

Web Application Development by Nonprogrammers: User-Centered Design of an End-User Web Development Tool

Jochen Rode

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy
in
Computer Science and Applications**

Dr. Mary Beth Rosson, Co-Chair
Dr. Manuel A. Pérez-Quñones, Co-Chair
Dr. Stephen H. Edwards
Dr. Naren Ramakrishnan
Dr. Loren P. Rees

July 1st, 2005
Blacksburg, Virginia, USA

Keywords: end-user web application development, nonprogrammers, mental models

Copyright 2005 by Jochen Rode

Web Application Development by Nonprogrammers: User-Centered Design of an End-User Web Development Tool

Jochen Rode

Abstract

This work investigates entry barriers and approaches for facilitating end-user web application development with the particular focus on shaping web programming technology and tools according to end-users' expectations and natural mental models. My underlying assumption and motivation is that given the right tools and techniques even nonprogrammers may become successful web application developers. The main target audience for this research are "casual" webmasters without programming experience – a group likely to be interested in building web applications. As an important subset of web applications I focus on supporting the development of basic data collection, storage and retrieval applications such as online registrations forms, staff databases, or report tools. First I analyze the factors contributing to the complexity of web application development through surveys and interviews of experienced programmers; then I explore the "natural mental models" of potential end-user web developers, and finally discuss my particular design solutions for lowering entry barriers, as embodied by a proof-of-concept development tool, called Click. Furthermore, I introduce and evaluate the concept of "Design-at-Runtime" – a new technique for facilitating and accelerating the development-test cycle when building web-based applications.

Acknowledgments

First and foremost, I would like to thank my advisors Dr. Mary Beth Rosson and Dr. Manuel Pérez-Quiñones for their kind support and guidance. It was them who sparked my interest in research and helped me appreciate the value of looking beyond the horizon of the immediately practical. My gratitude also extends to Dr. Stephen Edwards, Dr. Naren Ramakrishnan, and Dr. Loren Rees – the other three members of my Ph.D committee, as well as to Dr. James Arthur, my first mentor at Virginia Tech. Furthermore, without the encouragement of my German mentors Dr. Thomas Pietsch and Dr. Harald Brandenburg at the University of Applied Sciences (FHTW) in Berlin I may not have embarked on a graduate career in the United States of America.

I want to acknowledge the Fulbright commission for granting me the opportunity to study in the USA. I am also thankful to the National Science Foundation who sponsored much of my research, allowed me to collaborate with other researchers and travel internationally to present our findings.

I truly enjoyed working together with Yogita Bhardwaj. Our conversations and her help with the implementation of various prototype tools were invaluable. Jonathan Howarth influenced my work in a similar way, helping me throughout analysis and design. Furthermore, I would like to express my gratitude to Julie Ballin at Pennsylvania State University who I had the privilege and pleasure to collaborate with at a number of occasions. Betsy Blythe was much more than just the supervisor of my half-time appointment at the Information Systems and Computing department. She truly cared for me as a person and made room for experimenting with research ideas. I also want to thank Erv Blythe, the head of the IT department, who supported the idea of end-user web development and my previous supervisors Susan Olivier and James Powell who challenged me with projects that lead to my interest in web application development. I feel indebted to all of my coworkers, especially Janice Gibb, Kaye Kriz, and Andrea Coles for their valuable feedback and emotional encouragement and equally to B. Collier Jones for his always honest, helpful, and inspiring criticism.

For their expert advice and support I owe much to my friends Gregorio Convertino, Ryan Richardson, Pat Lehane, Shad Gilley, Rob Capra, Kibum Kim, Shaadi El-Swaifi and many other fellow students, professors, and staff at Virginia Tech.

Finally, I want to thank my family; my parents for bearing with me for 24 years at home and then for caring for me while abroad; my brother Olaf for keeping me in touch with reality and strengthening my desires for exploring the world and slacking off once in a while; my grandfather and grandparents for their emotional support and the regular German-chocolate-filled yellow “care-packages”; my parents-in-law for being the exact opposite of the bad cliché, and the rest of my uncles, aunts, cousins and friends for not forgetting me. Above all, however, I want to thank my wonderful, beautiful, humorous and witty wife Andrea for her love and care and for bestowing me with miraculous food for thought, the soul, and the stomach.

Table of Contents

Abstract.....	ii
Acknowledgments	ii
Acknowledgments	iii
Table of Contents	v
List of Figures.....	x
List of Tables	xiii
List of Abbreviations	xv
1 Introduction.....	1
1.1 Problem and Vision.....	1
1.2 Motivation and Significance	3
1.3 Target Audience and Domain	4
1.4 Research Questions and Objectives	5
1.4.1 What are the main entry barriers to EUDWeb?	5
1.4.2 How do novice developers naturally think about web programming concepts? 6	
1.4.3 What are viable approaches for making web application development more accessible for nonprogrammers?	7
1.5 Research Overview and Outline	7
2 Related Work	9
2.1 Web Engineering	10
2.1.1 Studies of Web Development Practice.....	10
2.1.2 Model-driven Approaches to Data-intensive Websites.....	13
2.1.3 Languages and Tools for Building Web Applications	15
2.2 Psychology of Programming.....	20
2.3 End-User Development.....	21
2.3.1 Goals and Trade-off Between Ease-of-Use and Power.....	21
2.3.2 The Spreadsheet Paradigm and the Concept of Liveness	22
2.3.3 The Concept of Naturalness in End-User Programming.....	23
2.3.4 Visual Languages and Direct Manipulation.....	24
2.3.5 Cognitive Dimensions Framework	24
2.3.6 End-User Development for the Web.....	25

2.4	Commercial Web Development Tools.....	27
2.4.1	Professional Productivity Tools.....	27
2.4.2	Database-centric Tools.....	28
2.4.3	Online Site Builders and eCommerce Tools.....	29
2.4.4	End-User WYSIWYG Editors and Web Application Builders	29
2.4.5	A Review of State-of-the-Art Web Development Tools.....	30
2.5	Summary and Conclusions	35
3	Entry Barriers and Status-Quo in End-User Web Application Development... 38	
3.1	Survey and Interviews of Experienced Web Developers	38
3.1.1	Methods and Results	39
3.1.2	Discussion and Conclusions.....	53
3.2	Survey of Web Developers: From Amateurs to Professionals	57
3.2.1	Methods.....	57
3.2.2	Results.....	58
3.2.3	Discussion and Conclusions.....	66
3.3	Concepts and Components of Typical Web Applications	68
3.4	Summary and Conclusions	71
4	Mental Models of End-User Web Developers..... 74	
4.1	Exploring End Users' Concepts and Language Use	75
4.1.1	Participants and Methods	75
4.1.2	Results.....	78
4.2	Mental Models of Typical Web Development Concerns	80
4.2.1	Participants.....	81
4.2.2	Methods.....	82
4.2.3	Results.....	84
4.2.4	Summary and Conclusions.....	91
4.3	Summary and Conclusions	95
5	Click – A Web Application Development Tool for End Users 97	
5.1	Design-at-Runtime.....	97
5.2	Early Prototyping Efforts and Lessons Learned	100
5.2.1	FlashLight	100
5.2.2	Custom Extensions to Existing Tools	103
5.2.3	Click Prototype #1 and #2.....	104
5.3	Click's Development Paradigm and Key Features	107
5.4	Design Rationale.....	112

5.4.1	Introduction Video & Tutorial	115
5.4.2	Application Templates	116
5.4.3	Support for Opportunistic Development and Design-at-Runtime.....	116
5.4.4	Support for Continuous Workflow.....	117
5.4.5	To-Do List.....	117
5.4.6	Sensible Defaults and Strong Affordances.....	118
5.4.7	Context-sensitive Help	119
5.4.8	Sitemap	119
5.4.9	Domain Specificity	122
5.4.10	Session Layer	122
5.4.11	Database Layer.....	122
5.4.12	Security Layer	123
5.4.13	Input Validation Layer	124
5.4.14	Authentication Layer.....	124
5.4.15	Authorization Layer	125
5.4.16	High-level Components	126
5.4.17	Button Action Rules.....	129
5.4.18	Event-based Web Programming	129
5.4.19	Separation of Layout and Behavior.....	130
5.4.20	Templating	132
5.4.21	Parameter Passing and “Current Data Record”	132
5.4.22	Wizards	133
5.4.23	Pixel-based Positioning.....	133
5.4.24	“Global” Components.....	134
5.4.25	Layers of Programming Support and Gentle Slope of Complexity	136
5.4.26	Collaboration Support.....	138
5.4.27	Integrated Development and Runtime Environment.....	138
5.5	System Architecture and Implementation.....	139
5.6	Summary and Conclusions	142
6	Evaluation of Click	143
6.1	Formative Evaluations	143
6.1.1	Evaluation of Prototype #1	144
6.1.2	Evaluation of Prototype #2	146
6.1.3	Evaluation of Prototype #3	147
6.2	Summative Evaluation	148

6.2.1	Propositions.....	149
6.2.2	Participants.....	150
6.2.3	Methods.....	152
6.2.4	Results on the Overall Success	155
6.2.5	Results on the Problem of Complexity	156
6.2.6	Results on the Problem of Integration.....	161
6.2.7	Results on the Problem of Security.....	166
6.2.8	Results on the Problem of Feedback.....	168
6.2.9	Critical Incidents and General Observations.....	171
6.3	Summary and Conclusions	176
7	Conclusions and Future Work	178
7.1	Summary of Findings.....	178
7.1.1	What are the main entry barriers to EUDWeb?	178
7.1.2	How do novice developers naturally think about web programming concepts?	179
7.1.3	What are viable approaches for making web application development more accessible for nonprogrammers?.....	180
7.2	Summary of Research Contributions.....	181
7.3	Future Directions	182
	References.....	184
Appendix A	Survey of Virginia Tech Webmasters	194
A.1	IRB Approval.....	194
A.2	Survey Questionnaire and Summary Results.....	195
Appendix B	Interviews of Semi-Professional Developers.....	204
B.1	IRB Approval.....	204
B.2	Pre-Interview Questionnaire	205
Appendix C	Comprehensive Survey of Web Developers.....	215
C.1	IRB Approval.....	215
C.2	Questionnaire and Summary of Results.....	216
Appendix D	Mental Models Study 1	236
D.1	IRB Approval.....	236
D.2	Participants' Instructions	237
D.3	Screen Labeling Example	239
D.4	Screenshots of Example Application.....	240

Appendix E	Mental Models Study 2	245
E.1	IRB Approval.....	245
E.2	Scenarios	246
Appendix F	Click	251
F.1	Screenshots	251
Appendix G	Summative Evaluation of Click	256
G.1	IRB Approval.....	256
G.2	IRB Informed Consent.....	257
G.3	Online Screening Questionnaire	259
G.4	Study Procedure Instructions	261
G.5	Ride board Example Application Screenshots.....	262
G.6	Ride board Example Exploration Instructions.....	265
G.7	Help Instructions.....	266
G.8	Facilitators' Functionality Checklist.....	266
G.9	Post-study Survey Questionnaire.....	267
G.10	Visualizations of Participants' Development Timelines.....	271
Appendix H	Grant Information	274
Appendix I	Publications and Presentations	275
I.1	Peer-reviewed Full-length Conference Papers.....	275
I.2	Book Chapter, Abstract, Technical Reports	276
Vita	277

List of Figures

Figure 1: User-centered methods for building web development tools.....	8
Figure 2: EUDWeb is the cross-section of web engineering, psychology of programming, and EUD.....	9
Figure 3: Java code that outputs JavaScript code that outputs HTML code containing CSS	16
Figure 4: Example Flex application.....	18
Figure 5: Survey question targeted at exploring end users' needs for “interactive websites”.....	40
Figure 6: Virginia Tech webmasters reporting their reasons for not developing web applications themselves (N=40).....	42
Figure 7: Responses to question about problems in web application development (1=not a problem at all; 7=severe problem). The square markers show the mean of the responses from the survey (value is right of the square marker in italics; N=31). The round markers show the mean of the responses from the pre-interview questionnaire (value is left of round marker; N=10). In order to facilitate comparison, the survey responses have been scaled from a 1-5 scale to a 1-7 scale.....	43
Figure 8: Results from question 5: “The following question asks you to judge the value of these same 10 features in your web development projects, regardless of whether you have worked with them yet or not.” (N=314 to 318).....	60
Figure 9: 90% of responses to question 8 “What are the three things you like MOST about your primary web development tool?” were coded into 17 categories	62
Figure 10: 88% of responses to question 9 “What are the three things you like LEAST about your primary web development tool?” were coded into 16 categories. ...	63
Figure 11: Responses to Question 14 “How often do you experience problems with the following kinds of issues that sometimes arise in web development work? Please use a scale from 1 (one) to 5 (five) where 1 means hardly ever, and 5 means quite often.” (n=267 to 276).....	64
Figure 12: Two screenshots of example application used for MMODELS-1.....	76
Figure 13: Example of an annotated screenshot of the “Add Member” dialog from the member registration application (MMODELS-1).....	77
Figure 14: Example of a participant’s description of the behavior of the “Add Member” dialog from the member registration application (MMODELS-1).....	78

Figure 15: Scenario 1 of 9 as shown to each participant (MMODELS-2) 83

Figure 16: Defining button actions in FlashLight..... 100

Figure 17: Click prototype #1: an external WYSIWYG editor is used in conjunction with Click 105

Figure 18: Defining a “Register” button and associated action using the form-based UI of Click 107

Figure 19: A sitemap automatically generated by Click..... 120

Figure 20: Legend for Click's sitemap as shown in Click's user interface 121

Figure 21: Layers of Click's programming support that illustrate a “gentle slope of complexity” 136

Figure 22: Click's HTML frames setup 139

Figure 23: Click's system architecture and file system layout..... 140

Figure 24: Example of a specification used during the formative evaluation sessions of Click 144

Figure 25: Screenshot of the "Offer ride" page from the example application 152

Figure 26: Visualized timeline of participant 6’s behavior as derived from the activity log 169

Figure 27: IRB approval for survey of VT webmasters 194

Figure 28: IRB approval for interview study..... 204

Figure 29: IRB approval for comprehensive survey of web developers 215

Figure 30: IRB approval for MMODELS-1 236

Figure 31: Screen labeling example provided to study participants..... 239

Figure 32: Screenshot of example application from MMODELS-1: Login..... 240

Figure 33: Screenshot of example application from MMODELS-1: View all members 241

