Exploration of End User Debugging in Web Application Development

Libby Johnson and Mary Beth Rosson, PhD

Pennsylvania State University
State College, Pennsylvania

August 2005

Introduction

Picture an owner of a bed and breakfast empowered with tools he can use to develop an interactive web site. He builds a site to advertise his bed and breakfast and allow guests to make reservations online. Picture yourself planning a nice vacation, visiting his site, and making reservations at the bed and breakfast. What if the calendar and reservation system is faulty? You might arrive at the bed and breakfast to discover you have an invalid reservation and there are no rooms available. What if the website is insecure? The credit card information you use to make your reservation can be accessed by others. Your financial information is vulnerable and your failed reservations hinder your vacation. The bed and breakfast owner is faced with an angry customer, has lost income from that customer, has a diminished business reputation, and may be liable for damage caused by credit card information stolen from his insecure site.

Research indicates that end users not trained in programming can create basic web applications if they are supplied with a high level development tool.[1, 2] Commercial tools such as Filemaker Pro, Microsoft FrontPage, and Macromedia Dreamweaver already enable end users to create web applications. However, Editor in Chief of "IEEE Software" Warren Harrison wonders about the viability of programming by end users because they have no formal training,

> "Can it be true that software manipulating my credit history could have been written by an accountant with no concept of software testing or development processes? How many e-businesses have failed because of lost orders or payments placed through a Web site written by a self-taught Perl or HTML "programmer" who is really a marketing assistant and has never heard of file locking?"[3]

The bed and breakfast scenario demonstrates small scale economic consequences for a business and client. However, the diversity of web applications and the power to reach large audiences across national borders bespeaks potential for far reaching consequences. Since end users are being empowered to create web applications, they must also be given the means to insure the quality of those applications. In addition to improved web application development tools, debugging methods and tools appropriate for end user programmers must be provided.

To support end user debugging activities, we must first understand end user debugging needs. This paper describes a "think aloud" study that explores the end user mental model of debugging web applications. By understanding end users' mental model of debugging, more effective tools and methodologies supporting web application debugging can be built.

Related Work

Research shows that debugging techniques and tools can be made considerably easier and more "natural", to the point that they can be successfully applied by end user programmers. "End-user Software Engineering" and "Natural Programming" are two areas of primary focus for research in improving debugging for end users.

Margaret Burnett, Curtis Cook, Gregg Rothermel, and others at Oregon State University hope to bring software engineering practices to end users without requiring them to adopt a professional programmer's mindset. They have termed the concept "End-user Software Engineering." Their research has successfully combined an interactive testing methodology, fault localization capabilities, interactive assertions, and motivational devices to help end users ensure correctness in their software.[4]

Burnett et al. designed a "What You See Is What You Test" (WYSIWYT) testing methodology to interactively communicate testing information to end users as they develop spreadsheets. Colored borders are used to indicate the degree to which a cell has been tested and a display indicates the percentage of the spreadsheet that has been tested. This visual feedback responds to changes the user makes, allowing the testing process to be integrated into the incremental development of the spreadsheet.[4, 5, 6, 7]

Once failures have been identified with the WYSIWYT methodology, visual fault localization techniques are used to help users locate the source of failure. To indicate likely faultiness, the interior of cells is tinted red – the darker the tint the more likely the cell contains a fault. The tinted cells reduce the search space needed for tracing dependencies from the failing cell to contributing cells.[7]

Visual feedback and motivational devices have been shown to encourage spreadsheet programmers to adopt good software engineering practices by using testing and debugging tools they are being supplied. For example in one study, users could place X marks in spreadsheet cells as a debugging tool to indicate they noticed a failure. The X marks were used to generate fault localization feedback by tinting cells red according to fault likelihood. The study found that a majority of users perceived benefit from placing the X marks and went on to place more X marks. Additionally, fault localization feedback was shown to cause users to abandon ad hoc searching when debugging and adopt a more successful dataflow strategy.[7] Finally, assertions were added to the arsenal of spreadsheet tools developed so far. A surprise-reward-explain strategy enabled and motivated end users to incorporate assertions into their spreadsheet programs without prior knowledge or even an explanation of the use of assertions.[6]

