WebQuilt: A Framework for Capturing and Visualizing the Web Experience

Jason I. Hong and James A. Landay
Group for User Interface Research, Computer Science Division
University of California at Berkeley
Berkeley, CA 94720-1776 USA
+1 510 643 7354

{jasonh, landay}@cs.berkeley.edu

ABSTRACT

WebQuilt is a web logging and visualization system that helps web design teams run usability tests (both local and remote) and analyze the collected data. Logging is done through a proxy, overcoming many of the problems with server-side and client-side logging. Captured usage traces can be aggregated and visualized in a zooming interface that shows the web pages people viewed. The visualization also shows the most common paths taken through the website for a given task, as well as the optimal path for that task as designated by the designer. This paper discusses the architecture of WebQuilt and also describes how it can be extended for new kinds of analyses and visualizations.

Categories and Subject Descriptors

H.1.2 [**Models and Principles**]: User/Machine Systems – *Human factors;* H.3.5 [**Information Storage and Retrieval**] Online Information Services – *Web-based services;* H.5.2 [**Information Interfaces and Presentation**] User Interfaces – *Evaluation / methodology;* H.5.4 [**Information Interfaces and Presentation**] Hypertext/Hypermedia – *User issues*

General Terms

Measurement, Design, Experimentation, Human Factors

Keywords

usability evaluation, log file analysis, web visualization, web proxy, WebQuilt

1. INTRODUCTION

There are two problems all web designers face: understanding what tasks people are trying to accomplish on a website and figuring out what difficulties people encounter in completing these tasks. Just knowing one or the other is insufficient. For example, a web designer could know that someone wants to find and purchase gifts, but this isn't useful unless the web designer also knows what problems are preventing the individual from completing the task. Likewise, the web designer could know that this person left the site at the checkout process, but this isn't meaningful unless the designer also knows that he truly intended to buy something and is not simply browsing.

WWW10, May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

There are a variety of methods for discovering what people want to do on a website, such as structured interviews, ethnographic observations, and questionnaires (for example, see [3]). Instead, we focus here on techniques for tackling the other problem, that is, understanding what obstacles people are facing on a website.

Traditionally, this kind of information is gathered by running usability tests on a website. A usability specialist brings in several participants to a usability lab and asks them to complete a few predefined tasks. The usability engineer observes what stumbling blocks people come across and follows up with a survey and an interview to gain more insights into the issues.

The drawback to this traditional approach is that it is very time consuming to run usability tests with large numbers of people: it takes a considerable amount of work to schedule participants, observe them, and analyze the results. Consequently, the data tends to reflect only a few people and is mostly qualitative. These small numbers also make it hard to cover all of the possible tasks on a site. Furthermore, small samples are less convincing when asking management to make potentially expensive changes to a site. Lastly, a small set of participants may not find the majority of usability problems. Despite previous claims that around five participants are enough to find the majority of usability problems [14, 19], a recent study by Spool and Schroeder suggests that this number may be nowhere near enough [17]. Better techniques and tools are needed to increase the number of participants and tasks that can be managed for a usability test.

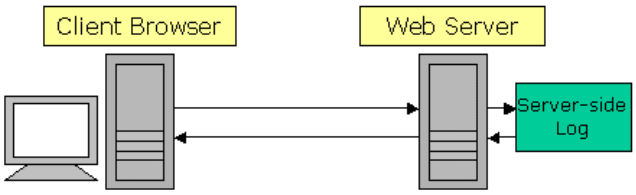

Figure 1. Server-side logging is done on the web server, but the data is available only to the owners of the server.

In contrast to traditional usability testing, server log analysis (see Figure 1) is one way of quantitatively understanding what large numbers of people are doing on a website. Nearly every web server logs page requests, making server log analysis quite popular. In fact, there are over 90 research, commercial, and freeware tools currently available [1]. Server logging also has the advantage of letting test participants work remotely in their own environments: instead of coming to a single place, usability test

WebQuilt source code and documentation can be downloaded at http://guir.berkeley.edu/projects/webquilt

participants can evaluate a website from any location on their own time, using their own equipment and network connection.

However, from the perspective of the web design team, there are two problems with server logs. The first is with deployment. Access to server logs are often restricted to just the owners of the web server. This can make it difficult to analyze subsites that exist on a server. For example, a company may own a single web server with different subsites owned by separate divisions. For the same reason, it is also impractical to do a log file analysis of a competitor's website. A competitive analysis is important in understanding what features people consider important, as well as learning what parts of your site are easy-to-use and which are not in comparison to a competitor's site.

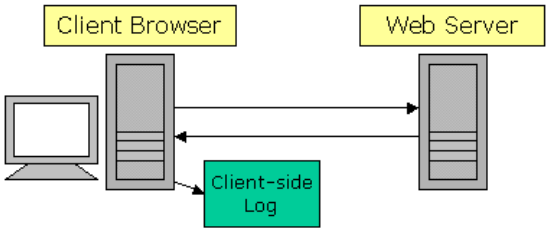

Figure 2. Client-side logging is done on the client computer, but requires special software running in the background or having a special web browser.