Figure 34: Screenshot of example application from MMODELS-1: Add member..... 242

Figure 35: Screenshot of example application from MMODELS-1: View member..... 243

Figure 36: Screenshot of example application from MMODELS-1: Search..... 244

Figure 37: IRB approval for mental models study 2..... 245

Figure 38: Defining a “Register” button and associated action using the form-based UI of Click 251

Figure 39: The "Database" view of Click allows the modification of database schema and data 252

Figure 40: The "Sitemap" view of Click showing the example "Ride board" application 252

Figure 41: The property dialog of the "Text field" component 253

Figure 42: Parts of the properties dialog of the "Dynamic table" component 254

Figure 43: Dialog to specify action rules 255

Figure 44: IRB approval for summative evaluation of Click 256

Figure 45: IRB informed consent form for summative evaluation of Click..... 258

Figure 46: Screenshot of ride board example application: Home..... 262

Figure 47: Screenshot of ride board example application: Search 262

Figure 48: Screenshot of ride board example application: Offer ride 263

Figure 49: Screenshot of ride board example application: Login/Logout 263

Figure 50: Screenshot of ride board example application: Details..... 264

Figure 51: Click - summative evaluation: Visualization of development timeline from participant #1 271

Figure 52: Click - summative evaluation: Visualization of development timeline from participant #2 271

Figure 53: Click - summative evaluation: Visualization of development timeline from participant #3 272

Figure 54: Click - summative evaluation: Visualization of development timeline from participant #4 272

Figure 55: Click - summative evaluation: Visualization of development timeline from participant #5 273

Figure 56: Click - summative evaluation: Visualization of development timeline from participant #6 273

List of Tables

Table 1: Scenario: Anna’s Ventures into Web Application Development.....	2
Table 2: Green and Petre's cognitive dimensions framework of notations.....	25
Table 3: Guidelines for EUDWeb tools derived from our review.....	33
Table 4: Summary of the findings and trends of the related work.....	35
Table 5: Virginia Tech webmasters reporting their needs for “interactive websites” a.k.a. web applications (number in brackets indicates the frequency of requests; N=67, with some respondents reporting needs in multiple categories).....	41
Table 6: Responses to questions asked in the pre-interview questionnaire on scales from 1-7.....	48
Table 7: Responses from 5 participants regarding their appreciation of Macromedia Dreamweaver MX as a web development tool.....	52
Table 8: Question 16: statements ranked from 1 (strongly disagree) to 5 (strongly agree)	61
Table 9: High-level components, concepts & functionality of typical basic web applications.....	69
Table 10: Web developers’ behaviors, barriers to development, and a “wish list for the dream tool”	71
Table 11: Examples of labels chosen by the participants of MMODELS-1. Numbers in brackets denote the number of participants who chose the particular label.	78
Table 12: The Mental Model of the “Prototypical” Novice Web Application Developer	95
Table 13: Click - Beginner's tutorial.....	108
Table 14: Mapping from problems to design solutions (the numbers in parenthesis represent the sections discussing the issue in detail; issues marked in bold are the focus of the summative evaluation).....	112
Table 15: Layout code for one web page of a simple conference registration application	131
Table 16: Behavior code for one web page of a simple conference registration application	131
Table 17: Click’s Novel Concepts and Features.....	142
Table 18: Example from a usability problem list as used during formative evaluation ...	145
Table 19: Propositions for the summative evaluation of Click.....	149
Table 20: Two questions about "web master knowledge" from participant selection questionnaire.....	151

Table 21: Question about "web programming knowledge" from participant selection questionnaire.....	151
Table 22: Participants from summative evaluation and their self-reported experience on the online pre-study selection questionnaire (1=no knowledge, 5=expert knowledge).....	151
Table 23: Excerpt of participant's 6 activity log (facilitator's logging of critical incidents in bold).....	154
Table 24: Times, critical incidents, participant's and facilitators' ratings from summative evaluation	156
Table 25: Question targeted at exploring participants' expectations towards state persistence	159
Table 26: Participants' ratings on frequency of use and usefulness of To-do list	163
Table 27: Participants' ratings on frequency of use and usefulness of sitemap	165
Table 28: Results of Click's formative studies and summative evaluation	176
Table 29: Questionnaire and summary results from survey of Virginia Tech webmasters	195
Table 30: Pre-Interview questionnaire of semi-professional web developers	205
Table 31: Comprehensive survey of web developers: Questionnaire and summary of results.....	216
Table 32: Participants' instructions for mental models study 1	237
Table 33: Nine scenarios from MMODELS-2.....	246
Table 34: Online screening questionnaire for formative and summative studies of Click	259
Table 35: Participant's instructions for summative study of Click	261
Table 36: Click - summative study: Ride board example exploration instructions.....	265
Table 37: Click - summative study: Help instructions.....	266
Table 38: Click - summative study: Facilitators' functionality checklist.....	266
Table 39: Click - summative study: Post-study questionnaire.....	267

List of Abbreviations

AJAX – Asynchronous JavaScript + XML
API – Application Programming Interface
ASP – Active Server Pages or Application Service Provider
CGI – Common Gateway Interface
CSS – Cascading Style Sheets
DB – Database
DBMS – Database Management System
DHTML – Dynamic Hypertext Markup Language
EUD – End-User Development
EUDWeb – End-User Development of Web Applications
HCI – Human-Computer Interaction
HTML – Hypertext Markup Language
HTTP – Hypertext Transfer Protocol
IDE – Integrated Development Environment
IT – Information Technology
JSP – JavaServer Pages
LDAP – Lightweight Directory Access Protocol
MVC – Model-View-Controller
PHP – PHP: Hypertext Pre-processor (recursive acronym)
RPC – Remote Procedure Call
SQL – Structured Query Language
UI – User Interface
UIML – User Interface Markup Language
UML – Unified Modeling Language
URL – Uniform Resource Locator
WML – Wireless Markup Language
WWW – World Wide Web
WYSIWYG – What You See Is What You Get
XAML – eXtensible Application Markup Language
XML – eXtensible Markup Language
XSL - eXtensible Stylesheet Language
XUL – XML User Interface Language



Chapter 1

Introduction

1.1 Problem and Vision

The World-Wide-Web has become an important platform for interactive applications. Web-based calendars and forums facilitate collaboration; e-commerce web sites enable the convenient acquisition of goods and services; and many other applications address simple day-to-day problems like reserving a room or registering for the participation in an event. Because of the web's ubiquity and ease-of-access a web application is often the first choice of technology.

Tim Berners-Lee designed the web as a collaborative tool (Berners-Lee 1996). His early vision was one of document sharing between researchers. The recognition of the web's potential as a platform for interactive applications has been an emergent phenomenon. However, the web's infrastructure has not changed significantly from being document-centered with the result that much of it is ill-suited for application development. Currently, the development of an interactive web application requires knowledge not only of traditional programming languages, but also of technologies and problems specific to the web (e.g., HTML, JavaScript, CSS, HTTP, web-browser-compatibility issues, etc.). As a result, the creation of even a basic web application is difficult.

* All icons used within this dissertation and the prototype tool "Click" have been designed and kindly been made available by David Vignoni (<http://www.icon-king.com>)

Despite the diversity and rapid evolution of web technology, many users carry out some degree of web-based development. For instance, in 1998, a survey found that over 50% of web users have published at least one web page (Pitkow and Kehoe 1998). Some users build web pages because they are fascinated by this ubiquitous delivery medium; others have the more practical goal of efficient information exchange. However, for the most part, the web sites that end users produce today are *static information displays*, with navigation links providing the only interactivity. According to the findings of my work these limitations in users' web development activities are not due to lack of interest but rather due to the difficulties inherent in web application development. Today, an end user with a need for an interactive web application must locate and collaborate with a programmer to pursue his or her goal, and this is not always feasible. The research reported here takes an initial step towards rectifying this situation. I believe that given the right tools and techniques even nonprogrammers can develop simple web applications.

Why would end users want to develop web applications? Why are they unable to do this with today's tools? Who are these end users? What are they like? To gain insight into these questions – and the topic of this dissertation – contrast these two scenarios:

Table 1: Scenario: Anna's Ventures into Web Application Development

Anna uses today's web tools	Anna uses tomorrow's web tools
<p>As webmaster Anna manages a database for registering clients in her company's courses. Recently, she used a survey authoring tool to build a web-based system: clients now submit a registration form, which Anna receives by e-mail. She reads and re-enters the information she receives into a spreadsheet. If a course has seats she registers the person and emails a confirmation; if not, she contacts and coordinates with the client to re-schedule. Often Anna's boss asks for summary reports, which she creates by hand, a tedious process. Anna knows that these repetitive and time-consuming activities should be handled by the computer. But while she knows how to create websites using WYSIWIG editors she has no programming experience. She has heard of Javascript, so she enters "javascript registration database" into a web search engine. She is overwhelmed with results and quickly becomes discouraged because few of the pointers relate to her needs, and the information is highly technical.</p>	<p>A few weeks after her initial effort, Anna learns from a friend about a web development tool that has been targeted at nonprogrammers like her. She decides to give it a try, clicking on the "create new web application" link. The development environment guides her through the process of creating the screens for her registration application as well as the database behind the scenes. Designing the application becomes even enjoyable when Anna notices that the tool asks her the right questions at the right time and uses familiar language instead of the typical "techno-babble." At times it even seems that the tool reads her mind. It allows her to quickly try out different options, entering her own test data and seeing what happens. Anna loses track of time, totally engaged by her design activity. Before the day is over she has fully automated the registration process. Anna has even managed to create a basic web-based report generator for her boss. She feels empowered and is proud of her achievement.</p>

The contrast shown in these two scenarios sketches out the challenges and motivation underlying the work reported here. The following chapters discuss what end-user developers need, how they think, and what can be done, so that a sophisticated user like Anna will not only be able to imagine that she should automate the tedious computing procedures in her life, but also have at her fingertips the support she needs to do it.

1.2 Motivation and Significance

Why is “end-user web application development” desirable and important? I argue that providing end users with tools that increase their involvement in web development will have several important consequences:

- End-users will no longer depend solely on programmers to create custom web applications, enabling the production of a wide range of applications in a shorter amount of time than currently possible;
- The increased number and diversity of people creating web applications will promote innovation, as Deshpande and Hansen suggest: “[releasing] the creative power of people.” (Deshpande and Hansen 2001); and
- Work-processes may become more efficient as individual personnel are enabled to make better use of web infrastructure and connectivity.

Finally, apart from empowering end users to pursue new goals, we must also consider how best to help novice developers create web applications that are more secure, cross-platform-compatible, and universally accessible. User-friendly but “dangerously powerful” web programming languages like PHP (Lerdorf, Gutmans et al. 1995) are becoming popular even among people who do not have the necessary training and experience to develop web applications of high quality. Harrison (2004) calls this the “dangers of end-user programming”. The web engineering community may advocate abstinence from end-user web development (but see it happen nonetheless) or embrace the needs and motivations of nonprofessional developers and support them as much as possible. I advocate the latter.

1.3 Target Audience and Domain

My research mission is making web application development more accessible – particularly to “*casual*” or “*informal*” *webmasters*, people who have created a variety of web content, but who have not been trained in web programming languages or tools. These individuals are likely good candidates for end-user web application development (EUDWeb) – they have already shown themselves to be interested in web development but have not (yet) learned the languages and tools needed to add interactivity to their development efforts.

All of the work reported in this dissertation (with the exception of a broad survey of web developers reported in 3.2) investigates EUDWeb within the context of an academic institution, in particular Virginia Tech. I reasoned that while some webmasters may have been professionally trained in web development, in a university environment many are more casual developers, people who have not been trained as programmers but nonetheless have learned enough about web development to take responsibility for site development and maintenance. Typical examples are the webmasters for academic departments, research labs, or student organizations. Such individuals represent the population I wish to target: end users who are sophisticated enough to know what they might accomplish via web programming but unlikely to attempt it on their own. Although the focus on an academic environment may limit the generalizability of some of my findings (such as the methods used by experienced developers), it is likely that many others (such as general entry barriers to web programming or end-users’ mental models) are also applicable in a different contexts.

A step towards defining a scope for my work in EUDWeb was to investigate the kinds of web applications my target audience would like to build. An initial survey of webmasters at Virginia Tech (see 3.1) indicated that approximately one third of these end users’ needs were *basic data collection, storage and retrieval applications* (such as online registration forms, surveys, or reference databases) – which is particularly interesting as such functionality seems quite reasonable to provide via an EUDWeb tool. Therefore, I have chosen this particular set of applications as the target domain for my investigations.

1.4 Research Questions and Objectives

The work reported in this dissertation focuses on the following three questions:

1. What are the main entry barriers to EUDWeb?
2. How do novice developers naturally think about web programming concepts?
3. What are viable approaches for making web application development more accessible for nonprogrammers?

1.4.1 What are the main entry barriers to EUDWeb?

The first research question explores the status quo in web application development, with the particular focus on uncovering what is needed to enable end-user development. I analyze the factors contributing to the complexity of web application development through surveys and interviews of semi-professional programmers. My rationale for studying experienced web developers is two-fold. First, as opposed to my core target audience, semi-professional developers have already encountered the problems of web application development, yet are not as far removed from an “end-user developer” as a true professional would be. Second, I believe that issues that are troublesome for experienced developers may be insurmountable hurdles for novices and therefore need to be addressed for realizing EUDWeb.

The investigation of the experiences of semi-professional developers is complemented by formative and summative evaluations of Click – a prototype web development tool. For these evaluations I consider only members of my target audience.

As the following chapters (particularly Chapter 3) will discuss in more detail, *web application development is simply too complex*, involving too many concepts, technologies, and relationships which are often out of line with the expectations and “natural mental models” of nonprogrammer developers (see Chapter 4). Current tools that are targeted at end-user developers *lack a holistic approach* towards guiding developers from start (requirements phase) to finish (publishing and maintenance). Perhaps, the main concrete obstacles are the need for integration of numerous diverse technologies, cross-platform compatibility issues, ensuring security, and the process of debugging.

1.4.2 How do novice developers naturally think about web programming concepts?

The second research question explores the “natural” problem-solving approaches and “mental models” of potential end-user web developers (see Chapter 4). In this context, “mental model” is meant to characterize the way that people visualize the inner workings of a web application, the cognitive representations they hold of the entities and workflows comprising a system. The concept of “natural” or “naturalness” (Miller 1974; Pane, Ratanamahatana et al. 2001) refers to the mental model that users hold before they start learning to use a tool or programming language. The underlying rationale for this investigation is that we can build better EUD tools if we know how end-user developers think. If a tool works in the way that a tool user expects and it feels “natural” from the beginning it is likely to be easy to learn and use.

