Searching for Software Learning Resources Using Application Context

Michael Ekstrand[1,2], Wei Li[1], Tovi Grossman[1], Justin Matejka[1], and George Fitzmaurice[1]

[1]Autodesk Research
210 King St. East
Toronto, Ontario, Canada, M5A 1J7
{firstname.lastname@autodesk.com}

[2]GroupLens Research
CS&E Dept., University of Minnesota
4-192 Keller Hall, 200 Union St.
Minneapolis, MN, USA 55455
ekstrand@cs.umn.edu

ABSTRACT

Users of complex software applications frequently need to consult documentation, tutorials, and support resources to learn how to use the software and further their understanding of its capabilities. Existing online help systems provide limited context awareness through "what's this?" and similar techniques. We examine the possibility of making more use of the user's current context in a particular application to provide useful help resources. We provide an analysis and taxonomy of various aspects of application context and how they may be used in retrieving software help artifacts with web browsers, present the design of a context-aware augmented web search system, and describe a prototype implementation and initial user study of this system. We conclude with a discussion of open issues and an agenda for further research.

Author Keywords

Help search, context-based search, software learning.

ACM Classification Keywords

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces---training, help, and documentation; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval---search process

General Terms

Algorithms, Experimentation, Human Factors

INTRODUCTION

For about as long as there has been software, there have been help resources for using it. These have come in the form of reference manuals, vendor-provided training and consultancy, third-party books, integrated electronic documentation, in-program assistance (tooltips, agents, etc.), and more recently, Internet-based documentation from software vendors, professional third parties, and other users.

Users typically access computerized help resources via text queries issued web search engines or the search facility of a program's help browser [12], or via in-program help interfaces and agents [10,22]. Search queries allow users to express nuanced questions about their needs, such as "How do I draw a circle in Illustrator?". Search engines, however, are largely unaware of the specifics of particular applications or of the user's context (the contents of their document, what dialogs they have open, etc.). Context-sensitive, in-program help, on the other hand, is generally restricted to answering descriptive questions about user interface elements.

We attempt to combine the strengths of these two approaches – the context-awareness of in-program help and the expressiveness of text searches – by using the user's context to improve results when they search for help artifacts on the Internet.

The user's interaction with the program contains a wealth of information related to their needs: what menu items they have tried, what types of objects they are trying to manipulate, what panels and dialogs they have seen, etc. It seems that it should be possible to use this information to retrieve help artifacts that are more closely targeted to the user's current situation, particularly when they may not know the correct terms to find the resources they need. We focus on help queries that are issued within they user's workflow – while they are using an application – as context is most readily available in that situation. It may be possible to extend the ideas we present to support search use cases, such as general queries for ideas or tutorial material.

In this paper, we first present a taxonomy for understanding context as it relates to user help needs. This framework serves as a basis for understanding how context can be harnessed to help locate help artifacts. We then describe the design of a help retrieval system that takes some elements of user context into account, a prototype implementation of this system, and a preliminary user study. We conclude with a discussion of open issues and directions for future work on this topic.

BACKGROUND AND RELATED WORK

Our present work sits at the intersection of two lines of research. We connect work on software documentation and learning with developments in context-aware search and user-oriented information retrieval.

Context-Sensitive Software Help and Documentation

The computer science community has long been interested in the nature and use of software documentation resources. As early as 1981, EMACS was using the term "self-documenting" to advertise itself [26]. O'Malley et al. wrote of the need to organize documentation around user needs, with varying types of documentation aimed at meeting different software learning or reference goals [25].

Context-sensitive help systems use contextual information from the user's current interaction with a particular application to drive or aid the access for help. Frequently this takes the form of a button or command the user can use to request documentation for the currently-open dialog or a particular widget on the screen; Lee's "?" [22] and the "What's This" help in Microsoft Windows both take this approach. Balloon help [10] and tooltips integrate short help descriptions into an application in the form of small panels shown when the user hovers the mouse pointer over an interface element. Recently, ToolClips has extended this to include short videos showing the particular tool in action [13].

Myers et al. extended the notion of contextual help to allow the user to ask more sophisticated questions about interface elements or application behaviors with Crystal [24]. Crystal allows users to ask "why did this happen?" and other types of why/why not questions while using a program.

The Lumière [18] and EUROHELP [28] projects built complex models of user goals, needs, and knowledge to provide customized instruction or assistance. Lumière integrated with Excel, attempting to infer the user's likely goals from their interactions with the software and provide goal-based, context-sensitive guidance.

Simple contextual help is limited by only allowing the user to contextually retrieve descriptive information about the user interface elements on their screen, while the full user modeling approaches of Lumière and EUROHELP require complex models that are difficult to design and apply. Our context retrieval lies in the gap between these extremes: we use more sophisticated observations of user behavior and application context to disambiguate and clarify user help requests, but do not attempt to build comprehensive user models or provide automated intervention. Beyond improving help searches, strengthening the connection between learning resources and the software environment may also improve software learnability [14].