While end user software engineering works to bring software engineering practices to nonprofessional programmers, a second area of study intends to make programming processes more "natural". For Myers, Pane, and Ko, making programming more "natural" means aligning the process more closely with how a non programmer would approach it. They propose that programming tools and environments should be natural because programming is the process of translating a human idea into terms a computer can understand. Naturally, the closer the language and environment of the computer is to the original idea; the easier it is to translate.[8]

As part of the "Natural Programming" effort, Andrew Ko and Brad Myers have worked on methods for making debugging activities more natural for programmers. They conducted studies that found that programmers tend to ask *why did* and *why didn't* questions when debugging. Studies also found that incorrect hypothesis about causes of failure inhibit a programmer's ability to successfully debug a program. For example in an event based language called Alice, 50% of errors were caused by false assumptions made by programmers as they were debugging existing errors.[9] The study findings lead Ko and Myers to develop a new debugging paradigm called Interrogative Debugging. The goal of Interrogative Debugging is to allow programmers to ask questions about program behavior thereby supporting the hypothesizing activities of debugging. The Whyline debugging interface successfully implements Interrogative Debugging and allows programmers to test their assumptions by asking *why did* or *why didn't* questions about their program's behavior. Answers about a program's behavior based on run time data prevent a programmer from spending time debugging based on a false assumption and possibly introducing more errors.

Whyline illustrates the success of Natural Programming methods of debugging. Ko and Myers found that Whyline made debugging almost 8 times faster. The decrease in debugging time allowed programmers to complete 40% more tasks than programmers not using Whyline.[9]

The research just discussed demonstrates the potential for end user debugging in the dataflow paradigm of spreadsheets and the event based language (Alice) that Whyline was applied to. Brown et. al demonstrates that the same ideas used for spreadsheets can be generalized to visual programming languages as well[10]. However, can debugging methodologies be developed to reduce the complexities of web application debugging to a level appropriate for end users?

Rode, Howarth, Pérez-Quiñones, and Rosson explored the current state of web application debugging by analyzing commercial web development tools marketed to end users. The tools use three methods to help end users address errors. The first method is to help users avoid errors by providing a constrained set of options and preventing direct access to underlying implementation. In the second method, the tool helps identify errors and provides error messages. However, tools with external application servers - such as Microsoft FrontPage and Macromedia Dreamweaver - are limited in detecting errors because their code execution depends heavily on server configuration. Method three of addressing errors provides a built in debugger to step through code execution. Unfortunately, debuggers may be difficult for end users to understand because they require knowledge of how code executes.[11]

The three methods described for end user debugging of web applications present opportunities for improvement. None of the methods provide the interactive, visual means for identifying failures and locating faults that have proven helpful for debugging spreadsheets, event based languages, and visual programming languages. Before researchers try to apply these debugging techniques to web applications, it is important to determine if they are appropriate. To this end, our study will examine the needs end users have in debugging web applications.

Study

To investigate end user debugging approaches for web applications we conducted a "think-aloud" study with 6 end users. The end users were asked to enhance a website, in which bugs were planted, using Microsoft FrontPage. FrontPage was chosen because it is a popular web application tool marketed to end users.

Procedure

Sessions were conducted one-on-one between an examiner and a subject. The subject was given a scenario in which he was hired to maintain a web site. The scenario gave four enhancement tasks and asked the subject to fix any bugs in the site and make any changes that would increase the site's usability. The subject was given time to read the scenario and ask any questions. The subject was asked to vocalize his thought process as he worked. A session was started with two windows open. One window contained the browser Internet Explorer for viewing the site. The other window contained the live site opened in FrontPage so that users could make changes directly instead of having to publish to the server. Data was obtained through screen and audio recordings, the examiner's observations, and a post session questionnaire.

Web site

The web site used for the study was modeled after a web site for a slide library at a university art department. The model was chosen as being representative of a web site that an end user without programming experience would be likely to create or maintain using Microsoft FrontPage. The web site home page provides an introduction to the slide library and a menu of

links to a page explaining library services, a database of the library's slide collection, faculty pages, and a page of contact information. Figures 1 and 2 illustrate the website.