Client-side logging has been developed to overcome these deployment problems. In this approach, participants remotely test a website by downloading special software that records web usage (See Figure 2). However, client-side logging has two weaknesses. First, the design team must deploy the special software and have end-users install it. Second, this technique makes it hard to achieve compatibility with a range of operating systems and web browsers. What is needed is a logging technique that is easy to deploy for any website and is compatible with a number of operating systems and browsers.

Another problem with using either server- or client-side web logs to inform web design is that existing server log analysis tools do not help web designers understand *what* visitors are trying to do on a website. Most of these tools produce aggregate reports, such as "number of transfers by date" and "most popular pages." This kind of information resembles footsteps in the forest: you know someone has been there and where they went, but you have no idea what they were trying to do and whether they were successful. What is needed are logging tools that can be used in conjunction with known tasks, as well as sophisticated methods for analyzing and visualizing the logged data.

To recap, there are four things that could greatly streamline current practices in web usability evaluations:

1. A way of logging web usage that is fast and easy to deploy on *any* website
2. A way of logging that is *compatible* with a range of operating systems and web browsers
3. A way of logging where the *task* is already known
4. Tools for *analyzing* and *visualizing* the captured data

To address these needs, we developed WebQuilt, a tool for capturing, analyzing, and visualizing web usage. To address the first and second needs, we developed a proxy-based approach to logging that is faster and easier to deploy than traditional log analysis techniques (See Figure 3). This proxy has better compatibility with existing operating systems and browsers and requires no downloads on the part of end-users. It will also be easier to make compatible with future operating systems and browsers, such as those found on handheld devices and cellular phones.

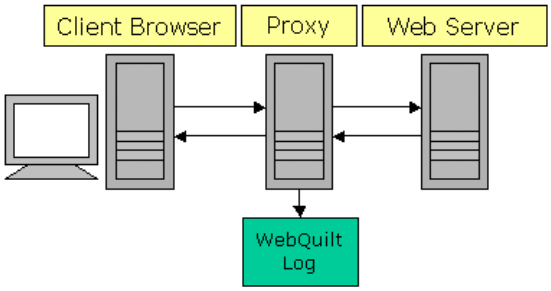

Figure 3. Proxy-based logging is done on an intermediate computer, and avoids many of the deployment problems faced by client-side and server-side logging.

To address the third need, we designed the proxy to be flexible enough that it can be used in conjunction with existing tools, such as those offering participant recruitment and online surveys. With these existing tools, we can know who the users are, what tasks they are trying to accomplish, and whether they were satisfied with how the site supported these tasks (for example, tools like these are provided by NetRaker [6]).

To address the fourth need, we designed a visualization that aggregates data from several test sessions and displays the web pages people viewed and the paths they took. However, we knew that we would not immediately have all of the solutions for analyzing the resulting data, so we also designed WebQuilt to be extensible enough so that new tools and visualizations could be implemented to help web designers understand the captured data.

WebQuilt is designed for task-based usability tests. Test participants are given specific tasks to perform, such as browsing for a specific piece of information or finding and purchasing an item. The WebQuilt proxy can track the participants' actions, whether they are local or remote. After a number of web usage traces have been captured, tools developed with the WebQuilt framework can be used to analyze and visualize the results, pointing to both problem areas and successful parts of the site. It is important that a task be attached to the test participants' interactions, because otherwise one must interpret the intent of visitors, something difficult to do based on web usage traces alone.

In the rest of this paper, we describe the architecture of WebQuilt and give a description of our current visualization tool. We then close with a discussion of related work and directions we plan to take in the future.

2. WEBQUILT ARCHITECTURE

WebQuilt is separated into five independent components: the Proxy Logger, the Action Inferencer, the Graph Merger, the Graph Layout, and the Visualization (See Figure 4). The *Proxy Logger* mediates between the client browser and the web server and logs all communication between the two. The *Action Inferencer* takes a log file for a single session and converts it into