Context and User Needs in Search

In its early days, web search operated under the traditional text information retrieval model: users would provide queries in the form of keyword lists, phrases, and Boolean expressions, and the system would respond with a list of relevant documents. Researchers have since sought to use context in web search [21]; IntelliZap [11] enabled users to form queries by selecting words in web pages, using the page content combined with a semantic network to augment those queries with additional context. Yahoo! implemented a web-scale system along similar lines, with Y!Q allowing the user to type free-text queries which would be augmented with contextual information from the page from which the query was initiated [19]. Watson [5] took a different approach, extracting text from a user's active word processor document to aid in refining web search for document-related research tasks. The work of Brandt et al. uses source code snippets to aid searches for example code [3].

Bringing contextual search closer to our domain, Wen et al. describe a model for incorporating information about a user's computing environment as context to refine searches related to PC troubleshooting [27]; our work is similar but more general, using information from the user's interaction to aid in help searches. Heckerman and Horvitz probabilistically related terms in user search queries to an engineered help database to match results to the user's information needs [16]. Our taxonomy can serve as a theoretical basis for further development of such systems and the system we present is both simpler and applies to non-engineered sources of help information.

Meeting a user's information need does not end with providing a list of relevant documents. The user must also be able to recognize which documents meet their particular need and make use of its content. The recommender systems field has seen significant work on providing users with explanations of their recommendations [17]; Coyle and Smyth argue that this should be done in search as well [7]. Chi et al. present "information scent" as a paradigm for understanding how users decide whether a particular link is likely to result in useful information [6]. Explanations can be viewed as deliberate scent markers leading the user to a particular resource.

We extend previous work to locate software help by augmenting search queries with contextual data from a user's software application. We also use this contextual data to provide additional information in the search interface so the user can more easily relate retrieved documents back to the application and their context.

UNDERSTANDING CONTEXT

In order to understand how we might use context to aid software help search, it is first necessary to examine what context is and how we understand it. Lieberman and Selker [23] provide a useful functional definition of context as "everything that affects a computation except the explicit input and output". To apply this understanding to the design of an information retrieval system, we can say that context is everything that a perfect information retrieval system would need to know about the user, their situation, the do-

main, and anything else in order to return exactly the results relevant to the user's stated query.

Dourish [8] further argues that context cannot be understood only as a static set of facts providing the background for an action such as a web search. Context and action have a cyclical relationship as context induces actions which give rise to further context; context is therefore dynamic and relational, changing and arising in relationship to actions. Anand and Mobasher [2] provide a useful approach to computationally managing this view of context: rather than trying to entirely model context, their approach looks for definable, observable *contextual cues* that can be incorporated into the context-sensitive retrieval process.

There are many factors which may help determine the intent of a help query and the relevance of particular artifacts to that query. In the remainder of this section, we describe a framework for understanding contextual factors and how they relate to each other and to the user's information need. This framework provides two major benefits. First, it gives us a way to reason about what information we may want to incorporate into the search process and what cues might yield that information. Second, it allows us to understand how particular cues relate to the broader picture of the user's context. Our framework describes contextual factors in terms of their visibility, type, and scope; Figure 1 provides a graphical summary of these dimensions (dimensions in bold are used by our prototype implementation).

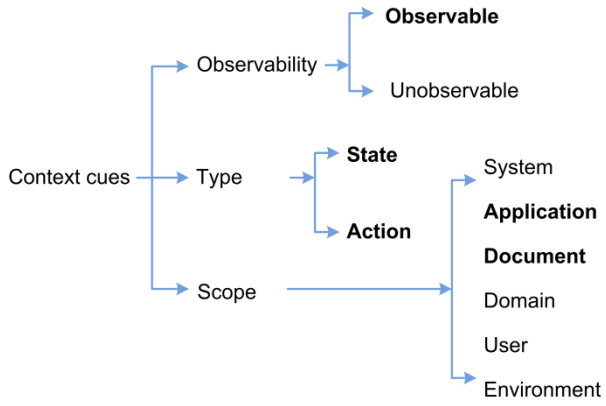

Figure 1: Context factors in help needs

Context Observability

Not all contextual factors are immediately visible to the system. Some things, such as the software settings and the user's recent actions, are directly observable by the application and help system. Others, such as the user's high-level goal (e.g. "draw a basement remodeling plan") or whether they prefer text or video tutorials are not directly observable, at least in any meaningful fashion, and must be inferred, approximated, or ignored.

While it is difficult to make direct use of unobservable contextual factors, recognizing and identifying them allows us to better understand how observable context relates to the user's entire information need, identify strengths and weaknesses in a particular context-aware system, and look for ways to infer or approximate the unobservable factors.

Context Factor Type

Many obvious contextual factors can be described as *state*, or a snapshot of a portion of the system at a particular time. The currently open windows, document, screen resolution, and available hardware are all elements of the current state.

If we only considered state, we would be left with the static background understanding of the context. We therefore approximate the cyclical relationship of context and action by allowing actions themselves to be considered context. This results in our second context type: *action*.

Actions can be performed by the user, the system, or some external agent. Drawing a square, deleting text, and plugging in a graphics tablet are all actions. Actions result in new states; states and actions can therefore be thought of as nodes and edges, respectively, in a graph representing the user's changing context.

