VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

A Study of Usability of Web-Based Software Repositories

Neil Clayton, Robert Biddle, and Ewan Tempero

Technical Report CS-TR-99/10
June 1999

Abstract

Software repositories have a key role to play in software reuse strategies to improve software development productivity. However, repositories have not been conspicuously successful, and have not been able to play their role very successfully, despite improvements in supporting technology such as the web. We speculate that a weakness in repositories is their usability. In this paper we present a study of the usability of web-based software repositories, using the "cognitive walkthrough" usability inspection method. We found that repository usability did indeed seem to be a difficulty, and identified several significant points where usability could be improved.

1 Introduction

The idea of developing software by re-using existing software "components" has been around since the dawn of software engineering as a discipline [11]. These components are stored in software repositories so as to be available to the software development process (see [10] for example). Despite this long history, software components have not significantly reduced the cost of software development. It is widely held that this is in part due to a failure of software repositories [1]. While there have been many attempts to build repositories, they have had at best limited success. We believe a major reason why repositories have not been successful is that they are too hard to use. In this paper, we describe a study we performed on the usability of web-based software repositories.

There could be a number of reasons for this lack of success for repositories. For example, repositories may not have enough components to be useful, or the components in existing repositories might not be of high enough quality that developers trust them. Another reason why repositories have not been successful might be that they are just too hard to use. If component reuse is to be successful, the cost of reusing a component from a repository must be perceived to be less than writing the component from scratch. If using the repository is a frustrating experience for its users, they are just as likely not bother with it. Usability is now recognised as an important factor in the lack of use of CASE tools [6]. We have been investigating the hypothesis that existing repositories have poor usability.

This paper describes our initial investigation into the usability of repositories. We have concentrated on publicly available web-based repositories, because web technology makes repositories very accessible. We used an evaluation technique from the HCI literature, namely cognitive walkthroughs[16].

The paper is organised as follows. In section 2, we provide a brief overview of the kinds of repositories we have been studying, and in section 3 we discuss the concept of usability. We then present our results in section 4, and conclude in section 5.

2 Software Repositories

In order to evaluate a software repository, we must first have a good understanding of what we expect from one. The motivation for using repositories is reduce the amount of software that needs to be developed by reusing existing software components. The expectation is that a repository must:

- store the software components (or references to them)
- provide a mechanism for finding components
- provide a mechanism for acquiring components
- provide a mechanism to manage (adding and organising) components

The functions listed above describe basic repository functionality. However, in order for a repository to be useful for supporting our goals of reuse, it must perform each of these functions well. If the repository does not contain enough useful components, it will not be used. Similarly, if the effort involved to find, acquire or manage the components is perceived as being unproductive, then the repository will not be used. So, in addition to the basic functionality, we expect a repository to have other features to enhance its usefulness. Some desirable features of a usable repository include:

- sophisticated search mechanisms that allow queries to be easily formulated
- documentation about the component that allows its semantics and quality to be evaluated

- category structure to allow navigation as an alternative to searching

Within the domain of web-based repositories, we have identified three categories of repositories: FTP sites, custom web sites, and repository engine-based sites.

2.1 FTP Sites

The File Transfer Protocol was created to ease the difficulty of transferring files from one host to another. Because FTP technology has been available for some time, it has been used as the basis for providing locations for storing, and providing access to files — that is, primitive repositories. In a number of FTP sites, the files are in fact software in various forms, and so are of interest to us. Although pre-web, most web browsers have rudimentary support for FTP sites, and so we have included them in our study.

The management of components is done via the underlying file system. Typically the file system will be used to provide some form of categorisation through the use of directory structures, possibly with "well-known" files to describe the structure (such as README text files). This is the mechanism that users use to find components in the repository. Acquiring a component is provided for through FTP itself.

One obvious difference between FTP and a more sophisticated repository is that the FTP site itself does not provide support for searching for components. Also missing is the ability to attach descriptions or abstracts to each component, or provide alternative category views. Nevertheless, such primitive repositories provide a useful baseline for our study.

2.2 Custom Web Sites

Most web sites are repositories of a kind. Some are intended to provide shared use of software, both applications and components. Such sites provide for component descriptions in the form of meta-data, a well defined category structure and searching capability.