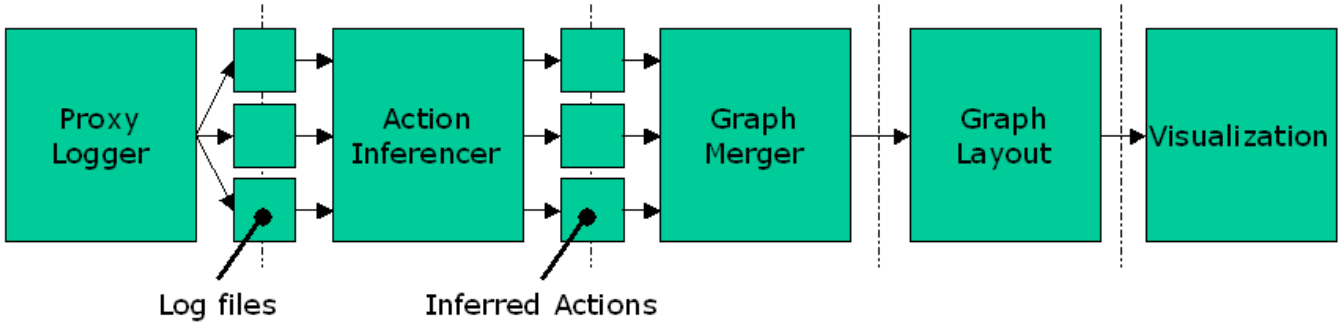

Figure 4. WebQuilt architecture overview. The Proxy Logger generates multiple log files, one log file per session. Each log file is processed by the Action Inferencer, which converts the log of page transactions into a log of actions. The results are combined by the Graph Merger, laid out by the Graph Layout, and visualized by the Visualization component.

a list of actions, such as "clicked on a link" or "hit the back button." The *Graph Merger* combines multiple lists of actions, aggregating what multiple people did on a website into a directed graph where the nodes represent web pages and the edges represent page requests. The *Graph Layout* component takes the combined graph of actions and assigns a location to each node. The *Visualization* component takes the results from the Graph Layout component and provides an interactive display.

Each of these components was designed to be as independent of each other as possible. There is a minimal amount of communication between each component, to make it as easy as possible to replace components as better algorithms and techniques are developed. In the rest of this section, we describe each of these components in detail.

2.1 Proxy Logger

The goal of the Proxy Logger is to capture user actions on the web. As a proxy, it lies between clients and servers, with the assumption that clients will make all requests through the proxy. In this section we first discuss problems with current logging techniques, describe how WebQuilt's proxy approach addresses these problems, and then continue with a description of the proxy's architecture.

2.1.1 Problems with Existing Logging Techniques

Currently, there are two common ways of capturing and generating web usage logs: server-side and client-side logging. Server-side logs have the advantage of being easy to capture and generate, since all transactions go through the server. However, there are several downsides to server-side logging, as pointed out by Etgen and Cantor [9] and by Davison [8]. One problem is that web caches, both client browser caches and Intranet or ISP caches, can intercept requests for web pages. If the requested page is in the cache then the request will never reach the server and is thus not logged. Another problem is that multiple people can also share the same IP address, making it difficult to distinguish who is requesting what pages (for example, America Online, the United States' largest ISP, does this). A third problem with server-side logging is with dynamically assigned IP addresses, where a computer's IP address changes every time it connects to the Internet. This can make it quite difficult to determine what an individual user is doing since IP addresses are often used as identifiers.

One alternative to gathering data on the server is to collect it on the client. Clients are instrumented with special software so that all usage transactions will be captured. Clients can be modified either by running software that transparently records user actions whenever the web browser is being used (as in [5]), by modifying an existing web browser (as in [18] and [13]), or by creating a custom web browser specifically for capturing usage information (as with [20]).

The advantage to client-side logging is that literally everything can be recorded, from low-level events such as keystrokes and mouse clicks to higher-level events such as page requests. However, there are several drawbacks to client-side logging. First, special software must be installed on the client, which end-users may be unwilling or unable to do. This can severely limit the usability test participants to experienced users, which may not be representative of the target audience. Second, there needs to be some mechanism for sending the logged data back to the team that wants to collect the logs. Third, the software is platform dependent, meaning that the software only works for a specific operating system or specific browser.

WebQuilt's logging software differs from the server-side and client-side approaches by using a proxy for logging instead. The proxy approach has three key advantages over the server-side approach. First, the proxy represents a separation of concerns. Any special modifications needed for tracking purposes can be done on the proxy, leaving the server to deal with just serving content. This makes it easier to deploy, since the server and its content do not have to be modified in any way.

Second, the proxy allows anyone to run usability tests on any website, even people that do not own that website. One can simply set up a proxy and ask testers to go through the proxy first. The proxy simply modifies the URL of the targeted site to instead go through the proxy. End users do not have to change any settings to get started. Again, this makes it easy to run and log usability tests on a competitor's site.

Finally, having testers go through a proxy allows web designers to "tag" and uniquely identify each test participant. This way designers can know *who* the tester was, *what* they were trying to do, and afterwards can ask them *how well* they thought the site supported them in accomplishing their task.

The proxy approach is also better than the client-side approach. It does not require any special software on the client beyond a web browser, making it faster to deploy. The proxy also makes it easier

to test a site with a wide variety of test participants, including novice users who may be unable or afraid to download special software. The proxy approach is also more compatible with a wider range of operating systems and web browsers than a client-side approach would be, as it works by modifying the HTML in a platform-independent way. Again, this permits testing with a more realistic sample of participants, devices, and browsers.