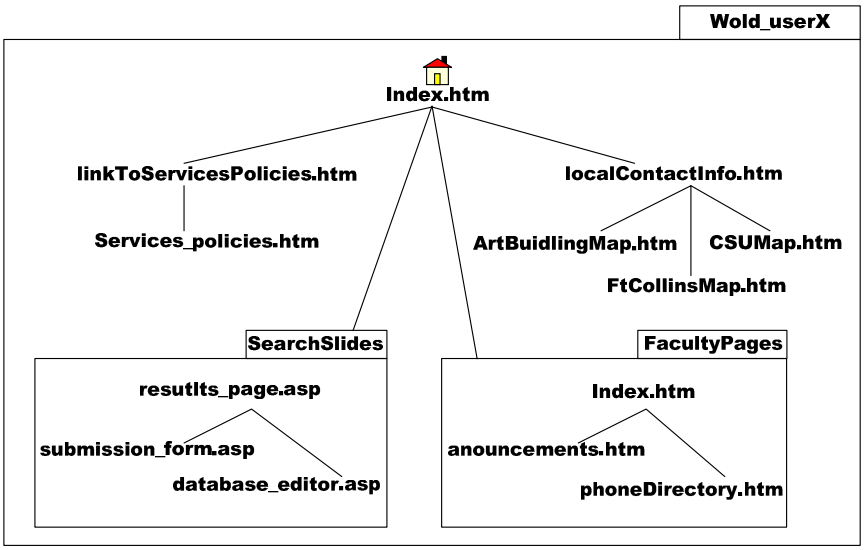


Figure 1. Architecture of the web site used in the FrontPage Study. Wold_userX is the site folder. FacultyPages and SearchSlides are sub folders. Edges in the graph represent links. Nodes are the names of web pages.

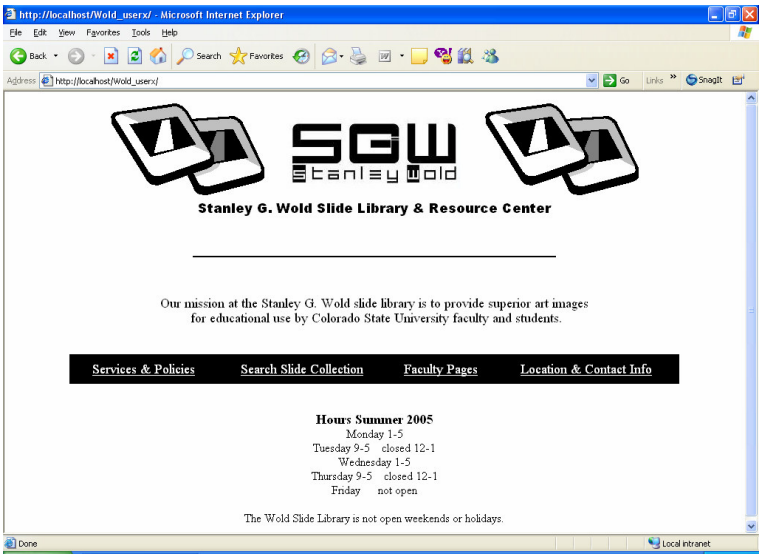
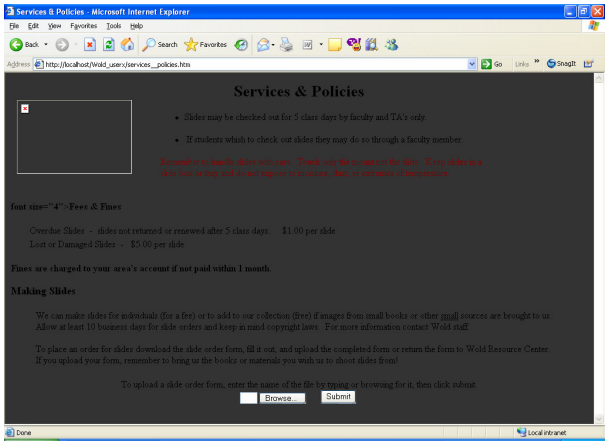
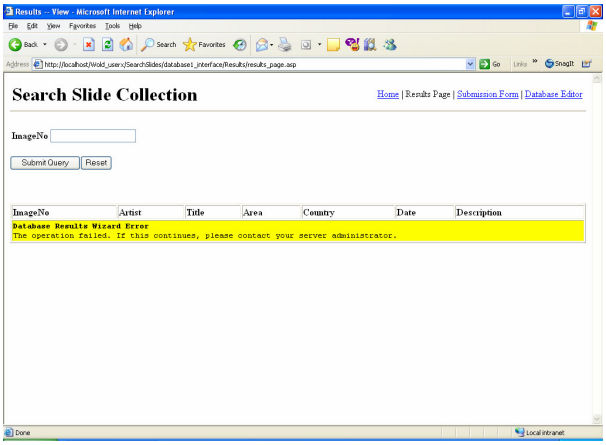