Users of custom web site repositories can browse the category structure or perform searches against the site. These searches are typically keyword based, where a match occurs if one of the keywords occurs in the component description. The search engines usually display a list of hits in a similar fashion to the popular web search engines like AltaVista.

Such repositories have typically grown from small, manually managed, hobby sites. As the sites have grown, more elaborate category structures have been introduced, and tools have been developed to help with management.

2.3 Repository Engines-Based Sites

The third category of repositories is web sites that are based on repository engines, that is, they use a set of tools explicitly developed to provide web-based repository support. We studied two such engines: Repository in a Box [14], and the MORE family [4]. Both of these engines support the creation of repositories of any kind of entity or "asset". We limit our discussion below to assets that are software components.

2.3.1 Repository In a Box

Repository in a box (RIB) is freely available repository developed by the National High Performance Software Exchange at the University of Tennessee [14]. RIB is implemented as a package of PERL scripts running against a web server using CGI.

RIB provides many features expected of a repository, including:

- the ability to define the domain and structure

- keyword based searching against component descriptions
- representing the contents of a remote repository as though it were local.

RIB does not store the component itself in the repository database, but instead stores meta-data about the component. RIB allows the user to browse and search the domain hierarchy. Browsing involves clicking on links to select and show a more specialised part of the hierarchy, or moving back up the hierarchy by clicking on a provided "parent" link.

A user can search the repository from the main page by entering keywords into the search text box provided. The RIB provided search tool returns a summary of those component matching the keywords. The user can then examine the complete component description. The description includes a URL to the location of the actual component, which would take the user either to some other site on the web, or perhaps allow them to directly down-load the component.

2.3.2 MORE

MORE (Multimedia Object Repository Environment) was developed as part of the Repository Based Software Engineering Program at the University of Houston — Clear Lake [4]. The initial concept has been extended in different ways, leading to four versions of MORE: MORE, MORE2.0, MOREjava and MOREplus.

MORE is a web based system, making use of stock HTTP servers. Like RIB, MORE stores meta-data about components, and provides standard browsing, categorisation, description and searching functions. However MORE is significantly more sophisticated than RIB. In particular, MORE allows the definition of groups and users, searching based on synonyms, natural language and pattern based searching.

MORE maintains the notion of a "class". The class provides a means of shaping the meta-data that MORE stores regarding a component. Components can be put into different classes, allowing different components to have different attributes.

The basic design goals of the original MORE were:

- An adaptable group/user definition mechanism for managing access to proprietary sub-collections of components.
- Optimisation in the storage of meta-data concerning components
- Use of the World Wide Web, and Mosaic in particular, as the sole user interface (including all administrative access)

MORE2.0 is the successor to MORE. MORE2.0's implementation is identical to MORE in that it runs on a stock HTTP server, using CGI. The main differences are that MORE2.0 provides more ways to cross-reference components and to display information about groups of components.

MOREplus is the commercial version of MORE2.0, developed by MountainNet. The main difference is that it allows administrators to define new classes. These classes are derived from existing classes, and can contain additional attributes.

Because MOREplus allows for the definition of administrator-defined classes, the search engine has been enhanced. Users can choose to search particular classes. When the class is chosen, a list of class attribute names is displayed and the user can enter values for any of these attributes.

3 Usability

> *Usability: the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. IEEE [7]*

> *The usability of an interface is a measure of the effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment with that interface. ISO [8]*

Usability concerns how easy computer systems are to use. Usability is often distinguished from *utility*, which more concerns functionality. In this paper we are not addressing the utility of repositories, and are assuming the functionality as outlined in the previous section. We *are* addressing the usability of repositories, and so are concentrating on how easy they are to use.

As the definitions above make clear, usability covers many aspects of the use of a system. For example, we must consider ease of learning, ease of regular use, memorability, error handling, and even subjective satisfaction. We must also consider the kinds of users who are likely to use the system, and what kind of goals they will be trying to achieve. Once we determine the aspects of usability and users that are most of interest, we can then conduct usability studies accordingly.