We make this distinction to allow states and the means by which those states were reached to be treated separately. If the user's document contains a single square, the user could have created that state by drawing a square on an empty document or by deleting circle from a document containing two shapes. It is conceivable that some information needs depend only on the current state, while others are influenced by the actions which produced the state.

Context Scopes

Finally, we define a number of scopes for contextual factors. *System-level context* pertains to the user's computer system as a whole, independent of any particular application: the operating system and version, configured hardware, installed applications, etc. *Application-level context* is related to the particular application the user is using. This includes the current display state and application modes, open palettes and dialogs, the currently selected tool, etc.

The next two scopes, *document* and *domain*, are related to the work the user is currently doing. The work as it is represented in the application, the objects making up the user's data file and the actions taken on them, makes up the document-level context. Domain-level context relates to the meaning of the system and its components in the context of the user's work domain. If a document contains a circle and a rectangle, that is a document-level contextual factor; the fact that the user is drawing a conference room with a circular table is domain-level. There can be substantial overlap between document cues and domain-level factors or none at all; bitmap image editors typically have a low-level representation of the user's work, while tools such as CASE software and Autodesk's Revit suite store high-level domain information in the document data model.

The next scope is *user-level context*: user states and actions which affect their information needs. This can include their current knowledge of the program, their goals, and other factors. Some observable cues may help in assessing elements of this scope, such as using browsing behavior to identify what types of documents a user finds most helpful.

Finally, there is *environmental context*. A good deal of attention is devoted to this aspect of context in some areas of research, such as ubiquitous computing, but for application help it seems to be less important. There may, however, be elements of the environment that are relevant, such as a physical source object the user is trying to model, or whether there are others in the area who would be distracted by playing a tutorial video.

USING CONTEXT IN HELP SEARCH: SYSTEM DESIGN

We see three primary places in the search process where this additional data can be incorporated. The first is in the normal information retrieval process of generating results for a user's query. Information about the user's context can be considered as an additional component of the query. How exactly this is done may vary across applications and context elements: for applications, it may be appropriate to restrict the scope of the search engine to only return results for that application; for other elements, such as the document contents, it may be more appropriate to simply favor documents mentioning portions of the user's context in the ranking process.

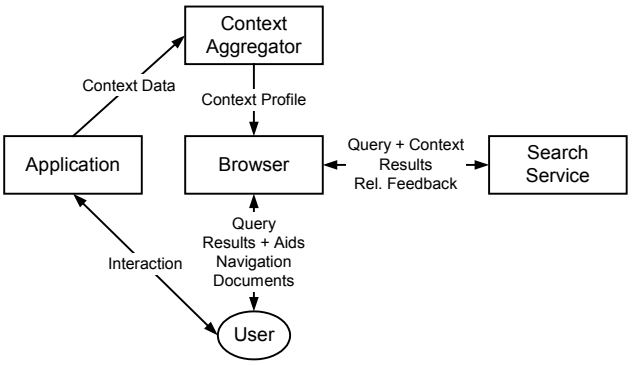

Figure 2: Architecture of contextual search system

The second point where contextual information can be introduced is in presenting the search results. Information scent theory [6] says that users look for markers or cues to indicate whether a particular link is likely to be useful or relevant. A search system aware of the user's context has the ability to provide deliberate cues in the search results to help the user determine which result is most likely to meet their needs.

Finally, the system can use context to provide navigational assistance when viewing the help artifact itself. Visual cues can be helpful to users when trying to find the location of on-screen elements [9]. It may be possible to help users more efficiently use help resources by showing how those resources relate to their context. Indicating what sections of a web page discuss particular tools or objects could help the user more quickly identify the content they are looking for. Calling out mentions of tools with explanations of how to activate them may help users more effectively apply what they learn from a particular resource.

Our proposed architecture, shown in Figure 2, contains several components to support these uses of context. Specific implementations may combine two or more of these components into a single subsystem.

Application Instrumentation

The application(s) for which contextual help search is to be supported must record information about the user's interaction so that it is available in the search process. This can be done internally by modifying the application and its underlying libraries to record this information or by using the operating system or widget toolkit's external inspection and instrumentation facilities. The application instrumentation reports user activities, such as button clicks or tool activations, as well as document-level events and actions to the context aggregator. It may also record other application- or document-level cues.

Context Aggregator

The context aggregator collects contextual information from applications and makes it available to the search service. It maintains a context profile to be used when the user searches for help, possibly aggregating from multiple applications and allowing multiple help browsers to all have access to context information. The context aggregator can also collect system and environmental cues.

Help Browser

The help browser takes context profiles from the aggregator, submits them to the search service along with the user's query, and uses them to augment the search results display and help the user browse help resources. When the user interacts with results, it can also send information back to the search service for relevance feedback and augment displayed resources with further contextual navigation aids. This component can be a dedicated help search and browsing application or an extension for a standard web browser.

Search Service

The search service takes queries and contextual information and provides search results. It can be a standard search engine, in which case the help browser would generate queries incorporating the context data and post-process the results, or a specific context- and application-aware service.