Figure 2 (A –C). Screen shots of web site. 2-A. Index.htm



2-B. services_policies.htm



2-C. results_page.asp

Bugs

To explore user's debugging approaches 9 faults were inserted into the site. Informal research using the web and FrontPage discussion groups helped target simulations of realistic faults in a variety of common areas – syntax, navigation, usability, dynamic form function. Having a variety of faults provides a more realistic situation as well as opportunity to see how users respond to different faults. Table 1 summarizes faults used in the study.

Faults Planted in FrontPage study Web Site			
No.	**Type of Fault**	**Description**	**Category**
1	Incorrect link	Link to faculty pages incorrect.	Static, functional, navigation
2	Image does not display	Image on services_policies.htm won't display.	Static, functional, syntax (image name misspelled)
3	Form-database interaction	Slide collection search does not work correctly.	Dynamic, functional, runtime
4	Poor use of color	Sevices_policies.htm is hard to read because background is too dark.	Static, usability, design
5	Html shows	Html shows on services_policies.htm.	Static, functional, syntax
6	Poor form design	Browse field of file upload form too small.	Static, usability, design
7	Shortest Path	Unnecessary page in path to services_policies.htm.	Static, usability, design
8	Html shows	Html shows on localContactInfo.htm.	Static, functional, syntax
9	Form function	File Upload Form does not work	Dynamic, functional, runtime

Table 1.

A good illustration of inserted faults appears on the site's "Services & Policies" page (figure 3). The page is too dark – a usability error. An image is not displaying and html code shows on the page – syntax errors. The form field at the bottom of the page is too short (another usability issue) and, least obvious, the form does not function.

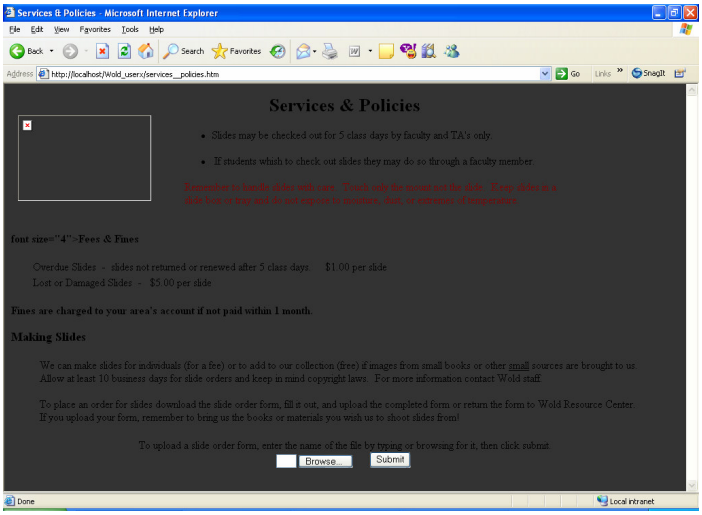


Figure 3. Example of planted faults.

Tasks

End users are likely to debug in the process of accomplishing other tasks. A traditional debugging study approach, requesting subjects to debug a web site, would not realistically capture end user debugging practices. Additionally, we hope to gain insight into how end users divide their time between accomplishing a task and debugging. Therefore, subjects were given four enhancement tasks and asked to fix any bugs in the site.

The four tasks were modeled after requests an employer might make for web site enhancements. The first task asked subjects to insert a link which allows site browsers to download a word document. The second task instructs participants to create an image map with a provided image. The third task requests that a new database be created for the library's video collection. The fourth task asks that a new field for searching by title be added to the form used for searching the library's database of slides.

Participants

We recruited from the general public with a flier and an email to a community FrontPage listserv. We also recruited from our local university population of professionals and students through listservs. Response to our recruiting efforts came from the university population and yielded 6 individuals relatively diverse in age, education, and experience. Three participants were students working on their bachelor's degree. One was a student who just finished a second master's degree. Two participants were professionals; one with a bachelor's and the other with an associate's degree. Ages of participants ranged from 21 to 42. Web development experience varied from students with only coursework to professionals who created web pages as part of their occupation. See table 2 for a summary of participants' profiles.