As Chapter 4 discusses in detail, most nonprogrammers *simply do not have deep mental representations* of technical concepts critical to web application development. Although, this may seem like a “non-result” it underlines the level of support and guidance end-user developers require. Although they generally show a good knowledge of the terminology of the web development arena they frequently use technical words (like “database” or “field”) in a nonspecific or imprecise way. They generally use a mix of constraints and rules to describe certain functionality, without paying attention to order or flow of control. They expect many functions (such as search) to be available as basic components. Only few nonprogrammers show any interest or awareness of implementation details for basic services such as session management, database connection, input validation, or security checking.

These findings demonstrate that the current implementation technologies for web application development are at odds with end users’ expectations and thereby create many entry barriers. One approach to overcoming these entry barriers is to shape technology and tools to be more analogous to the natural thought processes and mental representations of novice developers – the focus of the next research question.

1.4.3 What are viable approaches for making web application development more accessible for nonprogrammers?

The third research question explores potential solutions for some of the major problems found in the analysis of the status quo, leveraging the studies of novice web programmers' mental models. I discuss experiences with and particular design solutions for a proof-of-concept end-user web application development tool called Click, which attempts to lower entry barriers to web application development by shaping technology according to the expectations and natural mental models of novice developers. In particular, Click shows that the complexity problem can be overcome by *providing components with high level functionality*, by presenting technical concepts such that they are *close to end users' natural mental models*, and by *integrating all aspects and tools* needed for development such as layout, database design, testing, and production hosting support. Chapter 5 discusses many particular design solutions for lowering entry barriers to EUDWeb.

1.5 Research Overview and Outline

For my research I have adopted an approach that combines analytic investigations of solutions currently in use with detailed empirical studies of end users' needs, preferences, and understanding of web development, and finally a series of prototyping and evaluation efforts. Figure 1 gives an overview of my work, showing – from a bird's eye perspective – the major components that comprise this dissertation and how they are related to each other. The dashed lines show the information flow between the different components. The circle emphasizes the fact that I did not follow a strictly linear process but rather an evolutionary approach, refining my knowledge of end-user web development with consecutive iterations.

In this Chapter I have addressed the motivation, goals and the scope of my research. Chapter 2 discusses related work in the areas of web engineering, psychology of programming, and end-user development as well as web development tools that are currently commercially available. Chapter 3 investigates the status quo and summarizes

the entry barriers – issues that make web application difficult. Furthermore, Chapter 3 reports a study that was aimed at analyzing the components and concepts of typical web applications within my target domain. Chapter 4 describes the expectations of my target audience by exploring their “natural” mental models as they relate to the concepts commonly needed for web application development. Chapter 5 first introduces the concept of “Design-at-Runtime” – a new technique for facilitating and accelerating the development-test cycle, and then reports prototyping efforts focusing on the discussion of the design rationale for our prototype EUDWeb tool “Click”. Chapter 6 reports the results and conclusions of a series of formative evaluations as well as one summative study of the Click prototype. Finally, Chapter 7 summarizes all findings and shows some possible future directions for research in end-user web application development.

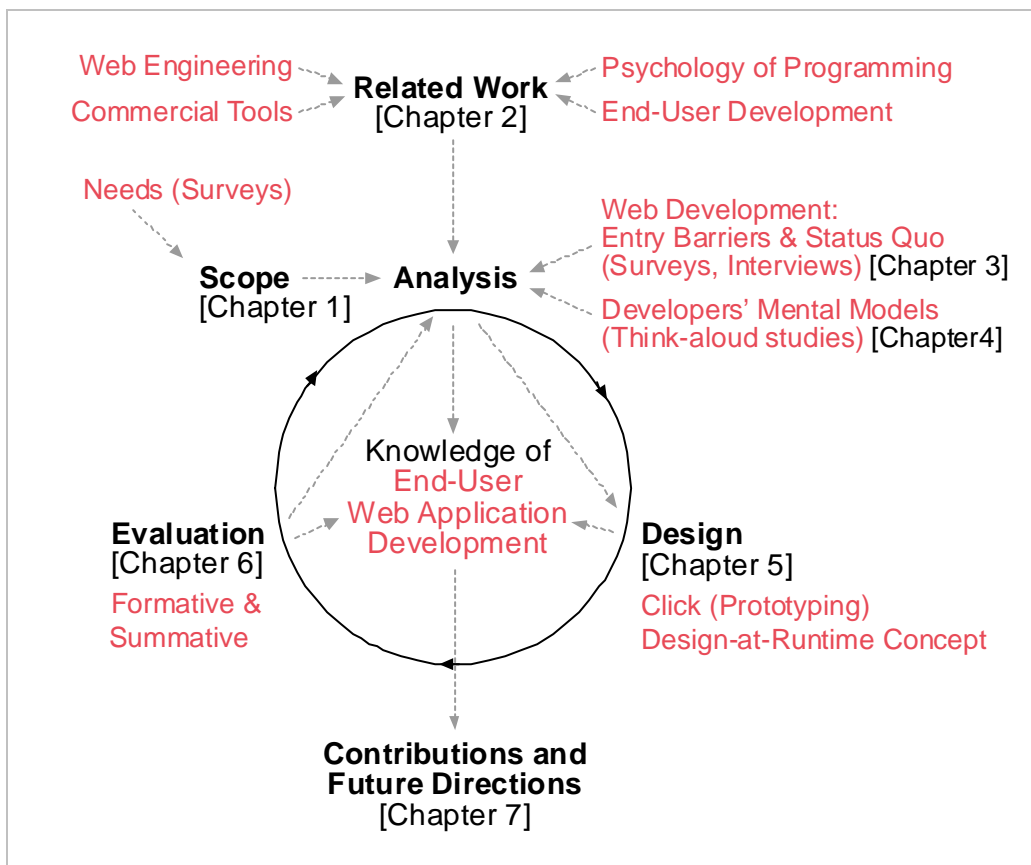


Figure 1: User-centered methods for building web development tools



Chapter 2

Related Work

Two complementary domains of research and practice – *web engineering* and *end-user development* – have focused on methods and tools that could better support the web development needs of both programmers and nonprogrammers. Research in the domain of web engineering concentrates on making web professionals more productive and the websites that they produce more usable, reusable, modular, scalable, and secure. In contrast, web-related research in EUD centers on the idea of empowering non-programmers to autonomously create websites and web applications. Furthermore, my research related to the exploration of end-user developers’ “mental models” also extends work in the domain of *psychology of programming* which studies the behaviors of programmers in general. Figure 2 illustrates the relationship of the different domains.

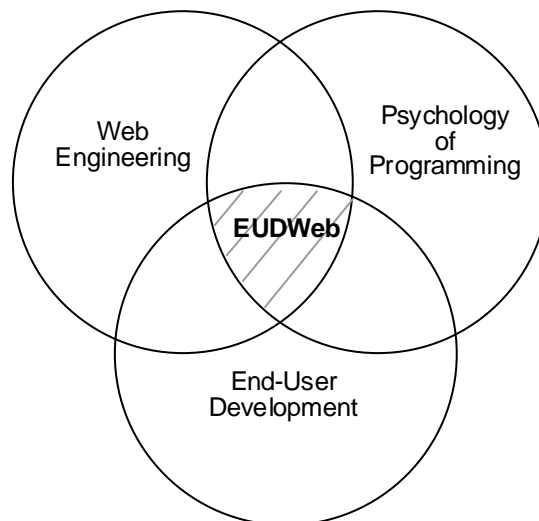


Figure 2: EUDWeb is the cross-section of web engineering, psychology of programming, and EUD

Because the rate of change for web technologies is much greater than that in other areas of engineering, published research often lags behind the solutions produced by the web technology industry. Thus, much of “the present state of knowledge” in web development resides in commercial products which will be discussed here along with research efforts.

2.1 Web Engineering

The establishment of web engineering as its own domain of research is an indication that we are increasingly dependent on web applications (Ginige and Murugesan 2001). In order to ensure reliable and high-performing software it is critical that we attend to the development methodologies, strategies, and tools used for web applications. Web engineering has been established as an area of research in response to what Ginige and Murugesan call the “Web crisis” – currently many web applications are developed in an ad-hoc fashion, relying on programmers to “hack” without sound methodologies, and often resulting in software of low quality. However, this unfavorable view of current practice is not undebated. In a recent survey of industry web development practice Lang and Fitzgerald (2005) found that “the talk of a crisis is largely unfounded” (see 2.1.1).

2.1.1 Studies of Web Development Practice

Researchers concerned about web engineering methods have studied the challenges inherent in web development and the tools in common use. For example, Vora (1998) surveyed web developers about the methods and tools they use, and the problems they typically encounter. In this survey developers reported that ensuring cross-web-browser compatibility, and usability issues associated with WYSIWIG editors were key problems. These findings were confirmed and extended by two recent surveys of web developers that we conducted in 2002 and 2004 respectively (see 3.1 and 3.2).

In a similar vein, Fraternali (1999) proposes a taxonomy for web development tools that suggests some of the major dimensions of web development tasks. For

example, he categorizes available web tools into Visual HTML Editors and Site Managers, HTML-SQL integrators, Web-enabled form editors and database publishing wizards, and finally, Web application generators.

Newman and Landay (2000) investigated the process of website development by interviewing 11 web development professionals. They found that these experts' design activities involve many informal stages and artifacts. Expert designers employ multiple site representations to highlight different aspects of their designs and use many different tools to accomplish their work. They concluded that there is a need for informal tools that help in the early stages of design and integrate well with the tools designers already use.

Lang and Fitzgerald (2005) investigated current practices in Ireland's web development industry through a survey of 167 companies and found that the majority (84%) uses well-documented and carefully designed processes, although overwhelmingly home-grown approaches (95%) rather than specialized nonproprietary methods advocated by the web engineering research community such as the Object-Oriented Hypermedia Design Method (Schwabe, Rossi et al. 1996). The fact that only 2% of respondents had ever used one of the common academic approaches indicates a disconnect between research and industry practice. According to the survey, "the two most troubling aspects [for web developers] were controlling project scope and feature creep and coping with requirements volatility". Although the average delivery time for web projects is rather low (63% of all projects were delivered in 16 weeks or less), there appears to be little indication of ad-hoc and desperate approaches to cope with the requirement for "Web time". The survey does not investigate technological problems in web development. These findings of a well-structured and organized development approach within the professional realm are in stark contrast to the more ad-hoc approaches we found in semi-professional web development (see Chapter 3).

An earlier survey and interview study of 25 United Kingdom-based organizations in diverse sectors of industry and government (Taylor, McWilliam et al. 2002) reflects our findings for semi-professional web development. Only few organizations had formalized procedures and techniques for web development. The majority (68%) used ad-hoc approaches. "IT staff in...20 organisations...indicated that they tended to create

individual web pages directly using the given web development tool rather than plan an outline of the given web page” (Taylor, McWilliam et al. 2002). Only seven of 25 used any formalized website testing methods and only nine companies routinely produced documentation accompanying a web project. The typical ad-hoc approach observed by Taylor et al. (2002) starts with a discussion between IT staff and the “client” department; continues with the construction of a prototype that uses text and pictures provided by the “client”, and completes through successive feedback-refinement loops until the “client” is satisfied and the website is made live. Taylor et al. (2002) also observed a disconnect between research and industry practice stating that “none of the IT practitioners interviewed within 25 organisations...mentioned academic literature or standards bodies as a useful source of website development guidance” (Taylor, McWilliam et al. 2002). This disconnect has been one of the motivating factors for developing the functional prototype tool “Click” and releasing it as open-source software (see Chapter 5).

Rosson et al. (2004) have also studied web developers, but with the particular focus on “informal” developers. In a study of web development in a community computing context, they interviewed 12 informal web developers about how they came to be doing web development, how they acquired their skills, the kinds of projects and programming issues they encountered in their everyday development, and what concerns, if any, they had about the tools they used. Rosson et al. (2004) found that these individuals’ development activities are situated in a collaborative context in which they depend on colleagues for content, expert advice, and testing. Their choice of tools was often based on organizational issues such as cost or who else was using the tool, rather than their own preferences or analysis of tools available. They learned new skills in an informal and as-needed fashion, often by tracking down and adapting or modeling the examples of others.

2.1.2 Model-driven Approaches to Data-intensive Websites

One major focus in the realm of web engineering has been on methods for designing data-intensive web sites such as e-commerce or online-catalog applications. Many of the approaches that are discussed in the following currently address only static data-driven web sites (e.g., brochure sites, product-information sites) as opposed to dynamic and truly interactive web applications (e.g., online membership management application). In most cases, these methods require the site designer to create a high-level model of the web site which then is automatically translated into a working implementation. General advantages of model-driven approaches are:

- The designer does not have to focus on implementation-level details
- Code that has been tested for performance and security is automatically generated
- Multiple views of the data can be derived from only one model (e.g., HTML and WML)

Considering my focus on nonprogrammers, general disadvantages are that these model-driven approaches typically:

- assume expert-knowledge, and sometimes even advocate the collaboration of multiple domain experts (e.g., see WebML in Section 2.1.2.3);
- require considerable planning effort; and
- have a long feedback loop which makes it difficult for end users who may just want to “playfully discover” the requirements

Most importantly, few of the approaches described below (WebML being the exception) currently go beyond data-intensive web sites to address interactive web applications. Research on the model-based paradigm ranges from a few full-scale processes like WebML (Ceri, Fraternali et al. 2000) to many light-weight code generators (e.g., Wolber, Su et al. 2002). Typically, the developer can customize the layout of HTML pages after they have been generated using an external web editor, but these customizations are lost as soon as the code needs to be regenerated because of a needed

change in the data or behavior. The lack of support for evolutionary development from start to finish is an important outstanding research problem.

2.1.2.1 WCML

Gaedke, Schempf, and Gellersen (2000) propose the “WebComposition Markup Language” (WCML) for component-based and object-oriented web development. The XML-based WCML can specify web components at different levels of abstraction and defines how they are implemented using standard web technologies. However, it focuses on the non-redundant definition of *static websites* rather than dynamic and more interactive *web applications*. For example, WCML alone would not be sufficient to specify the business logic for an online registration application. Furthermore, while WCML facilitates reuse, its concepts and syntax make it unsuitable for nonprogrammers.