One of the usual ways of classifying users for usability concerns their experience [15]. In particular, system characteristics leading to good usability are different for novice users and expert users, and intermittent or casual users are often identified as a third category. In studying usability of repositories, however, it seems there are several kinds of experience that are relevant: programming experience, application domain experience, and repository experience. This quickly leads to a wide range of user characteristics, each of them possible, and each leading to different usability characteristics desirable in a repository.

We decided to concentrate on one particular combination of user experience characteristics: an expert programmer in the language and environment they are using, with casual application domain experience, and casual repository experience. We believe this is an important combination because programmers do develop expertise in particular languages and environments that helps them even in novel situations, but still find themselves having to develop or maintain software across diverse application areas that they only work with intermittently. They are likely to understand the principle of repositories, likely to have used several over time, but likely not to use one in particular on a frequent basis.

This particular combination of user characteristics also leads to some illumination about the likely goals such users will be trying to achieve, and what assistance they will benefit from. We believe these users will be confident in their programming, but will look to repositories to find components providing some useful functionality relevant to the application domain they are working within.

These expert programmers will not need any help relating to the programming language or frameworks, and indeed most experts prefer complete but terse technical specifications.

They will have experience with the application domain, and be familiar with most of the technical terms. However, as intermittent users they will likely appreciate availability of well-structured domain information, glossaries, and clarifying documentation. Similarly, they will understand the basics of repository usage, and will not want to be impeded in performing straightforward tasks, but will appreciate reminders, navigational assistance, and the availability of documentation about the repository itself.

4 Cognitive Walkthroughs

In this study we have chosen to evaluate the usability of repositories using cognitive walkthroughs [16]. This approach is one of what Nielsen and Mack [13] refer to as "usability inspection" methods, which determine usability without requiring actual sample users. Like other usability inspection methods, cognitive walkthroughs are particularly suitable for early or initial usability evaluation studies, because they allow a lightweight and flexible approach that is difficult when working with sample users. In particular, cognitive walkthroughs allow early identification of usability issues that can then be explored in more detail.

The cognitive walkthrough approach is based on Lewis and Polson's CE+ theory of exploratory learning, which describes HCI in terms of:

- User sets a goal to be accomplished.
- User searches the interface for currently available actions (menus, buttons etc).
- User selects the action that seems most likely to achieve desired goal.
- User performs the selected actions and evaluates the systems feedback for evidence that progress is being made towards the goal.

The technique requires a description of the user and their relevant knowledge, specific descriptions of scenarios consisting of one or more tasks to be performed, and some way to check for achievement of the user's goal. Ideally, the team performing the walkthrough should consist of a small number of people, where one member records all notes and discoveries and another member facilitates in order to keep the evaluation moving. The remaining members evaluate the system. These roles are not fixed, and can be adapted to suit different sized teams, but the approach works best when performed by more than one person.

The job of each evaluator is to assume the role of the user. For each goal, they then suggest the users next action, and present believable stories as to why that user would choose that particular action. In some cases, this might be clear cut, with all evaluators agreeing on the next action to be taken. In others, the specific next action may not be so clear. To assess the ease with which the correct action will be selected, the process suggests four criteria for evaluating the stories told: consideration of the users goal, accessibility of the correct control, quality of match between controls label and the goal, and finally, feedback provided after the control is acted on.

4.1 Scenarios for repository cognitive walkthroughs

The scenarios used were simple, and representative of the most common actions a user will typically perform, in day to day use of a repository. The scenarios used take the form:

1. Locate a component that is known to be in the repository. This scenario allows us to determine how easy it is to find components.
2. Try to locate a component that is known not to be in the repository. This scenarios allows us to determine whether the repository can convince us that the component is not present in the repository.

When we as evaluators attempt to carry out the model users task, we are able to see what support is given by the repository, experience how easily the task can be accomplished and identify usability difficulties or unintuitive behaviour. We often discover more details about the user during this process. From the walkthrough, we can gain a better understanding of the usability of the repository.

We conducted cognitive walkthroughs of software repositories using scenarios as described above. For each repository, we tailored the scenarios as appropriate. We present below the results of our walkthroughs for three repositories, one for each category of web-based software repository described in section 2.