Profiles of FrontPage Study Participants					
User	**Student/ Professional**	**Gender**	**Age**	**Education**	**Experience**
1	Student	F	35	2nd Master	Self taught. Created websites for self, political campaign, 2 student organizations. Course in web design for teachers.
2	Professional	M	42	Bachelor	Maintained web pages as a lab manager. Web course offered by Penn. State University.
3	Student	M	21	high school	Work as student instructor tutoring people in FrontPage. Internship experience with FrontPage.
4	Student	F	22	high school	Self taught. Posted pictures in hand-coded html. Then took technology in the classroom, visual design on the web, & PGSIT 2000 course. Created websites for coursework.
5	Student	F	22	high school	Maintain websites for student clubs. Web art design course.
6	Professional	F	36	Associate	Creates basic web pages for finance department. Courses in Access, Excel, Word, Dreamweaver, FrontPage, PowerPoint, Computer Presentations, PageMaker.

Table 2.

Results

Site & Scenario

Most users in the study considered the site and scenario representative of what they might work on. The average user ranking of how realistic the site and scenario were is 5.00 on a scale of 1 to 7 with a standard deviation of 1.26. The one user who gave the site a below average rating did so because she felt that, "no art website would ever be black and white."

Site & scenario is realistic. Rated on a scale of 1-7							
User 1	User 2	User 3	User 4	User 5	User 6	Average	Standard Deviation
3	5	6	6	6	4	5.00	1.26

Table 3.

Tasks

The tasks were designed to start easy, increase in difficulty, and provide enough challenge to our user group. Rankings from the post session questionnaire (table 4) and user success rates for each task (table 5) indicate the study was appropriately designed in this regard. Subjects ranked task 1 as easiest and 5 out of 6 users completed it. Task two follows in rank and 4 out of 6 users completed it. Two users (3 and 5) that completed task 4 ranked it slightly less difficult than task 3. However, neither user completed the task correctly. Since neither user tested their implementation, they may have been unaware that they had not completed the task correctly. User 5 completed part of task 3 correctly, which seems to indicate that task 3 was easier for her despite her ranking it more difficult than task 4. User 6, who completed both task 3 and task 4 successfully, ranked task 3 as easier than task 4. Three out of six users did not get a chance to complete task 4 because they ran out of time.

Rating of Tasks on a scale of 1-7								
Task	**User 1**	**User 2**	**User 3**	**User 4**	**User 5**	**User 6**	**Average**	**Standard Deviation**
1	2	3	1	2	1	3	2.00	0.89
2	2	4	4	5	2	3	3.33	1.21
3	7	6	7	6	6	3	5.83	1.47
4	NA	NA	6	NA	5	4	5.00	1.00

Table 4.

Success Rate of Tasks S = successfully completed N = not successfully completed							
Task	**Description**	**User 1**	**User 2**	**User 3**	**User 4**	**User 5**	**User 6**
1	Insert a link to download a Word document	S	S	N	S	S – location inappropriate	S
2	Create an image map with 3 links	N	S	N – skipped	S	S	S
3	Create a database at a specified location & link it to the home page	Database – S Location – S Link – N	Database – S Location – N Link – Out of time	Database – N * Location – N Link - N	Database – S Location – N Link – Out of time	Database – S Location – N Link – N	Database – S Location – N Link – N
4	Add a search field to a database search form.	Skipped. Out of time	Out of time	Skipped. N	Out of time	N	S

Table 5. * The user generated a correct database interface but did not realize it was correct. He made another database interface that was not correct.

The primary goal of structuring the experiment around tasks is to see how end users deal with debugging as a part of web site development or maintenance. Questions we are investigating are:

Q1: For end users, does debugging hold greater, lesser, or equal significance compared to accomplishing a web development task?

Q2: How do end users divide their time between web development and debugging?

To answer these questions we combined responses from the end of session questionnaire with a breakdown of debugging activity by task for each user.

