire, F., & Farrell, R. (1997). *The art of software support*. Prentice-Hall.

TurboTax Support. (n.d.). Retrieved from http://turbotax.intuit.com/support/

Turk, K. L., & Nichols, M. C. (1996). Online help systems: technological evolution or revolution? *Proceedings of the 14th annual international conference on Systems documentation: Marshaling new technological forces: building a corporate, academic, and user-oriented triangle* (pp. 239–242). Research Triangle Park, North Carolina, United States: ACM.

Twidale, M., & Ruhleder, K. (2004). Where am I and who am I?: issues in collaborative technical help. *ACM conference on Computer supported cooperative work* (pp. 378–387). Chicago, Illinois, USA: ACM.

Tynan-Wood, C. (2010, August 17). The (Better) Future of Tech Support. *InfoWorld*.

Usability Professionals Association. (n.d.). UPA 2009 Salary Survey. Retrieved from http://www.usabilityprofessionals.org/usability_resources/surveys/2009salarysurvey_PUBLIC.pdf

Venturi, G., & Troost, J. (2004). Survey on the UCD integration in the industry. *Proceedings of the third Nordic conference on Human-computer interaction* (pp. 449–452). ACM.

Virvou, M., & Kabassi, K. (2000). An intelligent learning environment for novice users of a GUI. *Intelligent Tutoring Systems* (pp. 484–493). Springer.

Vredenburg, K., Mao, J., Smith, P., & Carey, T. (2002). A survey of user-centered design practice (pp. 471–478). ACM New York, NY, USA.

Vukelja, L., Müller, L., & Opwis, K. (2007). Are engineers condemned to design? A survey on software engineering and UI design in Switzerland. *Human-Computer Interaction–INTERACT 2007* (pp. 555–568). Springer.

Walsham, G. (2006). Doing interpretive research. *European Journal of Information Systems*, *15*(3), 320–330.

Weiss, C., Premraj, R., Zimmermann, T., & Zeller, A. (2007). How Long will it Take to Fix This Bug? (pp. 1–1).

Wiertz, C., & de Ruyter, K. (2007). Beyond the call of duty: Why customers contribute to firm-hosted commercial online communities. *Organization Studies*, *28*(3), 347–376.

Winograd, T., & Flores, F. (1986). *Understanding computers and cognition*. Ablex Publishing Corp. Norwood NJ USA.

Wolf, T. V., Rode, J. A., Sussman, J., & Kellogg, W. A. (2006). Dispelling design as the black art of CHI. *Proceedings of the SIGCHI conference on Human Factors in computing systems* (p. 530). ACM.

Yamauchi, Y., Whalen, J., & Bobrow, D. G. (2003). Information use of service technicians in difficult cases. *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 81–88). Ft. Lauderdale, Florida, USA: ACM.

Yeh, T., Chang, T. H., Xie, B., Walsh, G., Watkins, I., Wongsuphasawat, K., Huang, M., et al. (2011). Creating contextual help for GUIs using screenshots. *Proceedings of the 24th annual ACM symposium on User interface software and technology* (pp. 145–154). ACM.

Yin, R. K. (1992). The case study method as a tool for doing evaluation. *Current Sociology*, *40*(1), 121–137.

Zeng, Q., Kogan, S., Ash, N., Greenes, R. A., & Boxwala, A. A. (2002). Characteristics of consumer terminology for health information retrieval. *Methods of Information in Medicine-Methodik der Information in der Medizin*, *41*(4), 289–298.

Zimmerman, J., Forlizzi, J., & Evenson, S. (2007). Research through design as a method for interaction design research in HCI. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 493–502.