2.1.2.2 ARANEUS & HOMER

With a focus similar to WCML, the ARANEUS project (Mecca, Merialdo et al. 1999) approaches web site design by supporting development on a high level of abstraction. Site developers first create a conceptual scheme or model of the whole site, then define a hypertext structure and finally the mapping of data to the page layout. HOMER (Merialdo, Atzeni et al. 2000) is offered as a visual tool for the definition of the site’s conceptual scheme. The designers of ARANEUS recognized the advantage of letting the developer specify the layout visually rather than programmatically. Their solution includes the creation of HTML templates which are then translated automatically into a custom format (attribute styles). This approach allows for rapid prototyping and shields the developer from the burden of having to learn another layout language. However, the work reported (Mecca, Merialdo et al. 1999) again only supports the creation of static websites. An extension that covers web applications is planned.

2.1.2.3 WebML

Ceri, Fraternali, and Bongio (2000) recommend WebML as a modeling language for designing data-intensive web sites; a central application domain of WebML is e-commerce.

WebML is based on an XML syntax but also provides a graphical language. The developers create high-level models of the web sites, which are the:

- Data model (expresses data content; uses entity-relationship-like notation);
- Hypertext model (defines set of pages and navigation);
- Presentation model (layout and graphic appearance; XSL is used for transformation into target implementation languages like JSP or ASP.NET); and
- Personalization model (user and group modeling).

WebRatio (WebModels 2005) is offered as an IDE for creating web sites using WebML. For defining the backend business logic WebML offers the concept of “operation units” – and extensible set of operations (e.g., add to database, update, delete), which the developer specifies as part of the hypertext model. WebML is powerful enough for implementing the kind of web applications I have defined as my target domain (see 1.3); however due to its strict, top-down, technical, and layered development approach it appears unsuitable for nonprogrammers.

2.1.3 Languages and Tools for Building Web Applications

A variety of technologies is currently available for implementing the dynamic behavior of web applications. Some of these technologies are purely server-side languages, for example, PHP (Lerdorf, Gutmans et al. 1995), JSP (Sun Microsystems 2002a), or ColdFusion (Macromedia 2002a). Some languages only work on the client-side like Flash (Macromedia 2002b), or Curl (Hostetter, Kranz et al. 1997; curl 2005), and others cover both the client and the server like ASP.NET (Microsoft 2002), OpenLaszlo (Laszlo Systems Inc. 2005), or Flex (Macromedia 2005c).

2.1.3.1 PHP, ASP, Servlets, JSP, ColdFusion

Developing large web applications with a classical programming or scripting language like C (Kernighan and Ritchie 1978) or PERL (Wall 1987) is tedious and error-prone. As a result, new languages have been designed specifically for the web paradigm like PHP (Lerdorf, Gutmans et al. 1995), Microsoft’s Active Server Pages [ASP], Sun’s Servlets and JavaServer Pages [JSP] (Sun Microsystems 2002a), and Macromedia’s

ColdFusion (Macromedia 2002a). These languages offer abstractions for basic services needed by web applications such as session management, or HTTP request handling. Unfortunately, these “web languages” do not solve the web-programming problem of integrating a variety of loosely-related languages and technologies. For example, a web application might use PHP for server-side business logic, HTML for content presentation, Cascading Style Sheets for formatting, and JavaScript for dynamic client-side behavior. If the application includes dynamic client-side behavior, a line of Java code may even look as complicated as shown in Figure 3. Although perhaps an extreme example, a line of code such as this is very difficult to read because it contains four different syntaxes (Java, JavaScript, HTML, CSS) as well as escape characters like the backslash (“\”).

```
out.print("document.write(\"<a href=\\\"#\\\" style=\\\"color:#009900\\\"  
onClick=\\\"new_window('viewmatrix_ieonly.jsp')\"\\\">View  
Matrix</a>\\");");
```

Figure 3: Java code that outputs JavaScript code that outputs HTML code containing CSS

2.1.3.2 AJAX: Asynchronous JavaScript + XML

JavaScript has long been used on the client side to improve the user experience provided by web applications. More recently, web developers have started using JavaScript’s XMLHttpRequest object to request additional data from a web server without having to reload the entire web page, thereby providing smoother state transitions and facilitating interactivity. The web development consultancy firm Adaptive Path has termed this approach Ajax (Garrett 2005). Ajax applications are typically used in conjunction with the aforementioned server-side technologies such as PHP, Servlets, or ASP and can result in highly interactive applications otherwise only possible using non-HTML-based technologies such as Flash (Macromedia 2002b). However, Ajax is an implementation technology and does not at all address one of the main barriers to end-user web application development – complexity (see Chapter 3 for a detailed discussion of barriers to EUDWeb).

2.1.3.3 ASP.NET and JSF

With its .NET framework, Microsoft (2002) introduced ASP.NET and refined the concept of controls (i.e. components). For example, just a single line of code (i.e. `<asp:Calendar runat="server" />`) can now produce a complete web-based calendar, because it can be used to automatically produce the relevant HTML, JavaScript and CSS instructions. Sun has adopted a similar approach for Java-based web development. JavaServer Faces [JSF] (Sun Microsystems 2005) roughly mirror the functionality of ASP.NET.

Nevertheless, even with the help of wizard-laden Integrated Development Environments like Microsoft's Visual Studio .NET (Microsoft 2003a) these tools are not suited for use by nonprogrammers; they expect a thorough understanding of the web development paradigm, knowledge that even relatively sophisticated users are not likely to have.

2.1.3.4 Flash

Macromedia promotes Flash (Macromedia 2002b) as a platform for implementing so-called "Rich Internet Applications" – applications that provide a rich user experience. Flash addresses many of the shortcomings in HTML/CSS/JavaScript-based (a.k.a. DHTML) web application development by offering powerful user interface components and techniques (e.g., drag-and-drop, pixel-level layout control). Flash has been distributed widely as a web-browser plug-in and is now a viable alternative for cross-platform applications (Macromedia 2002c). However, the Flash language itself is not sufficient for building web applications. Typically it is used in conjunction with a server-side programming language such as ColdFusion, or PHP, ASP etc. Furthermore, Flash was originally developed to support graphical animations and because of that builds on a timeline metaphor which is a poor match for typical data-centric applications. Finally, the limited set of predefined user interface (UI) widgets is not always sufficient for building applications which frequently requires the laborious process of manually defining new widgets – a shortcoming that "Flex" (Macromedia 2005c) addresses.