Results showed that the 6 users were evenly dispersed across the three possibilities of Q1 (greater, lesser, or equal) for debugging verses development. Users 1 and 4 placed greater emphasis on debugging. When asked how she divided her time between the assigned tasks and debugging, User 1 responds, "Started out w/ doing my own fixing – realized I should get to what I was supposed to do." User 4 states, "More time was spent on fixing bugs, because I would rather have a working, simple page than an enhanced, but still messed-up buggy page." Additionally, user 1 spent 8 minutes debugging and fixed 3 out of the 4 planted bugs she fixed before starting task 1. User 4 spent 28 minutes debugging and fixed all 6 of the planted bugs she fixed before starting the tasks. Users 2 and 3 place equal emphasis on debugging and their assigned development tasks, interleaving the two activities. User 2 says of his method, "I would correct errors as I found them while doing assigned task." User 3 states, "[I] Looked for major

bugs like noticeable extra text or incorrect links while performing the tasks." After a one minute exploration of the site, user 2 went to task one. Bugs were noticed in the site exploration but not fixed until after the user began working on task 1. Bugs were fixed as they were encountered in the tasks. User 3 started immediately with the tasks. Like user 2, user 3 fixed bugs as he encountered them. Users 5 and 6 placed greater importance on completing their tasks than on debugging. When asked how they divided their time between debugging and enhancing the website, both users discuss how they approached the tasks but fail to mention debugging. User 5 completed the list of tasks without noticing or fixing any of the planted bugs. Since there was time, the experimenter prompted her to debug and she then noticed four and fixed three planted bugs. User 6 noticed 4 bugs but fixed only 2, the fewest of all users. However, she was the only user to complete all 4 tasks correctly, demonstrating the tradeoff between completing the tasks and taking time to debug in the time constraint given.

Studying end users' division of focus between web development and debugging indicates how aware end users are about ensuring correctness in conjunction with web development. This study shows that end users can be quite diverse in how they divide their time between web development and debugging activities.

Participants

The skill set of the user group was slightly less sophisticated than we were originally anticipating for the study. For example, we expected these web developers to be familiar with creating image maps. However, 4 out of the 6 users did not know how to create an image map. Additionally, we discovered that the university population did not use dynamic features of FrontPage because the university deemed it a security risk to install the required server extensions. Therefore, participants were challenged beyond their current skill levels with tasks 3 and 4, which use features requiring FrontPage server extensions. Despite having no prior knowledge of FrontPage database features, 5 out of 6 users were able to successfully generate a database for task 3 (although the location and link to it tended to be incorrect) and one user, user 6, successfully completed task 4. Although 3 out of 4 users reported some experience with Microsoft Access, user 6 was the only one who reported near expert use of it (4 on a scale of 0-5). User 6's Access experience is likely a factor in her successful completion of tasks 3 and 4 which involve databases.

Noticing and Fixing of Planted Faults

From the order and frequency in which planted faults were noticed and fixed we can make observations about what types of faults users found and fixed easily and what type of faults presented considerable challenge. These observations can guide us in supporting end users' debugging efforts. Study participants found syntax errors the easiest faults to find and fix (aqua, table 6). The most noticed bug was an image that was not displaying due to a misspelling of the image name in html. The bug was noticed quickly (1^{st}, 2^{nd}, or 3^{rd} of 9 bugs) and was the only bug noticed by all 6 users. The next most frequently noticed and fixed faults where syntax errors causing html to show on web pages and poor use of color, a usability bug that makes web sites difficult to read. Four out of six users noticed and fixed these faults. Study participants were fairly adept at finding these visually obvious errors. Fewer participants discovered and corrected two navigational errors planted in the site (purple, table 6). Four out of six users noticed the extraneous page, a usability issue, inserted in a link between the home page and another page and three of them fixed it. Likewise four users noticed the incorrect link and three fixed it. Since

only half of users fixed the navigation errors, support for verifying the navigational structure of a web site may prove beneficial. Faults related to forms were the most difficult for users to notice and fix (grey, table 6). Only one user noticed and adjusted a too short form field that hindered the usability of a file upload form. Likewise, only one user discovered that the file upload form did not function, although she did not attempt to fix it. Three participants viewed the site in a browser and discovered a bright yellow message indicating a database error. The error message successfully drew their attention to the fault. However, users were required to "test" their site in a browser before the message was displayed. In addition the error message simply stated there was a problem and gave no clue as to the cause. In this situation, fault localization assistance would be advantageous. Two participants never viewed the site in a browser and therefore did not notice the error. One participant did not view the site in a browser but found the error in a properties box while working on task 4. Only the user who found the error in the properties box was able to fix it because the properties box is where the correction needed to be made.