4.2 FTP Site

We chose *ftp://ftp.cdrom.com* [2] since it is a large FTP site. The site is home to a huge number of files, on a diverse range of subjects. Being the size it is suggests that it should be representative of the best organised FTP sites available publicly on the web. The scenarios used were:

1. Find a Delphi design time *expert* that can be inserted into the IDE to enlarge the IDE workspace. An *expert* is a piece of executable code that can be dynamically installed into the IDE to change its behaviour.
2. Find a component that allows us to activate/deactivate the installed IDE design time *experts*.

4.2.1 Overview of evaluation

Before the evaluation, there was a general agreement that while an FTP server did not provide full repository semantics, it did allow for an efficient and reasonably easy to use informal environment. We were surprised to find that the FTP server was less usable than we first thought. In particular, the limited length for names (we assume these limited name lengths were being used to remain compatible with older systems) made it difficult to clearly identify the content of a directory (hereinafter referred to as a category).

Without performing an exhaustive manual search by downloading each item in a category and examining its documentation, it was virtually impossible to know whether we had found the component we were looking for. In addition, the site had no indexes at the category level, and it appeared that names originally created for a category were poorly chosen.

We failed to find the component outlined in scenario one, and were not able to convince ourselves that component in scenario two did not exist.

4.2.2 Identified usability problems

- There were no category *00index* or *README* files. Thus it was very difficult to know what each category actually stored. Example: At first glance, we found a category named *Delphi*. This seemed like the right place to go for Delphi components, but we eventually found that it contained the pages for a Delphi related web site. The category that contained the components was called *delphideli*. Had we a short description of each of the categories of the site, we would have been able to pinpoint the right category the first time. This highlights the need for well maintained and accurate category descriptions.
- There was a complete category of the entire site. This was however a large 85MB uncompressed file, available as an 11MB compressed tape archive. The directory consisted of the output of the command *ls -lR*. At the time, it did not seem reasonable to download this to see if we could find either the correct category or perhaps the component in the list.
- The user's ability to identify the contents of a category is totally dependent on their interpretation of the category name. This is problematic because a category name provides little information. The problem is compounded in the case of this FTP site because an attempt has been made to keep category names less than 13 characters in length, in order to remain compatible with all machines that may view this FTP site.

- The categories are listed in case sensitive alphabetical order. Therefore *Xerox*, *delphi*, *beos* and *Delphi* would be ordered as *Beos*, *Xerox*, *beos*, *delphi*. The top level category list contains many pages of entries. We made the observation that a user unaware of this case sensitive ordering, who was looking for *delphi*, might think that it did not exist at all. This could happen because the fist page contains all category names whose first letter is in upper case, yet there are many more categories (including *delphi*) visible when the page is scrolled down.
- Some categories were aliases to other existing categories. This was not immediately obvious when looking at the category name. There was an indication of this on the far right hand side of the screen, which showed an arrow pointing to another category name. Considering the users expert programming experience we might conclude that they would be able to figure out that this was a "linked" directory. Conversely, the model user had only casual repository experience — so the repository should have clearly indicated that this category was in fact only an alias some other category.
- There were no short or long descriptions of the components contained in a category. The only way to find out what an component actually did was to download and examine it. In general, this is not going to be feasible. The *delphideli* category we were examining contained many hundreds of components.
- Components in a category had very short, non meaningful names. Again, it appeared that the site was sticking with the 8.3 naming convention used on PCs running MSDOS. Because of this, it was impossible to tell if an component met our requirements or not. We did not manage to find the component mentioned in the first point of this scenario, even though we *knew* it existed on the site.

4.3 Jars

Jars [9] is a web based collection of Java code components, associated with Gamelan [5]. Gamelan was one of the first web based collections of Java components which has now expanded into a more general Java "directory", including Java and browsers, Java and games, graphics, distributed computing and so on.

The basic layout of Jars is a menu down the left hand side of the browser, with content on the middle/right hand side. This allows for fast navigation to any other part of the site — no matter where you are at the time. Figure 1 shows the contents of the browser when you first arrive at the site. The scenarios were:

1. Search for a Java package that implements at least the following widgets: ganntt chart, table and tree view.
2. Find a free JavaBean that implements purely a ganntt chart, which uses Swing 1.1.x.