PROTOTYPE IMPLEMENTATION

We prototyped a context-aware search interface to search the Web for help related to the Inkscape vector drawing application[1]. We chose Inkscape because it is open source, allowing us to easily instrument it, and because its docu-

[1] http://www.inkscape.org

ment model contains somewhat structured data (high-level graphical shapes), allowing us to observe meaningful document-level cues.

We instrumented Inkscape to collect user actions and application state changes. Some instrumentation was done in GTK+[2], the widget toolkit used by Inkscape, to log actions such as windows opening and closing. Inkscape and GTK+ report the collected information to the context aggregator over a socket with a simple binary protocol.

We implemented the context aggregator and help browser as a single application using the Qt[3] toolkit. The browser injects custom JavaScript code into each web page after it loads to rewrite search results and augment page content with context-derived hints; context query results and information about pages' references to UI elements are cached to speed up subsequent searches. The browser also provides a navigational panel to allow the user to quickly jump to mentions of various Inkscape user interface elements.

We used Google as the search service, augmenting queries with contextual information and rewriting search results with JavaScript.

Collected Context

Many of the contextual factors discussed earlier are impossible to observe directly, and inferring them can be difficult. In our system, we will need to use context cues that can be observed. Because of little prior work in this area, it is unknown at present what cues are useful and practical to consider in a help system. Further, without an existing set of context, search, and relevance data, it is not possible to learn a model for result relevance to queries and cues. Therefore, from the large set of potential cues, we chose what we thought to be a reasonable selection of context cues at the application and document scopes for our prototype and experiment, preferring cues that can easily be converted into textual queries; the categories from which our cues are taken are indicated in bold in Figure 1. Each context profile contains:

- Last 5 active tools (application scope, state)
- Last 5 editing actions, such as "draw an ellipse", as reported by Inkscape's undo/redo framework (document scope, action)
- Titles of all open palettes, such as "Fill and Stroke" (application scope, state)
- Types of the currently selected drawing objects (document scope, state)

Search Augmentation

Kraft et al. [20] describe a variety of methods for integrating context into search. Since we built our implementation on top of an existing search system we do not control, we used *iterative filtering meta-search* to integrate context with the user's query. For each cue, our system creates a new query by concatenating a query relevant to that particular context element (such as "select tool" when the user has selected the Select tool). Each generated query also contains "inkscape"; this is prepended to the query if the user did not specify "inkscape" in their query. The injected JavaScript runs each of these queries in the background and merges results with the results returned for the user's initial query. 100 results are requested for each query. The final results are displayed in pages of 10.

The weighting and scoring method we describe represents our attempt to devise a mechanism that is reasonably well-principled. The methods employed and their parameters evolved iteratively but have not been empirically tuned.

Weighting Queries

Each query q_i is assigned a weight w_i. If the query entered by the user, denoted q_u, contains the word "inkscape", then its weight $w_u = 0.5$. Otherwise, $w_u = 0.2$ and a new query with weight 0.3 is generated by prefixing q_u with "inkscape". Thus the total weight for the user's initial query, adjusted to prefer results about Inkscape, is 0.5.

Table 1 describes the weights given to each context-generated query. Before blending with the user's query, these weights are normalized to sum to 0.5, resulting in a 50-50 blend of the user's query and contextually-generated queries when the final search results are determined.

Component	Weight
Event contexts	$\frac{1}{2^i}$ for the ith most recent context ($i = 1$ for the current context)
Editing actions	Same as event contexts
Open palettes	1
Selected object types	$\sum_{i=0}^{n-1} \frac{1}{2^i}$ where n is the number of objects of that type selected.

Table 1: Weights of context queries

Ranking Merged Results

In order to rank the merged result list, each resource c is assigned a score $s_i(c)$ for each query q_i. If c is not included in the first 100 results for q_i, then $s_i(c) = 0$; if c appears at position j, s_i is computed using the following formula:

$$s_i(c) = \frac{1}{2^{\frac{j-1}{\alpha-1}}}$$

This equation is derived from the half-life utility metric used by Breese et al. [4]. Using an exponential decay rather than simple rank as used by Kraft et al. works around a problem we observed in early testing where pages that matched multiple queries but only appeared deep in result lists would out-rank more relevant pages that scored very

[2] http://www.gtk.org
[3] http://qt.nokia.com

highly for one or two queries. We use $\alpha = 10$, the length of the default Google search results listing; this causes an article at position 10 to have a score of 0.5 and causes the scoring function to be nearly linear over the first page of results. Each article's cumulative score $s(c)$ is computed as the weighted sum $s(c) = \sum_i w_i s_i(c)$ of its per-list scores. The merged result list is then ranked in decreasing order of article score.

Search Result Presentation

To help the user assess the relevance of search results, our injected JavaScript also augments the search result display with information about the Inkscape tools, palettes, and menu commands mentioned in the page (see Figure 3). This display was designed to show this information in a manner which fits into the existing search results display, providing a context-aware augmentation of the document surrogate [15]. The displayed items are detected by searching the page text for the names of the interface elements along with some synonyms (such as "line tool" for the line and curve tool). Currently-open palettes are indicated in bold face and the current tool's icon is surrounded with a black box. Each item in each of these lists is also a hyperlink to the first mention of that item in the target page.