2.1.3.5 *OpenLaszlo and Flex*

In response to the shortcomings of Flash (inappropriate development metaphors, limited predefined UI widgets, and lack of server-communication features), Macromedia has developed Flex (Macromedia 2005c) which conceptually is based on an idea originally developed by Laszlo Systems. Just like the competing OpenLaszlo (Laszlo Systems Inc. 2005), Flex allows the developer to define an entire application in a custom XML notation; in the case of Flex this is called MXML (Macromedia XML). Figure 4 shows an example Flex application that will display the text a user inputs into a text field once the “Show” button is pressed. At runtime, when a user accesses an MXML file via the web browser, the Flex server compiles the MXML on-demand into a Flash file and delivers it (or a cached copy) back to the client. The Flash file then executes on the user’s computer and can communicate with the Flex server to load or save additional data, retrieve data using a web service, or communicate with Java objects via remote procedure calls (RPC). Visual integrated development environments (IDE) are available for OpenLaszlo as well as for Flex. Flex Builder looks like and operates similar to Macromedia’s Dreamweaver. It includes code editing features, WYSIWYG, split code edit/WYSIWYG views, a run mode and debugging facilities. IBM’s Eclipse plug-in “IDE for Laszlo” (IBM 2005b) offers similar functionalities.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.macromedia.com/2003/mxml">
  <mx:TextInput id="source" width="150"/>
  <mx:Button label="Show" click="destination.text=source.text"/>
  <mx:Label id="destination" />
</mx:Application>
```

Figure 4: Example Flex application

OpenLaszlo and Flex address many of the problems that currently complicate web application development, in particular they solve cross-platform compatibility problems, and offer more integrated languages instead of requiring the developer to manually integrate a multitude of different web technologies. Finally, both offer a rich set of

predefined UI widgets and expose an event-based programming model while completely hiding Flash's underlying timeline and movie metaphor. However, both OpenLaszlo and Flex are targeted at professional developers or even development teams rather than at nonprogrammer developers.

2.1.3.6 XUL and XAML

The XML User Interface Language [XUL] (Mozilla 2005) is a client-side, event-based language for the specification of Mozilla applications and extensions but can also be used to create web applications (limited to running within Mozilla Firefox). Similar to OpenLaszlo and Flex it provides a rich set of predefined UI widgets.

Microsoft's eXtensible Application Markup Language [XAML] (Microsoft 2004) is a new XML-based UI description language that allows, similar to XUL, OpenLaszlo, and Flex, the declarative definition of user interfaces. This language is used to define the user interface of Microsoft's next operating system code-named Longhorn. XAML files are compiled into .NET class files (using a .NET compatible language such as C# or Visual Basic.NET) which in turn implement the user interface. Although currently not advertised for web application development, commercial solutions exist that convert XAML/.NET applications into web applications. For example, Xamlon (2005) converts XAML/.NET applications into Flash files which can be deployed as part of a web application.

2.1.3.7 The <bigwig> project

The <bigwig> project (Brabrand, Moeller et al. 2002) approaches the problem of web development by suggesting different sub-languages that are specialized for a particular problem domain. Although this method solves a few typical problems like form-field validation in an efficient manner, it introduces another source of complexity through different syntaxes for the sub-languages. The sub-languages proposed by the <bigwig> project are tailored to the knowledge of professional programmers and are unsuitable for nonprogrammers.

While the main focus of work on web engineering is the support for professional programmers, researchers in this field are starting to consider the web's potential for end-

user computing. For example, Deshpande and Hansen (2001) state explicitly that end-users (i.e. nonprogrammers) should be supported in their efforts to create web applications. However, this discussion has just begun and thus far there has been little in the way of web development languages, tools, or library resources aimed at nonprogrammers – one of the original motivations for my research.

2.2 Psychology of Programming

Behavioral studies of programming were among the earliest examples of research in human-computer interaction (HCI). Well before the appearance of modern interactive computer applications, programmers used text-based command and programming languages to solve complex problems with computers. Cognitive scientists have long been intrigued by the complex and open-ended nature of software design and implementation; indeed many of the models and theories of current HCI had their inception in studies of programmers developing, comprehending, or maintaining code (Shneiderman 1980).

Studies of programming have analyzed several key subtasks—analysis, design, coding, and testing (Pennington and Grabowski 1990; Rosson 1996). Within each of these subtasks, a picture of *active programming* has emerged, similar to the strategies of active use described for end users working with word processors or other desktop applications (Carroll 1990; 2000). That is, programmers are goal-directed and use available resources to produce concrete results as rapidly as possible. This strategy may work against traditional structured and top-down methods of analysis and design (Dijkstra 1968), in that it tends to promote opportunistic and interleaved attention to the different subtasks. For example, an experienced designer may identify a low-level implementation question very early on and proceed in a depth-first fashion to explore the issue before jumping back to a more abstract level of analysis (Carroll, Thomas et al. 1979; Adelson and Soloway 1985; Guindon 1990). Similarly, programmers who are evaluating code resist comprehending it line by line, instead searching for “beacons” that signal key elements for attention (Brooks 1983).

When programming tools are available, programmers recruit them in support of an iterative and heterarchical design and development process. For instance, the Smalltalk environment provides a sophisticated debugger that enables programmers to identify and expand references to objects at runtime. This promotes a strategy of “debugging into existence,” wherein expert Smalltalk programmers do the minimum amount of analysis and programming needed to construct a running application, and then successively refine it by discovering where it “breaks,” correcting the source of the problem, locating the next problem, and so on (Rosson and Carroll 1993; 1996).

These studies of active programming strategies provide a scientific grounding for my work on nonprogrammer tools—I expect end users to be even more active and goal-directed than experienced software developers. Many studies of computer use have demonstrated that for the most part end users do not want to “learn” but rather to “produce”, and will use whatever information or resources available to help them make sense of a task just enough to make progress (Carroll 1990).

The prior work documenting a concrete and incremental style of programming by experts was the inspiration for my concept of design-at-runtime for web application development tools (as discussed in 5.1). This concept is also similar to the automatic recalculation feature found in spreadsheets. Tanimoto (1990) coined the term “liveness” to refer to the degree that a programming language supports testing while development is under way (see also 2.3.2).

2.3 End-User Development

2.3.1 Goals and Trade-off Between Ease-of-Use and Power

In its essence end-user development (EUD) or end-user programming is equivalent to “the psychology of programming for nonprogrammers”. Research in this area studies how programming can be made accessible to nonprogrammers, addressing the needs of both children and adults. Some of the earliest work in EUD had the goal of empowering computer users to pursue personal exploration and learning goals (Papert 1980; Fischer and Lemke 1988; Papert 1993; Reppenning 1994). Other work is more

pragmatic, aiming to provide more accessible support for tasks that could benefit from programming techniques, e.g., spreadsheet manipulations (Burnett, Atwood et al. 2001) or text formatting (Lieberman 2001).

The competing goals of ease-of-use and power lead to inevitable tradeoffs in the design of end-user programming languages and environments (Repenning 1994; Eisenberg 1995; Gilmore, Pheasey et al. 1995). On the one hand, designers want to build EUD systems that are as self-evident and easy to use as possible, so that users with little or no programming experience will be able to use them. But on the other hand, they want the systems to be powerful, supporting traditional constructs of programming languages such as abstraction, modularity, and reuse. This tension has been a pervasive influence on my work, in that I have relied extensively on empirical studies to determine how best to make my tools accessible to nonprogrammers, while at the same time giving them access to a broad range of useful and powerful functions.

2.3.2 The Spreadsheet Paradigm and the Concept of Liveness

Perhaps the most successful EUD system is the spreadsheet. Nardi and her colleagues (Nardi and Miller 1991; Nardi 1993) have documented considerable end-user expertise in spreadsheet development though perhaps not as much expertise in testing and debugging (Brown and Gould 1987; Burnett, Ren et al. 2001). According to Nardi, one reason for the success of the spreadsheet paradigm is that it builds from a specific and familiar visual formalism (i.e. ledger pages). Of course at the same time it also provides a very visible and valuable service, namely the calculation and automatic updating of mathematical formulas. I believe that a web page may act as a similar visual formalism, and that nonprogrammers might be able to reference, query, and manipulate elements of an interactive web page much like they now work with spreadsheet cells.

Tanimoto (1990) proposes the concept of “*Liveness*” to classify visual programming languages according to the degree they give “live” feedback to the programmer. He defines four different levels of this concept ranging from purely “informative” feedback on level 1 (e.g., a flowchart visualization of the program only used for comprehension), the “informative and significant” level 2 (e.g., a visual program

specification that can be executed), the “informative, significant, and responsive” level 3 (e.g., every mouse and button action triggers an update of the visual representation), to the “informative, significant, responsive, and live” level 4 (e.g., the system is continuously active and gives feedback to the programmer). With its automatic recalculation feature, the spreadsheet paradigm is a familiar example of level 3 liveness. The *Design-at-Runtime* paradigm discussed in Section 5.1 is an application of the concept of liveness at level 3 (or potentially 4) to the development of form-based and database-driven web applications.

2.3.3 The Concept of Naturalness in End-User Programming

A user-centered approach to EUD should take into account users’ abilities to produce and comprehend natural language. For example, Miller (1974; 1981) asked nonprogrammers to specify procedural tasks, and used their responses to recommend features of end-user programming languages. This approach has considerable face-validity with respect to minimizing the “cognitive distance” between a user’s intentions and a language specification (Green 1989; 1990).

Pane and Myers (2000; 2001; Pane, Ratanamahatana et al. 2001; 2002) followed a naturalness-oriented approach in their design of HANDS (Human-centered Advances for Novice Development of Software): they first performed behavioral studies to identify the control logic and data specifications that children and adults use in natural language, then designed a new language and environment based on these abstractions. Many of Pane et al.’s findings are applicable to the research on “mental models” reported in Chapter 4. In particular, based on the results from two studies (study 1: children program a video-game; study 2: children and adults program database scenarios), Pane et al. (2001) propose that:

- Rule- or event-based programming may be more natural than the imperative paradigm;
- A mix of different programming styles may improve usability;
- Operations on multiple objects are more often expressed in terms of sets than loops;

- Negation is rarely used; expressing negative concepts is harder than affirmative ones;
- The Boolean operator *AND* is frequently used where *OR* would be correct;
- Application state is expected to be maintained implicitly; state variables are rarely used;
- Sort is expected as a basic operator through the use of expressions like “alphabetical” or “from A to Z”;
- Complex conditionals are often expressed via a set of mutually exclusive rules or by stating a general condition, subsequently modified with exceptions.

2.3.4 Visual Languages and Direct Manipulation

Most languages designed for end users rely on visual interaction techniques to some extent, for example programming with graphical rewrite rules and agents (e.g., Repenning 1994). The emphasis on visual techniques stems from a variety of beliefs such as the relative naturalness of pictures as a representation medium, greater expressivity of pictures, and the rapid processing of image or spatial information (although note that empirical justification of such beliefs is rare: Blackwell 1996; Whitley and Blackwell 1997). Visual techniques are also an important component of direct manipulation systems (Shneiderman 1983), in which users point, grab, and drag visual components to interact with a system. Because of its ubiquity in operating systems and desktop applications, direct manipulation is a familiar interaction technique that nonprogrammers are likely to expect in EUD systems.

2.3.5 Cognitive Dimensions Framework

Green and Petre (1996) developed the “*Cognitive dimensions framework of notations*” – in its essence a set of dimensions that can be used to analyze the usability of visual programming languages and environments. Implicitly these dimensions define usability requirements for virtually any programming language. Table 2 lists these dimensions along with explanations similar to Green and Petre’s original descriptions.

Table 2: Green and Petre's cognitive dimensions framework of notations

Abstraction Gradient: What levels of abstraction can the user work on?
Closeness of mapping: How close are the language's concepts to the expectations of the users?
Consistency: How consistent is the language internally?
Diffuseness: How many different symbols does the language employ to express a meaning?
Error-proneness: To what degree does language's notation induce mistakes?
Hard mental operations: Do any tasks require substantial memorization or calculation efforts?
Hidden dependencies: Can dependencies exist that are not explicitly shown by the language?
Premature commitment: Do users need to make decisions before they have the necessary information?
Progressive evaluation: To what degree can not-yet-completed programs be executed?
Role-expressiveness: Do the language's components clearly show what they stand for?
Secondary notation: Can user annotate the language using notes, colors, comments...?
Viscosity: How difficult is it to make changes? Do small changes have global effects?
Visibility: Can all the code be viewed simultaneously or, at least, can different views be combined?

These dimensions have provided a general set of guidelines in my development of EUD web tools. For instance, during the prototyping efforts reported in Chapter 5 I focused on addressing particular issues such as ensuring the *Closeness of mapping* by employing concepts that are close to nonprogrammers' natural mental models (see Chapter 4), reducing *Premature commitment* by supporting opportunistic behavior, and facilitating *Progressive evaluation* through the *Design-at-Runtime* concept (see 5.1).

2.3.6 End-User Development for the Web

Well before the development of the World Wide Web, end-user development of basic data management applications was a topic for academia and industry. HyperCard (Apple 1987) is an early example of a successful EUD tool. More recently, web development research projects such as WebFormulate (Ambler and Leopold 1998), FAR (Burnett, Chekka et al. 2001), DENIM (Newman, Lin et al. 2003), BioPro (Shimomura 2004) and WebSheets (Wolber, Su et al. 2002) have explored specific approaches to end-user programming of web applications.

WebFormulate (Ambler and Leopold 1998) is an early tool for building web applications that is itself web-based and partly platform independent (the page layout must be defined with a desktop application). It includes a form-based visual language, which allows developers to construct new computations by referencing other objects via point-and-click. WebFormulate uses a message passing paradigm that reports any changes to an object immediately to all interested objects. The development environment running within the web browser communicates with the web server through a hidden HTML frame, an approach we have adopted for our prototype EUDWeb tool “Click” (see 5.5). It is not clear, however, how WebFormulate abstracts the process of defining the business logic without requiring the end user developer to write actual code.

FAR (Burnett, Chekka et al. 2001) is an online business development tool that combines ideas from spreadsheets and rule-based programming with drag-and-drop web page layout functionality. FAR’s direct manipulation programming paradigm seems suited for end users, but usability studies involving human subjects have not been reported. FAR may be ideal for calculation-intensive web applications (e.g., extending the spreadsheet paradigm). But how its expressive power will scale to the design of general web applications is still unclear.

DENIM (Newman, Lin et al. 2003) is a tool that can assist professional and nonprofessional web developers in the early stages of design with digital sketching of informal interactive prototypes. However, while professional web developers are trained to transform an informal prototype into a final application it is not clear how end-user developers may create a production web application.

BioPro (Shimomura 2004) is a visual tool that supports the construction of web applications by choosing components (such as hyperlinks, tables, text fields) from menus. It tightly integrates a basic database management system and allows the testing of partially developed applications by substituting example data for unknown inputs. The tool is extensible by allowing the developer to add custom Java code snippets to an application which are stored compartmentalized to facilitate code readability. Although the tool claims to support application development according to the developer’s “brain-

image” (mental model) of the application, it exposes many non-intuitive technical concepts (such as hidden fields to forward information between pages) and seems therefore unsuitable for end-user developers.