2.1.2 WebQuilt Proxy Logger Implementation

Our current implementation uses Java Servlet technology. The key to this component, though, is the log file format, as it is the log files that are processed by the Action Inferencer in the next step. To use our analysis tools, it actually does not matter what technologies are used for logging or whether the logger lies on the server, on a proxy, or on the client, as long as the log format is followed. Presently, the WebQuilt Proxy Logger creates one log file per test participant session.

Table 1 shows a sample log. The *Time* field is the time in milliseconds the page is first returned to a client, where 0 is the start time of the session. The *From TID* and the *To TID* fields are transaction identifiers. In WebQuilt, a transaction ID represents the Nth page that a person has requested. The From TID field represents the page that a person came from, and the To TID field represents the current page the person is at. The transaction ID numbers are used by the Action Inferencer for inferring when a person used the browser back button and where they went back.

The *HTTP Response* field is just the response from the server, such as "200 ok" and "404 not found." The *Link ID* field specifies which link was clicked on according to the Document Object Model (DOM). In this representation, the first link in the HTML has link ID of 0, the second has link ID of 1, and so on. This data is useful for understanding which links people are following on a given page. The last field is the *Current URL* field, which represents the current page the person is at.

This log format supports some of the same things that other log formats do. For example, the first row shows a start time of 20 msec and the second row 17825 msec. This means that the person spent about 17 seconds looking at the page http://www.yahoo.com. However, this format supports two things that other tools and formats do not. The first is the Link ID. Without this information, it can be difficult to tell which link a person clicked on if there are redundant links to the same page, which is a common practice in web design. This can be important in understanding which links users are following and which are being ignored. The second is exactly where a person used the back button. The highlighted cells in Table 1 show an example of where the person used the back button to go from transaction ID of 3 back to transaction ID of 1, and then forward again, this time to a different destination.

The proxy works by dynamically modifying all requested pages in four ways:

- Link URLs are modified to go through the proxy
- Transaction IDs are added to link URLs
- Link IDs are added to link URLs
- Base HREF is added to point to the original site

The proxy modifies all link URLs in a page to use the proxy again on the next page request. Thus, once a person has started to use the proxy, all of the links thereafter will automatically be rewritten to continue using the proxy. The proxy also adds Transaction IDs to each link. Again, Transaction IDs represent the Nth page that a person has requested. Embedding the transaction ID into a link's URL lets the proxy identify exactly what page a person came from. Link IDs are also added to each link URL. This allows the proxy to identify exactly which link in the page a person clicked on. Lastly, a Base HREF tag is added to the top of the page, in order to display images correctly.

There are two ways a person can start using the proxy. The first is by requesting the proxy's default web page and submitting a URL to the proxy (See Figure 5). The other way a person could start using the proxy is by using a link to a proxied page. For example, suppose you wanted to run a usability study on Yahoo!'s website.

Time	From TID	To TID	HTTP Response	Link ID	Current URL
20	0	1	200	-1	http://www.yahoo.com/
17825	1	2	200	94	http://docs.yahoo.com/info/
22302	2	3	200	59	http://docs.yahoo.com/info/pr/
24056	1	4	200	26	http://dailynews.yahoo.com/

Table 1. Sample WebQuilt log file in tabular format. The highlighted cells show where a person went back from the 3rd requested page to the 1st requested page, and then forward again.

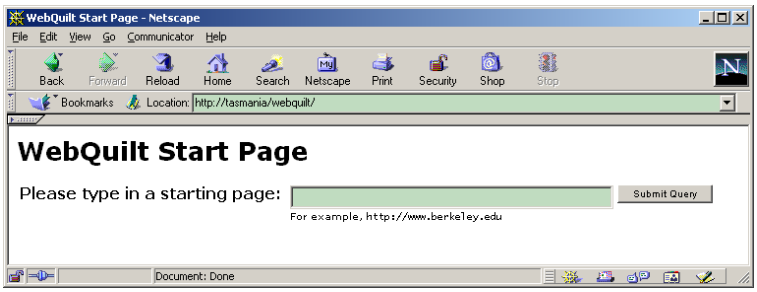

Figure 5. Default page for the WebQuilt proxy. The proxy will retrieve and dynamically modify the URL that is entered.

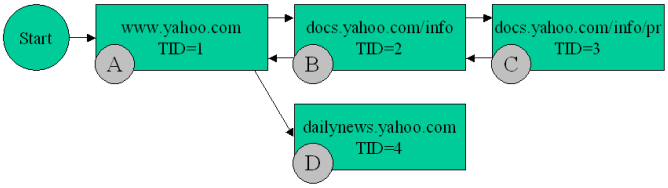

Figure 6. A graphical version of the log file in Table 1. The letters 'A', 'B', 'C', and 'D' are for this graph only and are not part of the log file.