Figure 3: Search result presentation

In-Page Aids

To aid the user in making use of particular web pages, our browser displays a summary panel containing the Inkscape elements mentioned in that page as in the search results list. This is to help the user quickly jump to portions of the page that discuss tools or commands they are trying to use. All mentions of Inkscape elements are also highlighted in yellow, and hovering over a mention yields a pop-up item describing that command or palette and where to find it in the interface (see Figure 4).

If the user clicks an item in the summary panel, that item is selected: all of its mentions are highlighted green, and the browser jumps to the next mention. The "Jump to Next" and "Jump to Previous" buttons jump to the next and previous mention of a selected interface element. This allows the user to quickly navigate to portions of the page which talk about portions of the interface they have seen or think might be relevant to their task.

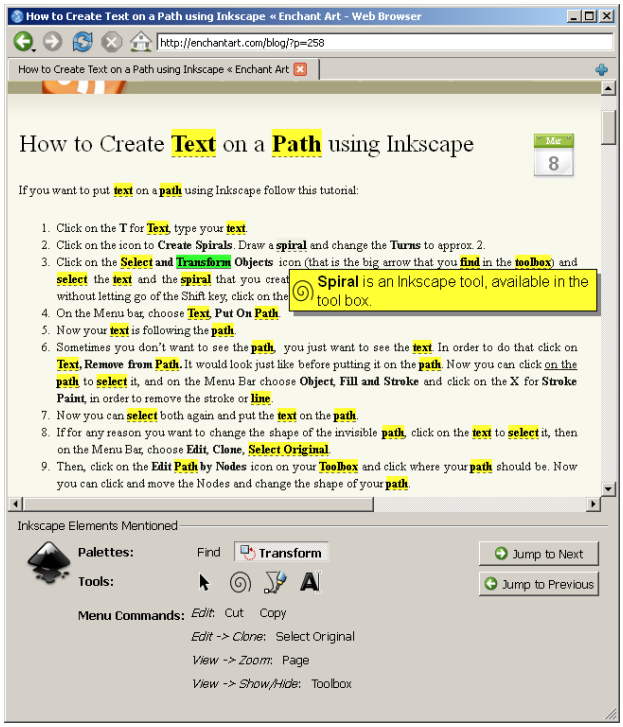

Figure 4: Summary panel and in-page aids; Transform has been clicked in the navigational controls.

USER STUDY

We conducted a user study to see how users interact with and respond to our enhancements. The purpose of this study is not to conclusively validate the benefits of contextual search; rather, it is intended to provide insights into the benefits and challenges inherent in both building and evaluating contextual software help interfaces, and gather user impressions and observations on both the general concept of context-aware search and our system prototype.

Experiment Setup

The study consisted of 4 image construction tasks. These tasks are intentionally arranged from a simple to line drawing using basic tools, to a complicated 3D effect where a sphere had to be drawn and shaded:

1. Draw shapes with the line/curve tool
2. Create text objects and effects
3. Manipulate shapes with Boolean operations
4. Draw and shade a 3D sphere

Users were instructed to work through each task, searching for help as they needed. They were also instructed to avoid

hunting through the interface by trial-and-error but rather to search when they did not know how to do something. The task instructions and completion criteria did not focus on details, as our primary interest was in the search behavior and document relevance. The instructions were primarily graphical to minimize the impact of task instructions on query vocabulary; only the first task specifically named any tool, technique, or menu item.

Each user used either our *enhanced* search system or the standard *stock* Google search for the first 2 tasks and the other interface for the remaining tasks. The standard Google search was performed via our browser with the navigation panel disabled to keep the interface consistent and facilitate logging. To keep response times consistent across conditions, we delayed stock Google results by performing the context-dependent queries but only showing the unenhanced results; this was to prevent users from preferring stock Google searches solely because they were faster. Each user started with empty query and info caches.

Prior to tasks 1 and 3 the observer briefly explained the search interface that they would be using, with a search for "inkscape" as the example results page. For the unmodified condition, the observer explained that it was a standard Google search results page. For the modified case, they explained the search result presentation and demonstrated the summary panel with the Wikipedia article on Inkscape. They also explained the basic browser controls with the first interface used. The observer monitoring the experiment session took notes for further analysis. In particular, the observer recorded which search results they found helpful. Results were considered helpful if the user was able to move forward with the task after referring to it; if uncertain, the observer asked the user whether they found the result useful or if it answered their question.

Users were limited to 20 minutes for each task, at which point they were instructed to move on to the next task. During the experiment, only clarifying questions about the requirements of the tasks were answered – no hints on how to carry them out were provided.

For the study, we posted an ad on Craigslist to recruit users with some experience using graphics software such as Photoshop, Illustrator, or Paint.NET but little to no experience with Inkscape. Fifteen paid participants between the age of 21 and 47 participated in the study.

Experiment Results

Most users did not complete more than 1 or 2 of the tasks; no one completed all tasks correctly. Of the 60 task attempts, 19 were completed (with varying degrees of attention to detail and correctness), 35 timed out, and 6 were stopped early due to user frustration. Search interface had negligible impact on task completion rates.After removing searches occurring too late in the task to be of any use and searches which were immediately corrected by the user, users performed a total of 263 searches with an average of 4.38 searches per task. 98 of the 263 searches were successful, leading directly or indirectly to a page that helped the user with the task. shows the search counts by task and search mode; there are no significant effects of task or mode, although users who used the enhanced search first performed more searches ($p < 0.01$).