The WebSheets tool (Wolber, Su et al. 2002), although currently limited in power, is close to our holistic vision of end-user web development. It uses a mix of programming-by-example, query-by-example, and spreadsheet concepts to help nonprogrammers develop fully functional web applications. The disadvantage of programming-by-example techniques (also known as programming-by-demonstration) is the error-prone computer-controlled process of induction (a.k.a. generalization) which can quickly become a source of frustration for developers if it is invisible, uncontrollable, or based on inappropriate heuristics (McDaniel 2001; Myers and McDaniel 2001). Although innovative and promising in its idea, it is unclear if the WebSheets approach will scale to even slightly more complex applications (e.g., currently WebSheets only supports one-to-one mappings between database tables and HTML data tables).

2.4 Commercial Web Development Tools

2.4.1 Professional Productivity Tools

Some of the most active work on web development is occurring in the marketplace. A major focus of research and practice is *tools* that assist web developers in becoming more productive. Many powerful computer aided software engineering (CASE) or rapid application development (RAD) tools have been developed for experienced developers like WebRatio (WebModels 2005), Rational Web Developer for WebSphere Software (IBM 2005a), or Visual Web Developer 2005 (Microsoft 2005b).

Helman and Fertalj (2003) briefly reviewed a number of professional code generating tools – CodeCharge Studio (YesSoftware 2003), CodeJay (2003), Visual Studio (Microsoft 2003a), and Web Matrix (Microsoft 2003b) – from the perspective of productivity tools for programmers. Apart from mentioning many convenient features

(such as automatic generation of code for forms and reports), they discuss typical shortcomings of current code generators including the following:

- “...there is almost no support for more complex reports that include user interaction and data input” (one of the main targets I identified for EUDWeb),
- Code generators that are designed to produce output for different languages (such as PHP, ASP.NET, JSP) often only adjust the syntax for the automatically generated code but do not produce code that takes advantage of the programming features specific to a particular language thereby creating sub-standard code,
- Code generators often have limited capabilities for producing object-oriented code,
- Generated code is often not well documented and lacks comments to indicate which portions of the code are meant to be customized by the developer,
- Lack of automatic generation of external code documentation (such as JavaDoc),
- Lack of code generation for web site navigation features (such as menus etc.),
- Tools often do not integrate well with other tools.

Even though these tools may simplify professionals' web development process by providing wizards and visual tools, none of them have been targeted at nonprogrammer developers, so in general they assume the knowledge, working culture, and expectations of an experienced programmer.

2.4.2 Database-centric Tools

Vendors of traditional databases have extended their products to include interfaces to the web. For example, using FileMaker Pro (FileMaker 2005) and its “Instant Web Publishing” functionality, database users can create a web user interface to control their databases. However, these approaches are typically very database-centric and often not ideal for the design of custom web applications. On the World Wide Web, a variety of Application Service Providers (ASPs) offer web-based solutions based on the database paradigm. A representative of this ASP model is FormSite (Vroman Systems Inc. 2005) – a tool targeted at users who want to create form-based applications.

Although FormSite is easy to use, it is also very domain-specific, limiting the developer to simple form-based data input applications. For example, an end user may use FormSite to create a survey or a multi-page registration form, but would be unable to create an online membership management application or an online reference database.

2.4.3 Online Site Builders and eCommerce Tools

A number of commercial services, like Homestead (2005), or ZyWeb (2005), offer web-based WYSIWYG editors that are frequently referred to as “site builders.” Some of these site builders like Trellix’ Web Express (2005) allow users to add dynamic elements like guest books, credit card processing modules, or shopping carts to their web site. Balthaser:Fx (2005) offers an Internet-based service targeted at professionals who use Macromedia’s Flash. The service allows these developers to create Flash web sites completely online. Balthaser:Fx has extensive libraries containing predefined elements that aid less experienced Flash designers.

Domain-specific web development tools already enable nonprogrammers to offer interactive services on their web site. For example, the commercial service YAHOO! Store (YAHOO! 2005) allows merchants to create, customize, and maintain a complete e-commerce web site. The limitation of these tools is that they have been created to support a very limited number of classes of interactive applications and thus cannot satisfy more situation-specific needs.

2.4.4 End-User WYSIWYG Editors and Web Application Builders

Closest to the focus of my research are tools that require little or no programming knowledge. Currently these tools exist in two flavors: desktop-based what-you-see-is-what-you-get (WYSIWYG) application builders and web-based application builders. Some of the widely-used desktop-based WYSIWYG web editors include Dreamweaver (Macromedia 2005b) and its “end-user friendly sibling” Contribute (Macromedia 2005a), FrontPage (Microsoft 2005a), and Adobe GoLive (Adobe 2003). Dreamweaver, for example, extends the standard WYSIWYG authoring environment to include database connections and “server behaviors” that implement common server functionality such as

user authentication or recordset paging. Similarly, Microsoft's FrontPage offers "web components," modules that authors can embed to create counters, advertisement servers, forum pages, and spreadsheets with no programming (Microsoft 2005a). Some of FrontPage's shortcomings (lack of integration, workflow) are discussed in 2.4.5. In contrast, Instantis SiteWand (Instantis 2003) takes an approach where users design and upload HTML page templates and then specify online how these pages should interact. This paradigm leverages existing knowledge well because users can design a site's "look & feel" using their favorite WYSIWIG editor. SiteWand abstracts many of the details of web development which makes it suitable for non-programmers. Compared to other tools, SiteWand is very close to my vision of end-user web application development, but it still places many challenges in the way of the nonprogrammer (e.g. a complex text-based templating language, and a non-intuitive programming framework based on an "engines" concept).

It is difficult to classify tools such as Dreamweaver as programmer or nonprogrammer tools—they do not require programming skills per-se, but still have a considerable learning curve. In its current state Dreamweaver is likely used more as a tool to enhance programmers' productivity than to extend nonprogrammers' capabilities. However, a trend towards improved ease of use and extended power is apparent. Another concern is that such tools have limited power for nonprogrammers. They do allow the creation of basic database applications but do not support ad hoc extensions of the basic application with custom features unless the developer is willing and able to write low-level code.

2.4.5 A Review of State-of-the-Art Web Development Tools

A number of commercial web development tools like FrontPage (Microsoft 2005a) have begun to directly support nonprogrammers in the creation of basic web applications. However, so far, the research community has devoted little effort to studying approaches and features found in those commercially available tools.

In order to better ground my research in related work and as yet another source of requirements for EUDWeb, we reviewed nine commercial web development tools (this

work was done in close collaboration with Jonathan Howarth). We analyzed each tool from the perspective of suitability for end-user development; looking across the nine tools we were able to compare and contrast alternative and best-of-breed approaches for many aspects of web application development. The full report is available as a tech report (Rode, Howarth et al. 2004). The following summarizes the study and key findings.

2.4.5.1 Overview of the Review Process

For our review we selected tools based on both their apparent market dominance and their potential sophistication. Although most web development tools have a particular focus regarding target development project and user group, we found that the majority of tools can be grouped into one of three categories: database-centric tools (we reviewed: FileMaker Pro 7), form-centric tools (we reviewed: Quask Form Artist), and website-centric tools (we reviewed: Microsoft Visual Web Developer 2005 Beta, YesSoftware CodeCharge Studio, H.E.I. Informations-systeme RADpage, Instantis SiteWand, Macromedia Dreamweaver 2004 MX, Macromedia Drumbeat 2000, Microsoft FrontPage 2003). To structure and constrain our review, we analyzed the commercial tools with a focus on how they approach the implementation of particular features that are common in web application development. To make these features more concrete and to convey our assumptions about a likely end users' goals and activities, we constructed a reference scenario and persona. In the scenario, a nonprogrammer was attempting to build what we feel is a typical example of a data-driven website – an online employee database. We reviewed each tool for the approach and features needed to implement this scenario.

2.4.5.2 Usability Findings and Recommendations

What does the ideal web application development tool look like? I believe that there cannot be only one such tool. Because developers have different needs and different skill sets, different developers will be best served by different tools. In general, our review suggests that while productivity tools for programmers like Microsoft Visual Web Developer have matured to provide significant support for web development, tools for nonprogrammer developers are still in their infancy.

Most of the end-user tools that we reviewed do not lack functionality but rather ease of use. For instance, even apparently simple problems such as implementing the intended look and feel become difficult when a novice has to use HTML-table-and-flow-based positioning instead of the more intuitive pixel-based positioning.

Although most tools offer wizards and other features to simplify particular aspects of development, none of the tools that we reviewed addresses the process of development as a whole, supporting end-user developers at the same level of complexity from start to finish. Indeed, Fraternali's and Paolini's comment about web tools of five years ago seems to be still true today: "...a careful review of their features reveals that most solutions concentrate on implementation, paying little attention to the overall process of designing a Web application" (Fraternali and Paolini 2000).

The otherwise comparatively novice-friendly FrontPage, for example, begins the creation of a new application by asking the developer to make a premature commitment to one of the following technologies: ASP, ASP.NET, FrontPage Server Extensions, or SharePoint Server. An excerpt from an online tutorial for FrontPage illustrates the problem: "...You can also use the Form page Wizard and Database Interface Wizard with ASP or ASP.NET to edit, view, or search records from a Web page. The Form page Wizard works on a Web site running Windows SharePoint Services 2.0, yet the Database Interface Wizard does not." Such a selection is likely to confuse anyone but a seasoned web developer.

Currently, all the tools that we reviewed would cause major problems for the informal web developer who wants to create more than a basic website. The tool that a user like Anna (from our introduction scenario, see 1.1) is looking for has to have multiple reference examples, well-guided but short wizards, an integrated zero-configuration web server for testing purposes, and good support during the deployment phase of the application. Also, as Anna becomes more familiar with the capabilities of the tool and her applications become more ambitious, the tool should help her learn by gradually exposing the inner workings of the wizards and forms. Ideally, by placing learners on a "gentle slope" (MacLean, Carter et al. 1990), the skills required to implement advanced features should only grow in proportion to the complexity of the

desired functionality – “Make simple things easy, and hard things possible.” The ideal tool for nonprogrammer web developers would provide ease of use with the appropriate abstractions but also offer power and flexibility by allowing integration of user-defined and automatically-created code. Until such a tool exists, we think that there may be a market for less flexible but easier to use special-purpose tools similar to Macromedia Drumbeat (which simplifies layout definition by abstracting the HTML-flow-based layout, and tightly integrates database management tools). Table 3 summarizes our findings in the form of guidelines and recommendations for future tools targeted at end-user developers.

Table 3: Guidelines for EUDWeb tools derived from our review

Recommended Solutions for Tools Targeted at End Users
<p>Getting Started</p> <ul style="list-style-type: none"> • Avoid technical jargon for startup options (e.g. non-technical descriptions of underlying required technologies) • Provide wizards (with minimal premature commitment) • Provide example solutions • Provide templates
<p>Workflow</p> <ul style="list-style-type: none"> • Take a holistic approach to web application development • Allow for gradual construction of the database
<p>Level of Abstraction</p> <ul style="list-style-type: none"> • Provide high-level components such as data tables but also lower-level components for flexibility • Make components customizable, skinable
<p>Layout</p> <ul style="list-style-type: none"> • Include the layout editor in the tool • Pixel-based editors are simpler than HTML-flow based editors • Provide templates and themes that can be applied site-wide

Database
<ul style="list-style-type: none"> • Allow for the creation of a new database including schema from within the tool or through a connection to an existing external database • Facilitate late changes to database schemas • Support populating and editing the database from within the tool • Provide a visual or form-based query builder
Application Logic
<ul style="list-style-type: none"> • Make session management transparent • Provide predefined high-level actions such as add, update, delete record, go to page, and send email • Offer wizards to create commonly used design patterns such as overview-detail or repeating regions
Testing and Debugging
<ul style="list-style-type: none"> • Facilitate fast iteration between building and testing, e.g. by using design-at-runtime (see 5.1) • Avoid syntax errors by constraining the development UI • Provide context-sensitive error messages
Learning and Scaling
<ul style="list-style-type: none"> • Allow for viewing and editing code parallel to design (e.g. Dreamweaver's split view) • Allow for viewing and editing code by component • Allow developer to edit automatically generated code or provide hooks or placeholders for custom code • Reintegrate custom modifications made by the end user into the automatically generated code (challenging research issue) • Document automatically generated code
Security
<ul style="list-style-type: none"> • Provide predefined user/permissions management and high-level security components (e.g. Visual Web Developer's Login control) • Provide high-level validation features for input components • Generate secure code (e.g. check inputs, SQL commands)
Collaboration
<ul style="list-style-type: none"> • Facilitate collaborative development by offering a file check-out or versioning system • Implement levels of access (e.g. develop, modify data, etc.)
Deployment
<ul style="list-style-type: none"> • Provide a built-in zero configuration test server, whether as a local server (e.g. Visual Web Developer) or a remote server (e.g. Form Artist) • Provide a built-in production server (e.g. FileMaker Pro) or easy to use deployment wizard

2.5 Summary and Conclusions

Table 4 shows a summary of the findings and trends of the related work for the three domains of *web engineering*, *psychology of programming*, and *end-user development* as well as a selected “lessons learned” from a review of commercial web development tools.

Table 4: Summary of the findings and trends of the related work

Web Engineering
<ul style="list-style-type: none"> • Key problems for web developers: cross-platform compatibility, and usability issues of web editors • Experts employ multiple (often informal) representations to highlight different design aspects • Declarative XML-based, event-based, component-based, object-oriented UI definition languages • Higher abstraction levels: widget sets, features to facilitate common tasks such as input validation • Better integration: less need for manually combining technologies such as HTML, JavaScript, CSS • Better interaction between client-side and server-side: simplified use of web services, or RPC • Better cross-platform compatibility through use of technologies like Flash • Visual tools integrate code-editing, WYSIWYG, and graphical notations • Model-driven approaches separate the data model, application logic, and presentation • Design patterns like MVC are increasingly becoming the modus operandi • Use of UML or UML-related design representations
Psychology of Programming
<ul style="list-style-type: none"> • Programmers are goal-directed and focus on producing concrete results fast (active programming) • Programmers evaluating code resist comprehending it line by line; instead search for “beacons” • Programmers work opportunistically, jumping often between high-level modeling, implementation • Programmers work iteratively using a “debugging-into-existence” approach

End-User Development

- Central problem is finding good tradeoff between ease-of-use and power
- Use of metaphors (e.g. ledger pages for spreadsheet)
- Concept of “Liveness”
- Findings related to concept of “Naturalness” in end-user programming:
 - Rule-or event-based programming may be more natural than the imperative paradigm
 - A mix of different programming styles may improve usability
 - Operations on multiple objects are more often expressed in terms of sets than loops
 - Negation is rarely used; expressing negative concepts is harder than affirmative ones
 - The Boolean operator AND is frequently used where OR would be correct
 - Application state is expected to be maintained implicitly; state variables are rarely used
 - Sort is expected as a basic operator by using expressions like “alphabetical”
 - Complex conditionals are often expressed via a set of mutually exclusive rules or by stating a general condition, subsequently modified with exceptions
- Visual languages can be beneficial but their general superiority has not been shown
- Cognitive dimensions framework of notations implicitly establishes key usability requirements
- Programming-by-example paradigm is powerful but induction process is difficult and error-prone

State-of-the-Art in Commercial Web Development Tools

- Many Web IDEs include layout tools, DB tools, code generation, and debugging features
- Database-centric tools allow web publication of typical databases but are not very customizable
- “Site builders” offer predefined modules such as guest books, shopping carts etc.