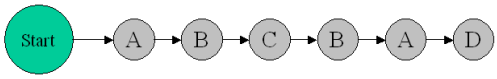

Figure 7. One possible way of interpreting the log file in Table 1. This one assumes that a person repeatedly hit the back button before clicking on a new link.

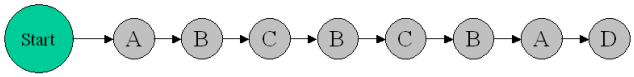

Figure 8. Another way of interpreting the log file in Table 1. This one assumes that a person uses the back and forward buttons a few times before clicking on a new link.

If the proxy's URL was:

http://tasmania.cs.berkeley.edu/webquilt,

then participants could just use the following link:

http://tasmania.cs.berkeley.edu/webquilt/redirect.jsp?replace=http://www.yahoo.com.

This approach makes it easy to deploy the proxy, as the link can just be emailed to people. Again, we expect other tools to be used for recruiting participants and specifying tasks for them to do. The proxy is flexible enough that it can easily be used with such other usability evaluation tools.

The base case of handling standard HTTP and HTML is straightforward. However, there are many special cases that must be dealt with. For example, cookies are typically sent from web servers to client browsers. These cookies are sent back to the web server whenever a client browser makes a page request. The problem is that, for security and privacy reasons, web browsers only send cookies to certain web servers (ones in the same domain as the web server that created the cookie in the first place). To address this, the proxy logger manages all cookies for a user throughout a session. It keeps a table of cookies, mapping from users to domains. When a page request is made through the proxy, it simply looks up the user, sees if there are any cookies associated with the requested web server or page, and forwards these cookies along in its request to the web server.

Another special case that must be dealt with is secure HTTPS. HTTPS uses SSL (Secure Socket Layer) to secure page requests and page data. The proxy logger handles HTTPS connections by creating two HTTPS connections, one from the client browser to the proxy, and another from the proxy to the web server.

Currently, the proxy logger does not handle server-side image maps, JavaScript window popups, Java applets, and style sheets.

2.2 Action Inferencer

Action Inferencers transform a log of page requests into a log of inferred actions, where an action is currently defined as either requesting a page, going back by hitting the back button, or going forward by hitting the forward button. The reason the actions must be inferred is that the log generated by the proxy only captures page requests. The proxy cannot capture where a person uses the back or forward buttons to do navigation, since pages are loaded from the local browser cache.

WebQuilt comes with a default Action Inferencer, but the architecture is designed such that developers can create and plug in new ones. It should be noted that given our logging approach, the inferencer can be certain of when pages were requested and can be certain of when the back button was used, but cannot be certain of back and forward combinations.

As an example, figure 6 shows a graph of the log file shown in Table 1. Figure 7 shows how the default Action Inferencer interprets the actions in the log file. We know that this person had to have gone back to Transaction ID 1, but we don't know exactly how many times he hit the back and forward buttons. Figure 7 shows what happens if we assume that the person went directly back from TID 3 to TID 1, before going on to TID 4.

Figure 8 shows another valid way of inferring what happened with the same log file. The person could have gone back and forth between TID 2 and 3 a few times before returning to TID 1.

2.3 Graph Merger

The Graph Merger takes all of the actions inferred by the Action Inferencer and merges them together. In other words, it merges multiple log files together, aggregating all of the actions that people did. A graph of web pages (nodes) and actions (edges) is available once this step is completed.

2.4 Graph Layout

Once the log files have been aggregated, they are passed to the Graph Layout component, which prepares the data for visualization. The goal of this step is to give an (x,y) location to all of the web pages. Since there are a variety of graph layout algorithms available, we have simply defined a way for developers to plug-in new algorithms. Currently, WebQuilt uses a simple force-directed placement algorithm as its default. This algorithm tries to place connected pages a fixed distance apart, and tries to spread out unconnected web pages at a reasonable distance.

2.5 Visualization

The final part of the WebQuilt framework is the visualization component. There are many ways of visualizing the information. We have built one visualization that shows the web pages traversed and paths taken (See Figures 9 and 10).

Web pages are represented by screenshots of that page as rendered in a web browser. Arrows are used to indicate traversed links and where people hit the back button. Thicker arrows indicate more heavily traversed paths. Color is used to indicate the average amount of time spent before traversing a link, with colors closer to green meaning short amounts of time and colors closer to red meaning longer amounts of time. Zooming is used to see the URL for a web page and to see a detailed image of individual pages (See Figure 10).