Number of users that noticed and fixed bug (out of 6)			
No.	**Type of bug**	**Number of users that noticed the bug**	**Number of users that fixed the bug**
1	Incorrect link	4	3
2	Image does not display	6	4
3	Form-database interaction	4	1
4	Poor use of color	4	4
5	Html shows	4	4
6	Poor form design	1	1
7	Shortest Path	4	3
8	Html shows	4	4
9	Form function	1	0

Table 6.

Debugging Approaches

Study participants employed a variety of approaches to debugging (table 7). All users but one viewed the underlying html of their project when trying to locate and fix errors. Even though end users are given a high level web development tool and although they are not trained programmers, they still understand and make use of the underlying html code. Visual cues such as colored syntax highlighting proved helpful to users 2 and 3. While fixing bugs, users 2 and 3 introduced their own syntax errors. However, both realized it immediately when the color of the syntax highlighting changed and they were able to correct the errors. It was rather disappointing to see that all users did not test the site in a browser, especially since they were provided with an open browser window at the start of the study. However, the users who did not view the site in the browser explicitly previewed their site in FrontPage. They may have thought FrontPage preview was sufficient testing. However, FrontPage preview is limited in indicating errors for dynamic web pages because they require server side scripts. Additionally, FrontPage preview will not reproduce the incompatibility issues that may arise when viewing the site in many different browsers. Therefore, end user developers need to be encouraged to test their site in a variety of browsers. User 4 used one of the most interesting debugging approaches. She viewed

a diagram of the site's navigation structure that FrontPage provides (figure 4) to find misdirected and non-functioning links. Additionally, four out of six users looked at pop-up boxes summarizing object properties in their debugging attempts and one user used the spell check feature that FrontPage Provides.

Debugging Approaches of FrontPage Study Participants						
	User 1	User 2	User 3	User 4	User 5	User 6
Look at html code	X	X	X	X	X	-
Watch color highlighting of html code	-	X	X	-	-	-
Test in browser	X	X	-	X	-	X
Preview in FrontPage	-	-	X	-	X	X
Compare to working code/features in site	-	-	X	-	-	-
Look at diagram of navigation structure	-	-	-	X	-	-
Look at object properties	X	-	X	-	X	X
Use spell check	-	X	-	-	-	-

Table 7.

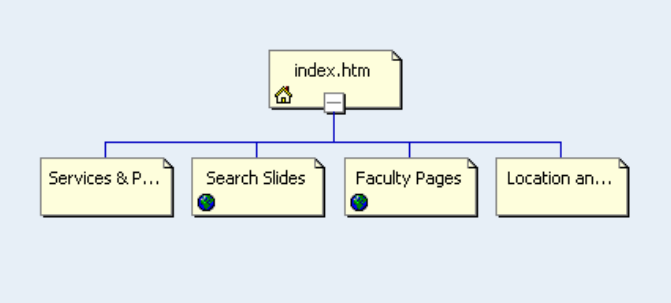


Figure 4. FrontPage diagram of web site navigation structure viewed by user 4 while debugging.

User Introduced Faults

Five out of six users introduced their own bugs into the web site while working on the development tasks and debugging (table 8). This argues for immediate and incremental fault identification and localization techniques so that users can spot errors as they introduce them. One such technique is the colored syntax highlighting discussed above that helped users 2 and 3 immediately recognize that they had introduced errors to the site. Such techniques need to be developed for areas additional to syntax. Out of a total of 9 faults introduced by users, 6 were incorrect or broken links. Since faulty links are such a prevalent error, it may be worth while taking the notion of a site navigation diagram, such as FrontPage currently employs, and developing it into a debugging tool. Additionally, users should be encouraged in some manner to test the web based forms they build. Two users introduced forms that did not function to their web site. This error is easily identified by testing but neither user tested the form after creating it.

Types of User Introduced Faults						
	User 1	User 2	User 3	User 4	User 5	User 6
Incorrect or broken link	X		X		X	X
Unintended element on page		X				
Form that does not function			X		X	
Incorrect form field name					X	

Table 8.