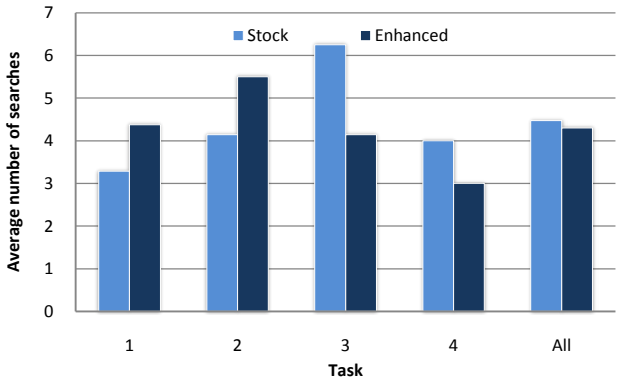

Figure 5: Searches per task by task and mode.

Search Results

To gain insights on the impact of the contextual search system, we looked at the utility achieved by the search results provided by each system. We computed achieved utility by having helpful pages accrue utility for the searches that led to them. Helpful pages (pages enabling the user to move forward) listed directly in the search results produced a utility score of 1; helpful pages linked indirectly by pages in the search results produced a score of 0.75. Only one search query generated multiple helpful pages (in this case, we only considered the higher-ranked of the two), and no search queries generated useful pages more than 1 link removed from the search results.

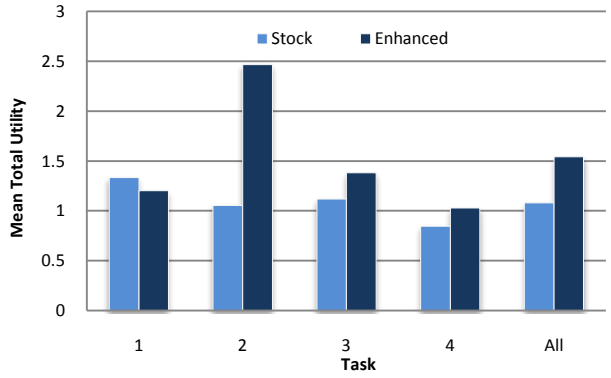

Figure 6: Average total utility per task.

The utility for each search results listing was then computed using the half-life utility metric with $\alpha = 5$ [4]; α was selected based on the empirical analysis of search behavior by Agichtein et al. [1]. If the user found the search results page itself helpful – something in a page summary or list of menu items met their information need – that was recorded

as a utility of 1 and a rank of 0. Half-life utility is similar to the commonly-used *discounted cumulative gain* metric, with the added benefit of being based on a probabilistic model of user behavior.

shows average total utility for each task attempt (the utility of each user's searches during a task is summed, and the sums are averaged over all users for a particular task and interface). Two-way ANOVA shows a marginally significant main effect of search mode on total utility ($p = 0.074$) and no effect of task or task-mode interaction.

Users achieved grater utility with the enhanced search than the stock search in all tasks but the first. One potential reason for this is the nature of the tasks: task 1 was the easiest and exercised a single tool in various ways, while the other tasks were more workflow-oriented where users had to combine several tools or operations to achieve the desired effect. Most searches in task 1 were for things such as "inkscape line tool"; it seems that such queries may already be sufficiently specified that additional context information only introduces noise into the search process. If we exclude task 1 from our analysis, the effect of search mode on total utility becomes significant ($p = 0.031$). This is an early, but encouraging, result.

We also found that the impact of different context sources in providing results varied across tasks – in task 3, edit actions were the most powerful source of useful links, while users found links located by open palettes or active tools more useful in other tasks. This suggests that the utility of various types of contextual cues varies by context – not all information needs will be met equally well by the same classes of contextual information.

User Behavior

Users exhibited some noteworthy behavior while browsing. The first is their use of information on the search results page itself. Sometimes users did not even need to click a link to have their question answered – the text snippet Google extracted from some page contained the information they needed (such as a keyboard shortcut). Two users commented explicitly on this in discussion after the study, saying they appreciated immediate information related to their query. Additionally, help documents sometimes provided keywords that the user used in a subsequent search.

Users also frequently scanned a page quickly to find things that looked relevant, particularly images of effects or screenshots that they could recognize as related to the task. We had intended the highlighting and summary panel navigation to help with this, but users did not make substantial use of the summary panel's navigational capabilities and rarely consulted the pop-ups on UI element references. This was quite possibly due to lack of familiarity. The highlighted references also did not seem to help users skim the page. Our browser highlighted all detected references; a more selective approach may make this feature more useful.

User Receptiveness

Users were divided on whether they found the enhanced results useful. Of the 14 users who told us their preference, 9 preferred the enhanced search to some degree. User response to the augmentation of the search results display was mixed; some users thought it was too distracting, while others found it helpful. One user commented that "the pictorial element helps a lot, just for visually honing in on something". Users were similarly mixed in their response to the summary panel; some found it useful for providing an overview, while others found it goes in the way.