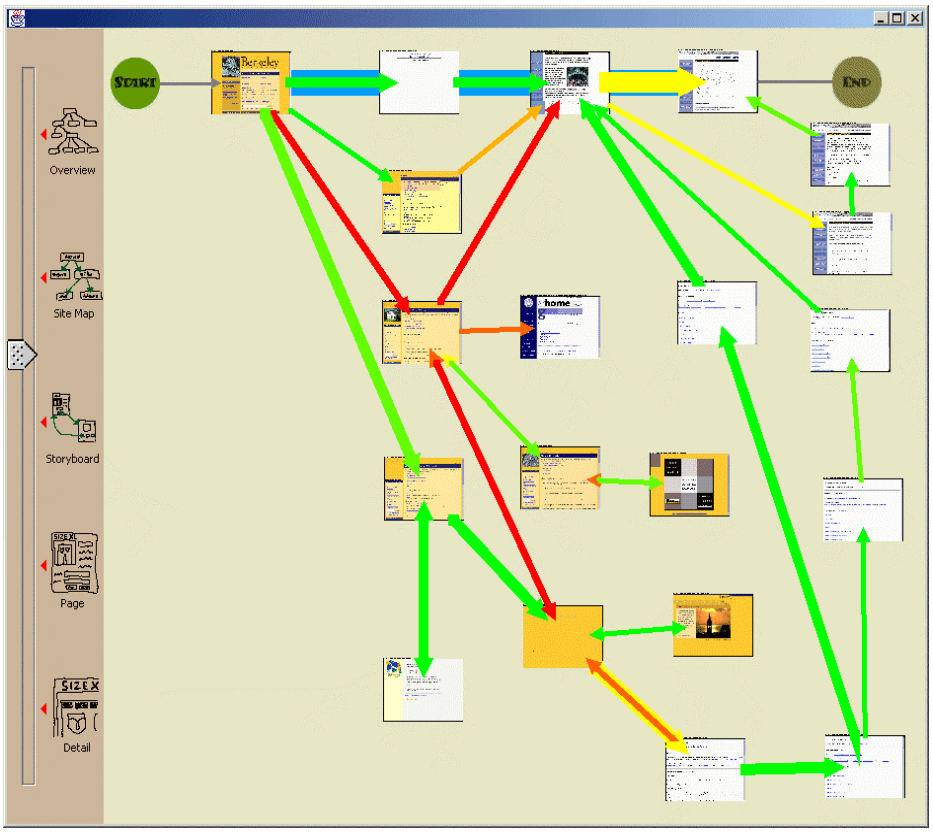

Figure 9. An example visualization of sixteen usage traces for a single defined task. The circle on the top-left shows the start of the task. The circle on the top-right shows the end of the task. Thicker arrows indicate more heavily traversed paths (i.e., more users). Red (darker) arrows indicate that users spent more time on a page before clicking a link.

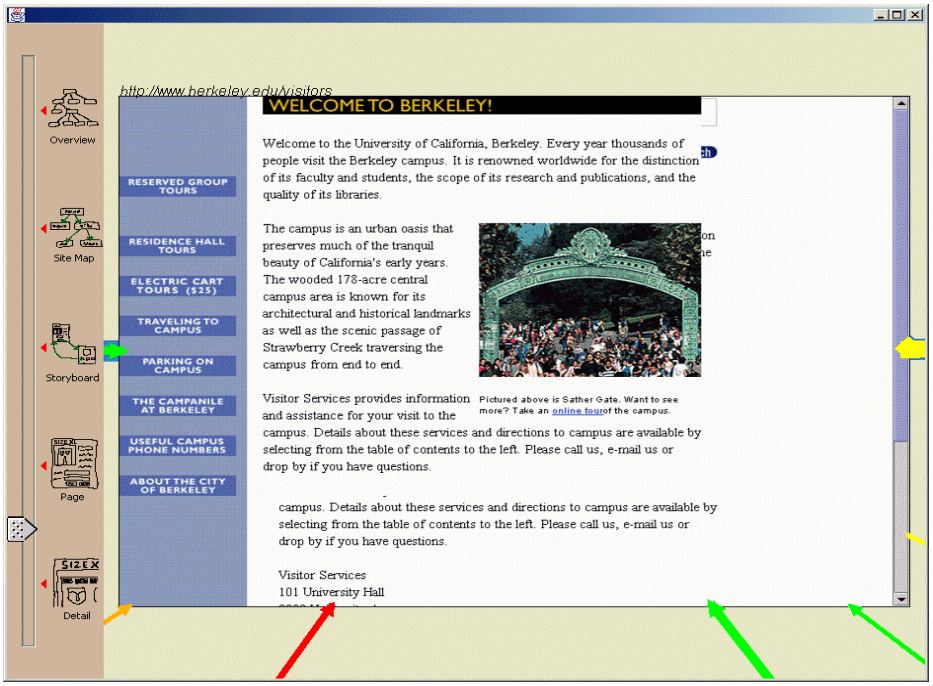

Figure 10. The zoom slider on the left is used to change the zoom level. Individual pages can be selected and zoomed-in on to the actual page and URL people went to.