4.3.1 Overview of evaluation

Our initial goal was to find out how we could find the component of scenario one, using the interface provided by Jars. This turned out to be non-trivial, in that there appeared to be multiple views into the category structure, and it was not clear what the differences were between these views. Also, a link to the search engine was not immediately visible (it was available at the bottom of the navigation panel). We first tried to search using the terms "ganntt", "tree" and "table". This produced no results. We then reduced the term list to "ganntt" — and eventually found the component some way down the list of search results. We were not able to find the component described in scenario two, but we lacked confidence that it was indeed absent due to our previous experiences with the search engine while attempting to perform scenario one.

Figure 1: Main page of www.jars.com

4.3.2 Identified usability problems

- At the top of the first screen on Jars there is an edit box. Initially, we assumed this to be a place where we could start a search. It turned out to be a way to enter your email address so that updates could be e-mailed to you.
- A link to the search page was available near the bottom of the left hand side menu. In our display we were able to see this search option. It might not be visible on other displays without first having to scroll down.
- Jars presents multiple views of the component hierarchy. The way in which it does so is confusing. Jars presents two main views: *categories* and *resources*. Listed *categories* include business, utilities, multimedia, games, programming whereas *resources* appear more technology based, including DHTML, Java, ActiveX, PERL and so on.

  The relevant category view given that we are searching for a reusable ganntt/tree/table component is "Programming". The relevant resource is "Java", however there are other possibilities including "JavaBeans listings", "Other listings" and "Enterprise beans". The confusion begins when we view both the category and the resource pages.

  Choosing the "Programming" category gives us a choice of resources. The choice is very similar to the resource listing found on the main page. Choosing the sub-category "Java" gives us a page, which seems to be structured with the same headings found on the resource list.

  If we then choose the "Java" resource from navigation menu on the left hand of the screen, we arrive at exactly the same page. So we are left wondering: *what is the difference between a category and a resource ?*
- The search engine failed to produce the desired results. The instructions for usage of the search engine indicated that multiple words could be entered. We initially attempted to use the words "ganntt", "table" and "tree". The search produced no results. We then tried simply "ganntt". The list appeared to be ranked by number of matches of any term used in the search phrase. Inspection of the result list also showed an component description which did not include the word "ganntt"!

  Eventually, we were able to find the component we thought we were looking for. Interestingly enough, the description did indeed contain all three words "ganntt", "table" and "tree". As a result, our confidence in the recall of the search engine was not high.

4.4 Cranberry Square Software Market

The Cranberry Square Software Market [3] is a public software repository, implemented using MOREplus [12]. The scenarios used were:

1. Find a C++ component to manipulate matrices that must be able to do matrix transposition.
2. Find an XML parser for PERL.

4.4.1 Overview of evaluation

Upon entering the repository, the most obvious choice was to perform a "quick search" for our desired component. The resulting list was quite long, and we were not able to find the component immediately. We attempted to narrow the scope of the search using the boolean "and" operator, as suggested by the help, but "and" was interpreted as a term in the query, producing a larger set of search results. We managed to find the component using the search mechanism, but were not certain we had found the right one since the descriptions did not tell us if any of the matrix components performed transposition. We assume this is because transposition is a very basic function. We concluded that the site did not contain a PERL XML parser, since the search produced no results when the single query term "XML" was used.

4.4.2 Identified usability problems

- Component descriptions were adequate but not necessarily complete. In particular there was no mention of transposition functions in the descriptions of matrix components.
- The search engine query language appeared broken. The boolean "and" operator did not work, even though on-line documentation suggested that we use it.
- On–line documentation of the repository interface was poor.
- It was unclear what function of "pattern search of local instance" meant, thus we stuck to using the basic search engine. It is possible that had we known more about the repository we might have been able to find the component more quickly. The repository itself could have suggested one of its alternative search methods.
- The component descriptions had user definable fields. These could be clicked on for more information, however the resulting help text was trivial and uninformative.
- In the case where the repository meta-data pointed directly to the component yet we had a minimal description of this component, there was no way to tell if we had the right one without first downloading and examining its documentation.
- Links from a component description were often not to the component itself, but to either pages which described the component, packages *containing* the component or even home pages for the companies owning the component. The result being lack of consistency in interface (pages describing the component were not formatted in the same manner as the repository) and a need to perform further searches.