Difficulties and Limitations

We encountered a number of difficulties in conducting this study. Users had difficulty adjusting to Inkscape's user interface and mode errors were common. The nature and quality of available documentation also caused problems. Inkscape tutorials and guides vary greatly in format as well as both technical and presentational quality. Further, much of the documentation is in the form of tutorials and other blog posts that seem good for showing users what kinds of things they can do and satisfying needs in a browse-for-options surfing mode, but are less suited to answering questions about the interface or particular drawing tasks. Many resources, including some of the best tutorials and manuals, were also broad in scope; since our system is currently limited to only retrieving entire documents, it had difficulty locating focused material on specific tools.

There were two areas in which terminology mismatches caused experimental issues. The first occurred in the first task, where we instructed the users to use the "line tool" to draw some simple shapes. The official name for this tool within Inkscape is the "pen tool", but documentation is inconsistent in referring to it and often calls it either the "line" or "curve" tool. With further refinement, context or at least application awareness could make it much easier to resolve these searches. The other mismatch was that many sites call Inkscape's node editor the "node tool", while our UI element detection code only looked for "node editor". Two users noticed that the tool was not detected when it should have been. The effect of these problems on our results is unknown.

LESSONS LEARNED

Our study yielded several promising observations and certain beneficial trends seemed to emerge. In light of previous work showing local context to be useful in search tasks [9], our experiences provide insight into some of the outstanding difficulties in using context to locate help resources. First, the relationship between context and page content must be carefully designed. When context is used to generate additional query terms, as it is in our implementation, the queries need to be appropriate and well-constructed. A combination of expert knowledge of the application and empirical testing should produce query expansions that produce useful results; the difficulties we had with the line tool demonstrate the consequences of errors in this process.

Contextual data is not all positive signal – spurious contextual information also introduces noise. Further, the utility of various contextual factors is not necessarily constant – it may vary based on query or difficult-to-observe factors such as user task. When the contextual data raises the rank of less-useful results, it can cause user confusion; in one of our experiment sessions, the user adjusted their queries (adding quotation marks, additional keywords, and other query refinement techniques) because they did not think the contextually-retrieved results were relevant and were looking for a query that excluded them in favor of more relevant resources. It would be useful to understand not only what factors were improving results, but what factors were introducing noise. More detailed future studies will hopefully shed more light on which types and scopes of context are most effective for providing useful results. Showing the context-generated queries and allowing users to adjust them may also be beneficial.

The utility of contextual search also depends on the existence of diverse, high-quality content. Without diverse content, it becomes less relevant to include context in the query, since results are already limited. Without quality content, the results will not be useful to the user, even if they accurately match the user's current context of use.

We limited our prototype and experiment to simple document- and application-level contextual cues in the current prototype, but have limited data on their efficacy. It may be that we need more sophistication either in the cues we collect or the way we use them (e.g. using more inferred factors, perhaps at the domain scope, or building better ways of incorporating the contextual data into search) in order to see significant impact. More detailed studies will hopefully shed light on how to improve or reenvision incorporating contextual data into the search process.

We have found that information retrieval does not end with presenting a list if results. The benefit users derived from information contained within the search results page itself is noteworthy: if users can have their need met without having to select and browse to a page, it seems that they will be able to continue with their work more quickly. One user specifically suggested overview or disambiguation pages – for example, including a survey of what ellipses are and some of the basic operations that can be performed on them when the user searches for "ellipse". Search engines currently incorporate some of this; besides the snippets extracted from pages, Google answers common queries such as arithmetic, ticker symbols, or currency conversions with immediate answers, and also provides its "I'm Feeling Lucky" to immediately take the user to the first result. The search engine Duck Duck Go[4] takes this a step further, providing "Zero-Click Info" from Wikipedia, Stack Overflow, and other informational sites in a box at the top of the search results page. This seems like a promising direction both for software help search and for general web searches.

CONCLUSION AND FUTURE DIRECTIONS

There is a good deal of work left to be done to make context-aware help search a practical reality. We have identified a number of types of contextual information, but have only used a few of them in the present work. How best to incorporate other types of information remains to be explored; in doing so, greater consideration needs to be given to how help and search systems can select and use appropriate contextual data. The relevant algorithms may well need to be adaptive, adjusting what context is used or how it is incorporated into the search process based on context and the user's search queries. The feature selection and search augmentation methods also need to be tuned, and indexing document passages as well as whole documents may help future systems identify more focused material.

Context could also be incorporated in places beyond the search process itself. More nuanced relationships between context and help artifacts could feasibly be defined as artifacts are created; if a tutorial contains a machine-readable description of the menu commands used, for example, then that could be matched against the user's context to find examples for recently-reverted user commands with high accuracy. Q&A services could similarly capture the context of the question-asker, allowing other users to later find that question when encountering similar problems.

Contextual search further has the potential to help address the information retrieval issues of polysemy and synonymy. Including search terms from the user's context can help guide the search engine to the user's intent. More sophisticated approaches may be able to explicitly use context to resolve identified ambiguities in query or document terms.