Figure 9 shows an example visualization of sixteen usage traces, where the task was to find a specific piece of information on the website. The pages along the highlighted path at the top represent the optimal path. By looking at the thickness of the lines, one can see that many people took the optimal path, but about the same number of people took a longer path to get to the same place. One other page, at the bottom center, has a thick arrow going in but no arrows coming out. This indicates a page which many people visited but then used the back button to exit out.

There are also several red arrows (the dark colored ones), which indicate that people took a long time before going to the next page. However, none of the red arrows are along the optimal path, meaning that people that took that path did not have to spend a large amount of time to get to the next page.

3. RELATED WORK

In order to address some of the problems faced by client-side and server-side logging, the National Institute for Standards and Technology (NIST) has recently developed WebVIP [15]. Intended as a tool to help run usability tests, WebVIP makes a local copy of an entire site and instruments each link with special identifiers and event handling code. This code is activated when a link is clicked on. WebVIP shares some of the same advantages that WebQuilt's logging software has, such as better compatibility with existing operating systems and browsers (since only the HTML is modified) and some ability to run logged usability tests on sites one does not own. However, WebQuilt's proxy approach to logging lets it work without having to download an entire site, which is unrealistic for many situations. WebQuilt avoids the problems of stale content and of invalid path specifications for complex sites, and also works with database-backed sites that dynamically generate HTML when page requests are made.

Another system that is similar to WebQuilt's logging software is Etgen and Cantor's Web-Event Logging Technique (WET) [9]. WET is an automated usability testing technique that works by modifying every page on the server. It can automatically and remotely track user interactions. WET takes advantage of the event handling capabilities built into the Netscape and Microsoft browsers. WET has more sophisticated event logging than WebQuilt currently supports, though there are plans for merging WET's advanced event handling capabilities into WebQuilt (see Future Work section). However, again, WebQuilt differs with its proxy approach, which again does not require ownership of the server and requires no changes to the server. These last two advantages are important when trying to accomplish web evaluations – the designers and usability team might not be *allowed* to make changes to or be given access to a production server.

Usability log visualization has a long history. Guzdial and colleagues review and introduce several desktop-based systems in [10]. There are other recent visualizations that use the notion of paths. For example, the Footprints web history system [22] displays aggregate user paths as hints to what pages have been followed by other people. VISVIP [7] extends the work in WebVIP and shows individual paths overlaid on top of a visualization of the website.

Vividence Clickstreams [21] is a commercial usability tool for visualizing individual and aggregate user paths through a website. However, it uses client-side logging and has all of the problems associated with that technique. Furthermore, WebQuilt's visualization differs in that it combines aggregate path information with designated optimal paths, to make it easier to see which pages people had trouble with. WebQuilt also uses a zooming interface to show different portions of the website, including screenshots of individual pages to provide better context.

The closest visualization work is QUIP [11], a logging and visualization environment for Java applications. WebQuilt builds on QUIP by extending the logging and visualization to the web domain. WebQuilt also adds in zooming, and uses screenshots of web pages for detail instead of abstract circles.

4. FUTURE WORK

One important next step we are working on is capturing a richer set of user interactions. For example, client-side logging can capture such things as when a person hit the back or forward button. With this kind of information, the Action Inferencer could be bypassed entirely. We are currently investigating the approach used by WET [9], which captures low-level events on the Mozilla browser and on Microsoft's Internet Explorer browser using JavaScript.

We are also looking at additional visualizations for displaying the traces. There has been some work done in visualizing server logs [4, 12, 16] as well as visualizing individual and aggregate user paths [7, 21, 22]. We plan to re-implement some of the ideas demonstrated by these visualizations, and add in interactions and visualizations that are more useful for web designers. As one example, a web designer could designate one path as optimal, highlight it in the visualization, and then see how people deviated from that path. As another example, the visualization in Figure 10 could be modified such that when zoomed in, the arrows could be re-anchored to show exactly which link was clicked on from a given page. This is an example of semantic zooming [2], where the details of the visualization change depending on zoom level.

5. CONCLUSIONS

We have described WebQuilt, an extensible framework for helping web designers capture, analyze, and visualize web usage where the task is known. WebQuilt's proxy-based approach to logging overcomes many of the problems encountered with server-side and client-side logging, in that it is fast and easy to deploy, can be used on any site, can be used with other usability tools such as online surveys, and is compatible with a wide range of operating systems and web browsers.

We have also described the architecture for WebQuilt, and shown how new algorithms and visualizations can be built using the framework. Again, we knew that we would not have all the solutions for analyzing and visualizing the captured data, so the system must be extensible enough so that new tools can be easily built. We have demonstrated one simple zooming interface for displaying the aggregated results of captured web traces, and are currently building more sophisticated visualizations and interactions for understanding the data.