- eCommerce tools allow nonprogrammers to setup online stores, e.g. Yahoo Stores
- WYSIWYG web editors include basic components but often do not sufficiently abstract
- General lack of attention towards start-to-finish/holistic guidance (including the publishing step)
- Current tools for the most part do not expose a gentle slope of complexity

In summary, much of the prior research within the web engineering discipline has contributed to solving the complexity problem and the reuse problem by proposing higher levels of abstraction (through model-based approaches or component-based approaches). However, very little work has targeted nonprogrammers as their audience, which is

indicated by the use of abstract concepts such as the object-oriented paradigm (inheritance etc.) or data modeling using versions of the entity relationship model.

The psychology of programming domain has uncovered behavioral patterns of expert programmers (e.g., active programming, debugging-into-existence), many of which extend to novice programmers as well (see 6.2.9) and should be facilitated for EUDWeb.

The two contributions within the end-user development discipline that are most relevant to my work are the concept of naturalness and the cognitive dimensions framework. The work reported in Chapter 4 examines naturalness within the context of EUDWeb. The cognitive dimensions framework has served as a set of high-level guidelines throughout the development of our prototype EUDWeb tool Click (Chapter 5). Although research in end-user development has investigated many different application domains, the area of web application development is still largely unexplored – a major motivation for the work reported here.

Despite the considerable progress in the power and ease of use of web development tools, none of the tools we have reviewed is sufficiently powerful while *also* being appropriate for end users. Typically, tools that target a narrow domain (such as survey creation tools) are easy to use but not very expressive. The more scalable tools frequently do not take a holistic approach and fail to guide developers from start to finish, and expose a steep learning curve as soon as the developer goes beyond the basics. Our prototype tool Click, as discussed in Chapter 5, attempts to attain the delicate balance between power and ease of use while exposing a “gentle slope” learning curve.

The particular problems that developers face when creating web applications are discussed in the following Chapter 3. Many of these issues create entry barriers which have to be overcome in order to make web application development accessible to nonprogrammers.



Chapter 3

Entry Barriers and Status-Quo in End-User Web Application Development

The research mission of my work is to lower the entry barriers to web application development thereby making it more accessible (see 1.4). This chapter identifies and discusses the particular entry barriers to web application development. I report the findings of one survey and one interview study of semi-professional web developers at Virginia Tech (3.1), and furthermore of one comprehensive survey of a more diverse audience which extends beyond the academic environment (3.2).

My goals are two-fold: I expect that the findings can contribute to the ongoing development of web technologies and tools for professionals and semi-professionals, but more relevant to *end-user* development, I want to anticipate and “hide” these problems as much as possible in the development of tools for nonprogrammers. My rationale was simple: issues that are troublesome for experienced developers may be insurmountable hurdles for novices.

3.1 Survey and Interviews of Experienced Web Developers

As one of the first sources for requirements development, I surveyed sophisticated developers at Virginia Tech regarding the challenges, tools, and processes within the domain of web application development. Note, however, that these participants should be considered semi-professionals rather than expert developers because for most of them

web application development was just a part of their work rather than a full-time occupation.

Findings from two distinct studies are reported – a survey and an interview study. The survey and interview study jointly highlight key challenges such as: implementing security, cross-platform compatibility, debugging, and technology integration. First, the findings from the survey and the interviews are reported separately and then summarized to paint a coherent picture of the status-quo of web development.

3.1.1 Methods and Results

The research presented here was initiated with a survey of web developers that asked for ratings and examples of various web development activities. In order to enrich and explain the findings and to increase the total number of reported experiences, I later conducted in-person interviews (see 3.1.1.2) with developers who (with the exception of one) had not participated in the survey.

3.1.1.1 The Survey

The survey data analyzed here is a subset of the data collected in a survey titled “Interactive Websites” conducted in May 2002. See Appendix A.2 for the questionnaire form and a summary of results. The individual response data along with a general summary can be browsed online (Rode 2002b). Survey participants were recruited from different sources: an invitation email was sent to all webmasters who maintained an organizational website on the universities’ web hosting system, as well as to all subscribers of the university’s web developers mailing list and of the computer science graduate students mailing list. The email invitation stated the purpose of the investigation and contained a link to the web-based survey. In order to encourage participation I advertised a raffle of lunch coupons ranging from \$5 to \$15. The survey was open for participant input for approximately three weeks at the end of the spring semester in 2002.

The survey had two distinct purposes. One purpose was to determine webmasters’ needs for web applications, the other to learn about the challenges inherent in web application development. For the latter purpose, the survey contained a section targeted at

experienced developers only. Participants were asked to respond to this section only if they had previously developed a web application.

On average, the 31 respondents who answered the questions about web development rated themselves just above the mid-point on a scale from 1 (no knowledge in web application development) to 5 (expert knowledge); the mean self-rating was 3.2 (SD=0.9). Their self-reported years of experience in web application development were approximately equally distributed between “less than a year” and “more than 5 years.” 8 respondents identified themselves as undergraduate students, 6 as graduate students, 7 as faculty, 6 as staff, and 4 as alumni.

In order to gauge the needs for web applications, one of the first questions in the survey asked the respondents to point out opportunities for “interactive websites” (the survey had previously defined this term) in their environment (Figure 5).

Where do you see opportunities for interactive websites in your environment (related *and* unrelated to Virginia Tech)?

For example, think about what is currently done on paper but may be done more efficiently or conveniently via the web. (examples: A website for ... may help our bowling club to...; A website for ... would help the people in our department to...)

Figure 5: Survey question targeted at exploring end users' needs for “interactive websites”

The analysis of the 67 responses to this question (Table 5) indicates that approximately one third of the respondents' needs could be addressed by a high-level development tool that offered basic data collection, storage and retrieval functionality. Another 40% of the requests could be satisfied through customization of five generic web applications (resource reservation, shopping cart and payment, message board, content management, calendar).

Research on tailorability (e.g., MacLean, Carter et al. 1990) has shown that software can be designed for easy customization by end users. Diverse requests for more advanced applications comprised the remaining 25%.

Table 5: Virginia Tech webmasters reporting their needs for “interactive websites” a.k.a. web applications (number in brackets indicates the frequency of requests; N=67, with some respondents reporting needs in multiple categories)

Generic web applications [30]

Resource reservation systems of some sort [9], Shopping cart & online payment systems [8], Message board systems [7], Content management systems [3], Calendar systems [3]

Basic custom web applications (data collection, storage & retrieval) [25]

Intra/Interdepartmental forms [8] (i.e. service requests [2], generic forms, forms for graduation, purchase requests, reimbursement for travel expenditures, Domain Name Service entries), Teacher/Course evaluations [2], Track meeting minutes [2], Taking job applications [2], Member database, Track technical info about faculty & staff, Computer repair database, Knowledge base, Updating info for student organizations, Phone book, Music database, Guest book, Register for undergraduate research, Event registration, Volunteer registration

Advanced custom web applications [18]

Wage employee time tracking [3], Web storage and sharing of files and pictures [2], Portal [2], Interactive tutorial [2], Collaboration tool, Research tool, Project management, Paper peer review, Nutritional guide, Tax forms, Purchase advisory tool, Online Auction, Chat

These results appear encouraging, in that about 75% of the requested applications seem to be good target tasks for end-user development tools (40% generic applications + 35% basic data collection, storage, and retrieval). In response to the survey findings I chose to focus on the subset of requests involving basic data collection, storage, and retrieval as the target domain for my research, because such functionality seems quite reasonable to provide via an EUDWeb tool. While these web applications may be quite diverse in their purpose or domain (compare a plant-pathology database with a conference paper review system), they are rather homogeneous and basic on a conceptual level, having only a limited number of well-defined features such as save data record, edit data record, delete data record, or display list of data records.

Of the 67 individuals who responded to the survey, 40 indicated that they had never developed an interactive web site. I asked these individuals to tell me why; Figure 6 summarizes responses to this question. Notice that half of the respondents replied that either “I would want to, but I expect it will be too difficult” or “I would want to, but I don’t have the time”. Another 6 people indicated similar knowledge-related and resource-related reasons. I concluded that 26 out of these 40 individuals (i.e., 65%) are potential

candidates for an easy-to-use web development tool – that is that they might use such a tool if we can assume that it would require minimal programming skills and little time investment.

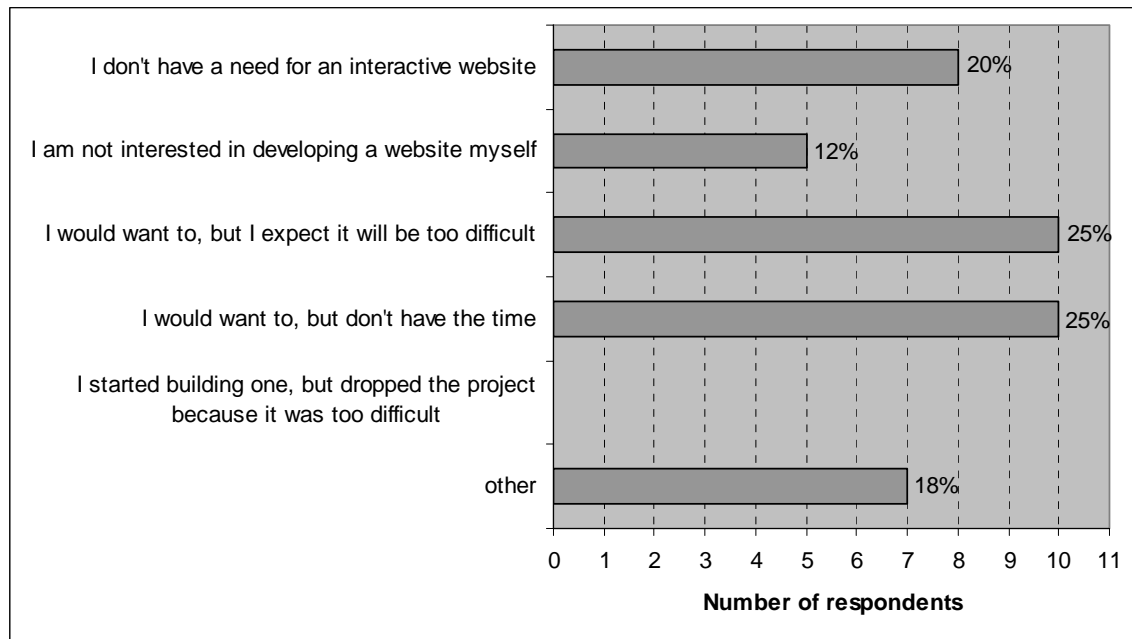


Figure 6: Virginia Tech webmasters reporting their reasons for not developing web applications themselves (N=40)

Out of a total of 67 participants, 31 responded to the section dedicated to web development challenges. The survey specifically asked participants only to respond if they had previously developed an interactive web site (a.k.a. web application).

With the intention of finding those issues that the respondents perceive as the biggest challenges in web development I asked them to rate a list of potential concerns on a scale from 1 to 5 (1=not a problem at all; 5=severe problem). The square markers in Figure 7 show these responses (along with those from the pre-interview questionnaire). In order to facilitate comparison, the survey responses have been scaled up to match the 1-7 scale from the pre-interview questionnaire. The (scaled-up) standard deviations vary in the range from 1.3 to 2.1.

As the average ratings suggest, no one concern stood out as generally severe; most of the average ratings were in the middle or lower half of the scale. The top issues

were ensuring security, browser compatibility, technology integration problems, and debugging. This suggests that these might be particularly common problems in web development, at least for developers at an intermediate level of expertise.

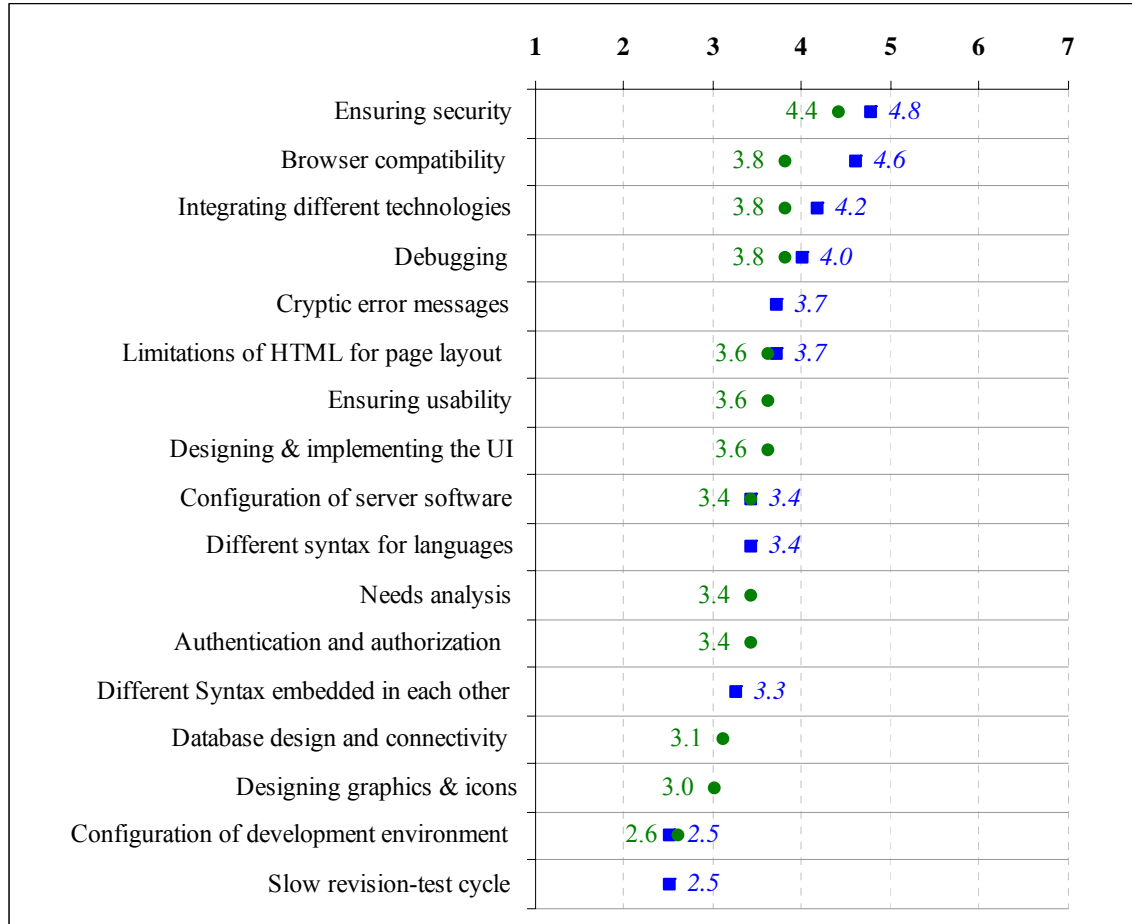


Figure 7: Responses to question about problems in web application development (1=not a problem at all; 7=severe problem). The square markers show the mean of the responses from the survey (value is right of the square marker in italics; N=31). The round markers show the mean of the responses from the pre-interview questionnaire (value is left of round marker; N=10). In order to facilitate comparison, the survey responses have been scaled from a 1-5 scale to a 1-7 scale.

Regarding: “Other problems that you typically encounter during web development:” I received the following answers (number in parenthesis indicates frequency): time available for development (2), web browsers bugs (2), race conditions (2), incompatibilities between development tools, preparation of images, hard-to-find

“random errors”, defining business/user requirements, time to learn new versions and upgrades, concurrency conditions, lack of consulting support.

I also asked developers: “*From the interactive websites that you developed consider one that was particularly challenging. What were the top 3 most challenging issues you encountered while developing this website?*” The answers to this question were very diverse and I summarized them into the following problem areas (number in parenthesis indicates frequency):

- Availability & setup of development environment and production servers (5)
- User interface layout, graphics (5)
- Integration issues (5)
- Needs analysis, user feedback and education (4)
- Database design and connectivity (4)
- Available time and funding (3)
- Concurrency (3)
- Authentication and authorization (2)
- Standard compliance, browser compatibility (2)
- Limitations of the web paradigm (2)
- Others (mentioned once each): security, fault tolerance, load issues, efficiency, maintenance of service, dealing with someone else’s code

Among this group of web developers, 12 (39%) reported using FrontPage (Microsoft 2005a) as a web design tool on a regular basis, 11 (35%) said they use Dreamweaver (Macromedia 2005b), and 5 (16%) indicated that they use Macromedia Flash (the multiple choice question with “others” option allowed for multiple selections). Apart from Microsoft Notepad which was mentioned 5 times, other tools were only mentioned once or twice.

The answers to the question: “*Describe your "dream" web application development tool? How would it facilitate development? Consider this question a "wish list"!*” were also quite diverse (number in parenthesis indicates frequency):

- Powerful layout & graphics functionality and asset management (5)
- Easy-to-use, “reads my mind”, “intelligent” (4)
- Pre-build scripts, widgets, components (4)
- Integrated toolbox that bundles everything needed for web application development (3)
- Automatic generation of clean, standard, cross-browser compatible code (3)
- Good, context-sensitive help and tips (3)
- Automatic site maintenance and reduction of tedious and redundant operations (3)
- Build-in testing and debugging tools (3)
- WYSIWIG-based with code-view option (2)
- Others (mentioned once each): clear error messages, free, changes take effect immediately, website usage tracking, tool is a native Microsoft Windows application, version control, check-in/check-out, workflow support, user has control over tool’s “intelligence”, website overview function, forms wizard

In summarizing the survey responses, no single issue stands out as severely problematic. However, ensuring security, integrating different technologies, debugging, and cross-browser compatibility seem to be the top problems from the perspective of our audience. Interestingly, Vora’s survey (Vora 1998) identified the problem of compatibility already more than 5 years ago, yet it persists. The survey has been a rather coarse measurement tool and did not reveal any details regarding the development process and general habits of semi-professional web developers. These questions were addressed by in-person interviews which were conducted later.

3.1.1.2 The Interviews

I conducted interviews with the same target audience (semi-professionals) for two reasons. First, I wanted to better understand the details of the web development process and have the opportunity to ask further questions. Second, I wanted to increase the total number of reported experiences; this is why I interviewed developers who did not participate in the survey (with one exception). Many of the questions were identical or similar to the ones asked in the survey.

I interviewed 10 web application developers in the period between May and September 2003. Out of these, 8 were conducted as one-on-one interviews, the remaining 2 (due to the unavailability of the participants) as online questionnaires with follow-up email communication. The one-on-one interviews lasted about one hour. Participants were selected by contacting webmasters of various web applications on the Virginia Tech campus as well by as contacting local web development businesses.

Prior to each interview, the participant filled in an online questionnaire which was targeted at collecting quantitative information and helped me to prepare for the in-person interviews. This questionnaire (see Appendix B.2) also contained all the main questions asked during the interview (so that participants could be mentally prepared) but participants were asked to not answer them online. The questionnaire was similar to, but much more detailed than the survey.

Five of the interviews were conducted at the workplace of the participants, three in our laboratory, and two online. However, the atmosphere was always private. All of the in-person interviews were voice-recorded and later transcribed in abbreviated form. The two participants who were not available for in-person interviews were asked to complete the questionnaire as detailed as possible online. Where necessary, I later exchanged emails to clarify and elaborate on answers. The participants included nine males and one female. Three participants were between 26-30 years old, three participants between 31-35 years, two between 46-50 years, one participant between 21-25 years, and one participant under 21 years old.

The question “*How do you rate your overall knowledge in web application development? (1=no knowledge, 7=expert knowledge)*” resulted in an average of 5.1 (SD=1.3) with only two participant rating themselves below 5. The average self-reported experience of the interview participants is somewhat higher than the (scaled-up) mean experience of the survey participants which was only 4.3 (SD=1.3).

Two participants reported that they have been developing web applications for 2 years, two participants for 3 years, one for 4 years, two for 5 years and three for more than 5 years. The participants included two full-time web developers, four IT personnel who develop web applications as part of their work, one professor who teaches web application development, two students who work in this area besides their studies, and one CEO of a small-business e-retail company who has autonomously created his e-business web site.