5 Conclusions

Our aim in this study was to explore the usability of web repositories. However, we were not certain we would find significant problems, and we had no particular theory about what weaknesses we might find.

The cognitive walkthroughs detailed in the previous section, and in other similar studies, did suggest that usability was indeed a significant problem. In fact, we always found more difficulty than we expected. We were surprised because we were felt we were the kind of users the repositories were intended to serve, and we were using the repositories in just the way that was intended. We were especially surprised to see how difficult it was to find a component that we had previously ascertained was indeed in the repository.

As we recorded the usability problems in each situation, several common points began to emerge. One point that quickly arose is that repositories should quickly and clearly direct the user to the functionality that is provided. We found over-comprehensive presentations (see figure 1) confusing and unhelpful. The main tasks that repositories support are browsing a category structure, searching by keyword, and inspecting entries for individual components. For each of these, we found significant common usability problems, and offer the following suggestions:

- Browsing
  - Categories should be clearly described so that users can determine what components will fall within the category.

  - Multiply connected structures should allow users to determine where they have already browsed.
  - The number of components in categories should be shown without needing to browse the category itself, so that users can easily distinguish populous and sparse categories.

- Searching

  - Search engines should use all available text about components in matching user queries.
  - Accurate documentation should be quickly available for the syntax required for complex queries.
  - Results from user queries should be shown ranked with the method of ranking clearly indicated.

- Components

  - Component entries should always be shown with a very brief but helpful description to facilitate selection of candidates while browsing or evaluating search query results.
  - Component entries should always have more complete documentation quickly available to allow rapid evaluation of the suitability of selected candidates.

We believe the suggestions above address the main common usability problems we encountered in our study. However, we acknowledge that making progress in this area requires further work, both to determine if our findings are consistent with other studies, and to use the findings to guide exploration of ways to design more usable repositories.

Software repositories have a key role to play in allowing the promise of software reuse to be fulfilled, and to facilitate more productive software development. The technology base of the web has made this easier than ever before. However, even with the web, we have found that repository usability is limited. We speculate that it is usability that holds back repositories from wider acceptance and support. We hope that by exploring this issue, we are working toward the design of more successful repositories, and the benefits they will bring.

References

[1] Frederick P. Brooks. *The Mythical Man-Month.* Addison-Wesley, Second edition, 1995.

[2] Public FTP Site for Walnut Creek CDROM. ftp://ftp.cdrom.com.

[3] Home page for Cranberry Square Software Market, by MountainNet. http://rbse.mountain.net/mpcs/.

[4] D. Eichmann, T. McGregor, and D. Danley. Integrating structured databases into the web: The MORE system. In *First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

[5] Gamelan — The Official Java Directory. http://www.gamelan.com.

[6] Juhani Iivari. Why are CASE tools not used? *Communications of the ACM*, 39(10):94–103, October 1996.

[7] Institute of Electrical and Electronics Engineers, New York, U.S.A. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*, 1990.

[8] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (VDTs) — Part 11: Guidance on usability*, 1998.

[9] Home page for JARS repository. http://www.jars.com.

[10] Carma McClure. *Software Reuse Techniques.* Prentice Hall, 1997.

[11] M. D. McIlroy. Mass produced software components. In P. Naur and B. Randell, editors, *Proceedings of NATO Software Engineering Conference*, volume 1, pages 138–150. NATO Science Committee, January 1969.

[12] MOREplus home page, at MountainNet. http://rbse.mountain.net/MOREplus/.

[13] Jakob Nielsen and Robert L. Mack. *Usability Inspection Methods.* Wiley, 1994.

[14] Home page for Repository in a Box (RIB), National High Performance Software Enchange, University of Tennessee. http://www.nhse.org/RIB/.

[15] Ben Shneiderman. A taxonomy and rule base for the selection of interaction styles. In Ronald M. Baecker, Jonathan Grundin, William S. A. Buxton, and Saul Greenberg, editors, *Human-Computer Interaction: Toward thhe Year 2000.* Morgan-Kaufmann, San Francisco, second edition, 1995.

[16] Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson. The cognitive walkthrough method: A practitioner's guide. In Jakob Nielsen and Robert L. Mack, editors, *Usability Inspection Methods*, chapter 5, pages 105–140. Wiley, 1994.