It appears particularly promising to use context in a manner more similar to collaborative filtering or relevance feedback: if the system stores contextual "fingerprints" of users that found each resource to be useful, those could be used to help later users find resources which helped others in similar contexts.

User context provides a wealth of information that can be used to improve the ability for search services to meet users' information needs. There are many opportunities in applying context to software help, particularly as the types and sources of that help diversify. We have presented a framework for understanding context with respect to help needs and the contextual cues which may be mined to improve the help retrieval process. We have also prototyped such a help system, studied users' interaction with it, and laid several outstanding challenges in making help systems more sensitive to the nuances of users' particular needs.

[4] http://duckduckgo.com

REFERENCES

1. Agichtein, E., Brill, E., Dumais, S., and Ragno, R. Learning user interaction models for predicting web search result preferences. In *Proc. SIGIR '06*, ACM (2006), 3-10.
2. Anand, S.S. and Mobasher, B. Contextual Recommendation. In B. Berendt, A. Hotho, D. Mladenic and G. Semeraro, eds., *From Web to Social Web: Discovering and Deploying User and Content Profiles*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, 142-160.
3. Brandt, J., Dontcheva, M., Weskamp, M., and Klemmer, S.R. Example-centric programming: integrating web search into the development environment. In *Proc. CHI '10*, ACM (2010), 513-522.
4. Breese, J.S., Heckerman, D., and Kadie, C. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. UAI '98*, (1998), 43-52.
5. Budzik, J. and Hammond, K.J. User interactions with everyday applications as context for just-in-time information access. In *Proc. IUI '00*, ACM (2000), 44-51.
6. Chi, E.H., Pirolli, P., Chen, K., and Pitkow, J. Using information scent to model user information needs and actions and the Web. In *Proc. CHI '01*, ACM (2001), 490-497.
7. Coyle, M. and Smyth, B. Supporting intelligent Web search. *ACM Trans. Internet Technol. 7*, 4 (2007), 20.
8. Dourish, P. What we talk about when we talk about context. *Personal Ubiquitous Comput. 8*, 1 (2004), 19-30.
9. Ehret, B.D. Learning where to look: location learning in graphical user interfaces. In *Proc. CHI '02*, ACM (2002), 211-218.
10. Farkas, D.K. The role of balloon help. *SIGDOC Asterisk J. Comput. Doc. 17*, 2 (1993), 3-19.
11. Finkelstein, L., Gabrilovich, E., Matias, Y., et al. Placing search in context: the concept revisited. *ACM Trans. Inf. Syst. 20*, 1 (2002), 116-131.
12. Goodall, S.D. Online help: a part of documentation. In *Proc. SIGDOC '92*, ACM (1992), 169-174.
13. Grossman, T. and Fitzmaurice, G. ToolClips: an investigation of contextual video assistance for functionality understanding. In *Proc. CHI '10*, ACM (2010), 1515-1524.
14. Grossman, T., Fitzmaurice, G., and Attar, R. A survey of software learnability: metrics, methodologies and guidelines. In *Proc. CHI '09*, ACM (2009), 649-658.
15. Hearst, M.A. Presentation of Search Results. In *Search User Interfaces*. Cambridge University Press, 2009.
16. Heckerman, D. and Horvitz, E. Inferring Informational Goals from Free-Text Queries: A Bayesian Approach. In *Proc. UAI '98*, Morgan Kaufmann (1998), 230--237.
17. Herlocker, J.L., Konstan, J.A., and Riedl, J. Explaining collaborative filtering recommendations. In *Proc. CSCW '00*, ACM (2000), 241-250.
18. Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, K. The Lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. UAI '98*, Morgan Kaufmann (1998), 256--265.
19. Kraft, R., Maghoul, F., and Chang, C.C. Y!Q: contextual search at the point of inspiration. In *Proc. CIKM '05*, ACM (2005), 816-823.
20. Kraft, R., Chang, C.C., Maghoul, F., and Kumar, R. Searching with context. In *Proc. WWW '06*, ACM (2006), 477-486.
21. Lawrence, S. Context in web search. *Bulletin of the Tech. Comm. on Data Eng.*, (2000).
22. Lee, W. "?": a context-sensitive help system based on hypertext. In *Proc. ACM/IEEE DAC 1987*, ACM (1987), 429-435.
23. Lieberman, H. and Selker, T. Out of context: computer systems that adapt to, and learn from, context. *IBM Syst. J. 39*, 3-4 (2000), 617-632.
24. Myers, B.A., Weitzman, D.A., Ko, A.J., and Chau, D.H. Answering why and why not questions in user interfaces. In *Proc. CHI '06*, ACM (2006), 397-406.
25. O'Malley, C., Smolensky, P., Bannon, L., et al. A proposal for user centered system documentation. In *Proc. CHI '83*, ACM (1983), 282-285.
26. Stallman, R.M. EMACS the extensible, customizable self-documenting display editor. In *Proc. Text Manip. '81*, ACM (1981), 147-156.
27. Wen, J., Lao, N., and Ma, W. Probabilistic model for contextual retrieval. In *Proc. SIGIR '04*, ACM (2004), 57-63.
28. Winkels, R. User modelling in help systems. In *Proc. CAL '90*, Springer (1990), 184-193.