6. ACKNOWLEDGMENTS

We would like to thank James Lin and Francis Li for their feedback during the development of WebQuilt. We would also

like to thank Sarah Waterson, Jeff Heer, Kevin Fox, and Andy Edmonds for their ideas and work on improving portions of WebQuilt. Lastly, we would like to thank NetClue for providing us with a copy of their Java web browser component.

7. REFERENCES

[1] Access Log Analyzers. http://www.uu.se/Software/Analyzers/Access-analyzers.html

[2] Bederson, B.B. and J.D. Hollan. Pad++: A Zooming Graphical Interface for Exploring Alternative Interface Physics. In *Proceedings of the ACM Symposium on User Interface Software and Technology*: UIST '94. Marina del Rey, CA. pp. 17-26, November 2–4, 1994.

[3] Beyer, H. and K. Holtzblatt, *Contextual Design: Defining Customer-Centered Systems*. San Francisco: Morgan Kaufmann, 1998.

[4] Chi, E., J. Pitkow, J. Mackinlay, P. Pirolli, R. Gossweiler, and S. Card. Visualizing the Evolution of Web Ecologies. In *Proceedings of ACM CHI Conference on Human Factors in Computing Systems*. pp. 400-407, 1998.

[5] Choo, C.W., B. Detlor, and D. Turnbull. A Behavioral Model of Information Seeking on the Web – Preliminary Results of a Study of How Managers and IT Specialists Use the Web. In *Proce*edings of 1998 ASIS Annual Meeting, 1998. http://choo.fis.utoronto.ca/FIS/ResPub/asis99/default.html

[6] NetRaker Corporation. NetRaker Suite. 2001. http://netraker.com/

[7] Cugini, J. and J. Scholtz. VISVIP: 3D Visualization of Paths through Web Sites. In *Proceedings of International Workshop on Web-Based Information Visualization* (WebVis'99). Florence, Italy. pp. 259-263. IEEE Computer Society, September 1-3, 1999.

[8] Davison, B. Web Traffic Logs: An Imperfect Resource for Evaluation. In *Proceedings of Ninth Annual Conference of the Internet Society* (INET'99). San Jose, June 1999.

[9] Etgen, M. and J. Cantor. What Does Getting WET (Web Event-Logging Tool) Mean for Web Usability? In *Proceedings of Fifth Human Factors and the Web Conference,* 1999.

[10] Guzdial, M., P. Santos, A. Badre, S. Hudson, and M. Gray, Analyzing and Visualizing Log Files: A Computational Science of Usability. GVU Center TR GIT-GVU-94-8, Georgia Institute of Technology, 1994.

[11] Helfrich, B. and J.A. Landay, QUIP: Quantitative User Interface Profiling. 1999. http://home.earthlink.net/~bhelfrich/quip/

[12] Hochheiser, H. and B. Shneiderman, Understanding Patterns of User Visits to Web Sites: Interactive Starfield Visualizations of WWW Log Data. Technical Report ncstrl.umcp/CS-TR-3989, University of Maryland, College Park, February 1999.

[13] Ergosoft Laboratories, ErgoBrowser. 2001. http://www.ergolabs.com/ergoBrowser/ergoBrowser.htm

[14] Nielsen, J. and T. Landauer. A mathematical model of the finding of usability problems. In *Proceedings of ACM INTERCHI '93*. Amsterdam, The Netherlands. pp. 206-213, 1993.

[15] NIST, WebVIP. 1999. http://zing.ncsl.nist.gov/webmet/vip/webvip-process.html

[16] Pitkow, J. and K. Bharat. WebViz: A Tool for World-Wide Web Access Log Analysis. In *Proceedings of First International Conference on the World-Wide Web,* 1994.

[17] Spool, J. and W. Schroeder. Testing Web Sites: Five Users is Nowhere Near Enough. In *Proceedings of ACM CHI Conference on Human Factors in Computing Systems*. Seattle, WA, 2001.

[18] Tauscher, L.M., Evaluating History Mechanisms: An Empirical Study of Reuse Patterns in WWW Navigation, MSc Dissertation, University of Calgary, Calgary, 1999. http://www.cpsc.ucalgary.ca/grouplab/papers/1996/96-Tauscher.Thesis/thesis.html

[19] Virzi, R.A., Refining the Test Phase of Usability Evaluation: How Many Subjects Is Enough? *Human Factors*, 1992. 34(4): p. 457-468.

[20] Vividence, Vividence Browser. 2000. http://www.vividence.com/resources/public/solutions/demo/demo-print.htm

[21] Vividence, Vividence Clickstreams. 2000. http://www.vividence.com/resources/public/solutions/demo/demo-print.htm

[22] Wexelblat, A. and P. Maes. Footprints: History-Rich Tools for Information Foraging. In *Proceedings of CHI '99*. Pittsburgh, PA. pp. 270-277, May 1999.