Conclusion

This study demonstrated the diverse emphasis users place on debugging in their web development activities. Some users emphasize having a correct web site while others fail to incorporate testing into their enhancement goals. We can not rely on all users to assign priority to ensuring the correctness of web applications given other development tasks. Debugging tools requiring users to initiate testing and debugging processes will not work for users who are focused on development activities and are less aware of ensuring a correct application. A means of automatically identifying and drawing users' attention to faults seems preferable for developers focused on task completion.

Our study participants actively used tools available to them in their debugging efforts, such as html syntax highlighting, a preview option, object properties, and spell check. User 4 even creatively employed a diagram of site navigation structure that was not necessarily meant for debugging. Users are willing to use debugging tools provided to them. It is a matter of discovering which current debugging tools are affective, how they may be improved, and what new tools can be added.

This study identified two areas of particular difficulty for end user web developers. Faults in dynamic forms were the hardest for users to find and fix. Incorrect hyperlinks were the most commonly overlooked fault as well as the fault most frequently introduced. These are areas in which to concentrate support for end user debugging.

Participants were asked to make enhancements to the web site that were technically beyond them. Therefore, it is interesting to note that rather than technical issues, all participants but one felt the most challenging aspect of enhancing someone else's web site is knowing what that person intends with the site. For example User 5 comments that, "Enhancing the design of a website isn't easy because your style might conflict with their vision of the website." User 2 talks about, "getting their[someone else's] concepts of look and feel translated to the page." This indicates that end users regard the technical aspects of web site development as a means to an end and are much more concerned with staying true to the intent of the web site. When designing debugging methods and tools for end users, it should be taken into account that end users are focused on accomplishing their goal. They do not have the same level of awareness of and interest in technical issues as professional developers do.

References

[1] Jochen Rode, Mary Beth Rosson, "Programming at Runtime: Requirements and Paradigms for Nonprogrammer Web Application Development," *Proc. IEEE Symposium on Human Centric Computing Languages and Environments*, Oct. 28-31, 2003, p. 23-30

[2] Jochen Rode, Mary Beth Rosson, Manuel Pérez Quiñones, "End User Development of Web Applications," End-User Development, eds. H. Lieberman, F. Paterno, and V. Wulf, F. Kluwer Academic Publishers, 2005

[3] Warren Harrison, "The Dangers of End-User Programming," *IEEE software*, Vol. 21, Issue 4, July-Aug. 2004, p. 5-7

[4][**] Margaret Burnett, Curtis Cook, Gregg Rothermel, "End-user development: End-user software engineering," *Communications of the ACM*, Vol. 47, Issue 9, September 2004, p. 53-58

[5] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Parabhakararao, M. Fisher II, M. Main, "Debugging & Finding Faults: End-User Software Visualizations for Fault Localization" *Proc. ACM symposium on Software visualization*, June 2003, p. 123-132

[6] A. Wilson, M. Bernett, L. Beckwith, O. Granatir, L. Casburn, C. Cook, M. Durham, G. Rothermel, "Issues in Software Development: Harnessing Curiosity to Increase Correctness in End-User Programming," *Proc. SIGCHI Conference on Human Factors in Computing Systems*, April 2003, p. 305-312

[7] S. Prabhakararao, C. Cook, J. Ruthruff, E. Creswick, M. Main, M. Durham, M.Burnett, "Strategies and behaviors of end-user programmers with interactive fault localization, "*Proc. IEEE Symposium on Human Centric Computing Languages and Environments*, Oct. 28-31, 2003, p. 11-22

[8][+] Brad Myers, John Pane, Andy Ko, "End-user Development: Natural Programming Languages and Environments," *Communications of the ACM*, Vol. 47, Issue 9, September 2004, p. 47-52

[9][A] A. J. Ko, and B. A. Myers, "Designing the Whyline: A Debugging Interface for Asking Questions About Program Failures," CHI 2004, Vienna, Austria, April 24-29, p.151-158.

[10] D. Brown, M. Burnett, G. Rothermel, H. Fujita, F. Negoro, "Generalizing WYSIWYT Visual Testing to Screen Transition Languages," *Proc. IEEE Symposium on Human Centric Computing Languages and Environments*, Oct. 28-31, 2003, p. 203-210

[11] J. Rode, J. Howarth, M. Pérez-Quiñones, M. Rosson, "An End-User Development Perspective on State-of-the-Art Web Development Tools," *Technical Report TR-05-03*, Department of Computer Science, Virginia Tech, 2004.