In the first question of the interview I asked about the most challenging issues in web development without pre-defining any categories. Top answers were (the number in parenthesis indicates how many participants mentioned the concept): finding time to develop (2), debugging (2), compatibility (2), and keeping the application maintainable (2), creating an attractive user interface (2). Many more concerns were expressed in this question and throughout the interviews such as eliciting requirements, getting people to test an application, and political issues such as gaining access to data sources.

The feedback to several rating scales from the pre-interview questionnaire is shown in Figure 7 (on page 43) and Table 6 below. Note that a 7-point scale is used throughout the interview study instead of the 5-point scale of the survey study. Figure 7 shows how the participants rate different web development concerns with regard to how problematic they are. As one would expect, the results are quite similar to the ones from the survey. The (mostly) small differences may be attributed to a rather small N of 10, to the higher level of experience of the interview participants when compared to the survey participants and perhaps to the different scales used (5 vs. 7-point scale).

Table 6: Responses to questions asked in the pre-interview questionnaire on scales from 1-7

Question (Scale)	Mean (Std-dev)
2.4. I search the web for snippets of code that I copy, paste & edit. (1=never, 7=very frequently)	3.9 (2.0)
2.5. I consult and scavenge code I have previously written myself. (1=never, 7=very frequently)	5.8 (1.6)
2.7. Do you use a HTML code validator to verify the standard-compliance of your code? (1=never, 7=always)	2.7 (2.0)
2.8. Do you check for cross-browser compatibility? (1=never, 7=always)	4.9 (2.1)
2.9. Do you check for usability? (1=never, 7=always)	4.7 (1.3)
2.10. Do you check for accessibility (for users with disabilities)? (1=never, 7=always)	2.7 (1.9)
2.11. Do you check for scalability & performance issues? (1=never, 7=always)	3.5 (2.1)
2.12. When learning about a new web technology I prefer learning from examples over learning from more general and verbose descriptions. (1=I strongly disagree, 7=I strongly agree):	5.7 (1.3)

Table 6 shows the summarized responses to different questions regarding the habits of the participants. During the interviews I asked the participants to explain their answers to the ratings provided in the pre-interview questionnaire (Figure 7 and Table 6). The responses confirm that the major concerns are security, compatibility, integration and debugging. The quote: “*How do I know it’s secure?*” illustrates the primary concern and the fact that most of the participants do not seem to have an organized approach to ensuring security.

Cross-platform compatibility is still regarded as a major stumbling block for creating rich user-interfaces. The participants are overwhelmingly conservative in the use of client-side technologies, mainly in fear of creating incompatibilities. For example, one participant remarked: “*Most of my designs are simple because of that.*” Most participants

reported that they frequently test for cross-platform compatibility (see Table 6, question 2.8). However, eight participants remarked that their testing is informal, for example they typically use the 3-4 web browsers they currently have at-hand to check the main functionality of the application.

“Remembering all the little quirks” appears to be a considerable annoyance while integrating different languages (e.g. PHP, JavaScript, HTML, CSS). Furthermore, participants remarked that keeping a growing web application consistent and maintainable is difficult.

Regarding debugging web applications, the participants report that they find it difficult (or impossible) to step through the code line-by-line and to locate the exact source of the problem. Simple print statements appear to be the modus operandi.

Contrary to my expectations only one participant seemed to be dissatisfied with the use of HTML for user interface layout. He mentioned the difficulty of creating complex layouts with HTML tables. Again, the use of advanced client-side features (e.g. CSS2 positioning) appears to be an exception among the participants.

The interviewees’ answers to question 2.5 (as well as the follow-up discussion) revealed that almost all of the participants quite often reuse code from previous projects. According to the interview responses this reuse is of an informal nature that might be characterized as a simple “copy & paste” strategy.

In addition to inquiring about the frequency of code reuse (see Table 6, Question 2.5) I asked the participants which components they reused most frequently. They responded as follows:

- HTML templates, snippets, header, footer (6)
- Various JavaScript functions (4)
- Database code (4)
- Authentication code (3)
- Validation code (2)
- Code for encoding/decoding data (2)

In order to determine what web developers regard as the key concepts within web application development, I asked the participants what questions they would raise and address in an FAQ (Frequently Asked Questions) for novice programmers. The following concepts were cited:

- Database connectivity and operation (5)
- Difference between client-side and server-side scripts; when to use one or the other (2)
- Page transition, receiving input data (2)
- Practical examples (2)
- Maintaining state (1)
- One-to-many relationships (1)
- Integration of different languages (1)
- User-centered design (1)
- Validation (1)

However, in general, the participants seemed to have difficulty answering this question even after repeated questioning. I speculate that they had mastered the basics of web development too long ago as to put themselves into the mindset of a novice. Also, they may have had little reason or opportunity to assist novice users.

The results from question 2.9 (Table 6) indicate that the participants frequently assess the usability characteristics of their web applications. As in their software debugging efforts, such evaluation is normally of an informal nature, for example asking colleagues or friends to test the application and send them feedback. Often, the participants forgo extensive testing in advance and rely instead on gathering user feedback once the application has been provided for actual use. In general, the majority of the participants conveyed that they saw no clear distinction between the activities of prototyping, development, testing, and production. Rather, the common development approach is an informal requirements elicitation phase through one or more meetings with the client and the evolutionary development of the application. Often an early

prototype is gradually developed into the final application. This finding is similar to Taylor et al.'s (2002) observations of semi-professional programmers in industry and government but in contrast to Lang and Fitzgerald's (2005) observations of professional web developers.

Checks for proper accessibility are even more informal than compatibility and usability testing. In most cases my participants follow what they viewed as "known principles" of accessible web design throughout the development process (e.g., using image-alt tags, considering table linearization by screen readers). Only two participants mentioned using accessibility validation tools like Bobby (Watchfire 2005), text-only browsers or screen-readers to verify the compliance with accessibility standards. From their comments I conclude that most developers perceive these tools still as being to cumbersome. Six participants said that they never or only rarely checked their HTML code against a code validator.

When asked what they enjoyed about web application development the participants mentioned the following factors:

- Enjoy the challenge; like building things ("*It's like playing*") (4)
- Quick feedback; ease of checking work (3)
- Quick results (i.e. being able to finish a job quickly) (2)
- Diverse work; always something to learn (2)
- Providing useful services to the user (2)
- Quick use of results (i.e. no deployment on users' machines required) (1)
- Ease of sharing (1)
- Richness of the medium (1)

Note that the *speed* of development, feedback, and results is a recurring pattern for an important "fun factor". This was one of the primary factors motivating my investigations into the concept of "Design-at-Runtime" which accelerates the development-test cycle (see Chapter 5.1).

Perhaps many of these motivational issues can be exploited to become supporting factors for end-user web development. Only one participant (CEO of a small-business e-retail company) said that he did not enjoy web development any more and resented the monotony.

In the pre-interview questionnaire and during the interview I asked the participants to identify, rate, and discuss their favorite web development tools. The tools mentioned ranged in complexity from simple text editors, to HTML-code editors like Homesite (Macromedia 2003) or BBEdit (Bare Bones Software 2003) or Emacs to WYSIWYG editors like Dreamweaver (Macromedia 2005b) and FrontPage (Microsoft 2005a), to development environments like Visual Studio (Microsoft 2003a). Because 5 out of the 10 participants named Macromedia Dreamweaver MX as their first tool of choice I will discuss it in more detail. Table 7 shows how the five participants rate Dreamweaver along the dimensions ease-of-learning, ease-of-use, functionality and overall satisfaction.

Table 7: Responses from 5 participants regarding their appreciation of Macromedia Dreamweaver MX as a web development tool

Macromedia Dreamweaver MX Evaluation (1=low, 7=high)	Mean (Std-dev)
Ease of learning	4.8 (1.8)
Ease of use	5.2 (1.3)
Functionality	5.8 (0.4)
Overall Satisfaction	5.6 (0.5)

Overall, these users of Dreamweaver seem to be satisfied with the tool. They mentioned the site management features (3), the template mechanism (2), its WYSIWYG editing style (2) and general feature-richness (2) as its main strengths. However, they also named some weaknesses. Three participants complained about the stability and reliability of the tool (it crashes or “destroys code”); one of them said that Dreamweaver “feels flimsy” (as opposed to other standard Windows productivity applications).

Another often-heard complaint was that Dreamweaver occasionally generates unnecessary complex code (while including JavaScript “behaviors”) and some mentioned a feeling of lack of control over the code. One participant remarked: “The code that gets written is not the code that I’d write myself. My code is cleaner.”

Towards the end of the interview I asked what could be done to simplify web application development if there were no limits as to changing standards, resources etc. Many issues were identified but few more than once (with the exception of consistent support for HTML, JavaScript and CSS across all platforms). The list includes: simplified debugging, introduction of high-level components like calendars, better support for reuse, better database connectivity, separation of layers (presentation, application logic), automatic maintenance of state information, and more code-assistance.

The next question inquired about the developers’ “wish-list” for their “dream” web development tool. The answers reflected the same issues named in the previous question. Participants also emphasized the desire for better integration of tools, and a responsive, visual user interface (including copy-and-paste and drag-and-drop functionality) with many predefined components. The exceptions were ideas for natural language style user interfaces, application behavior visualizations, or, at the other extreme, the total abstinence from WYSIWYG in favor of a robust text-only tool.

3.1.2 Discussion and Conclusions

The 31 survey responses were provided on a self-selection basis by students, faculty and staff associated with Virginia Tech. Nine out of the ten people from the interview study were associated with Virginia Tech. This may limit the applicability of my results although many of the issues discussed are likely to extend beyond the boundaries of the campus.

Furthermore, the foci for both studies were semi-professionals rather than professional web developers. Although I hesitate to generalize my findings to all web developers (with novices on one end of the spectrum and experts on the other) I do not see many reasons why semi-professional web developers outside of the academic environment should have much different needs and habits than our participants.

Although the two studies revealed a multitude of issues I see the following ones as most important.

3.1.2.1 Ensuring security

Web applications are vulnerable against exploits on many different levels (e.g. operating system, web server software, database, dynamic scripting language, interactions of the aforementioned). Today it is very difficult to build even a “reasonable” secure application or just to assess when an application is secure. Web developers are not confident about the security of their applications and therefore very concerned.

3.1.2.2 Cross-browser compatibility

The inconsistencies between different browsers, versions and platforms are not only a major time-sink for web developers but also seem to be the reason why most developers avoid enriching the user experience with advanced features that are only possible with JavaScript, CSS2, or Flash.

3.1.2.3 Integrating different technologies

While classical desktop applications are typically based on the syntax of only one programming language (perhaps two when considering database interactions), most web applications combine five or more (HTML, JavaScript, CSS, server-side language, SQL, and perhaps Flash, Curl, Java applets, Active X). The resulting complexity leads to code that is hard to develop and maintain. It also raises the bar for users who want to transition from static page design to more advanced web development.

3.1.2.4 Debugging

Most software developers have to deal with bugs. Web developers however face an extra challenge due to the number of technologies involved (see above) and the fact that a web application consists of a part that runs on the server and another on the client.

3.1.2.5 Developers' Habits

While the natural tendencies and habits of web developers are not a problem by themselves they can become problems if technology and tools do not account for them. As many members of our fast-paced society, web developers have little time to waste. Tedious development tasks run the risk of being circumvented or neglected. An example for a quite tedious process is ensuring accessibility. Current accessibility validation tools do not take into account that most developers are unwilling to spend much time designing for accessibility.

Humans deal with concrete examples easier than with abstract concepts. Web developers like to learn by and work with examples but today many tools start up with not more than a blank screen and a myriad of buttons. Wherever possible, web developers would rather modify existing code than rewrite code from scratch. This is particularly true for code that they know well and trust – their own. “Copy & Paste” behavior is often considered an “unclean” engineering practice, but these developers’ habits and preferences suggest that it should perhaps instead be embraced, exploited and “water-proofed” against its pitfalls.

Semi-professional developers are much more informal than the experts observed by Newman and Landay (Newman 2000). Written requirements documents and dedicated prototypes are the exception and a process of evolutionary prototyping of the final web application the rule.

The participants in my interview study like the idea of tools providing abstractions such as ready-made components that speed up development. At the same time they are very critical if the tool limits their control over the development process. Functionality that introduces hard-to-read and complex code (or as one participant calls it: “junk” code) typically fails to win acceptance.

Last but not least, I believe that the productivity and the “fun-factor” in web development would be further increased with “speedy” tools. Web developers appreciate the fact, that they can quickly test ideas, and create programs by what Rosson and Carroll call “debugging-into-existence” (Rosson and Carroll 1993). Each extra step or delay that is required for each change has a negative effect.

3.1.2.6 Recommendations for the web tool industry

Although the state-of-the-art web standards and tools are generally seen as being appropriate I argue that professional and novice developers alike would benefit from:

- Tools that assist developers in producing secure applications,
- Tools that are more robust and fast, that facilitate iterative development and better support debugging,
- Tools that provide a large library of high-level components while still giving the developer great control over the created code,
- Tools that speed up and automate tedious tasks like HTML validation, cross-platform testing, accessibility checks (which may solve the problem of the general lack of testing),
- Tools that work and act very similar to standard productivity applications like Microsoft Word, integrate well with those and readily exchange data,
- Tools that account for and support the informal tendencies of web developers to learn and work from examples, copy & paste from the web and scavenge prior projects.

Addressing the complexity caused by the plethora of web technologies and working towards better standard-compliance and cross-browser compatibility are challenges for the web engineering community as a whole but at the same are also the main barriers to overcome for facilitating EUDWeb. Chapter 5 discusses how our prototype tool Click addresses most of these challenges.

3.2 Survey of Web Developers: From Amateurs to Professionals

The findings from both the survey and interview study reported in the previous section are based on relatively small sample sizes (N=31 and N=10 respectively). Furthermore, they probe for the experiences of developers at Virginia Tech only. Finally, the two studies focused on semi-professional developers as opposed to exploring the whole range from web-design amateurs to professional web developers.

In order to address these limitations and to refine the requirements for EUDWeb we have designed a large-scale survey targeted at both informal and experienced developers. This survey has been designed, conducted, and analyzed in collaboration between Dr. Mary Beth Rosson, Julie Ballin, Heather Nash, Brooke Toward, and I. The findings presented here are published in a conference paper at the International Conference on Web Engineering (Rosson et al. 2005).

In this work, we build on the surveys from Vora (1998), Taylor et al. (2002) and Lang and Fitzgerald (2005), but with the goal of reaching out to the combined population of professional and more casual web developers. Our sampling is intentionally biased towards these casual (nonprogrammer) web developers and therefore care should be taken when viewing the results in the context of professional web development.

3.2.1 Methods

To develop a broad characterization of the current web developer population—both professional and casual—we conducted an online survey and recruited participants from a variety of web development communities. The survey was based on our prior surveys and interviews of local web developers (see 3.1 and Rosson, Ballin et al. 2004); it contained questions about web development experiences, including problems encountered; whether and how testing was carried out; desirable features or applications to incorporate in web development (e.g., databases, authentication); development style, including individual working style variations, and basic demographics. For a full list of the 37 questions see Appendix C.2.

We took two general approaches in recruiting participants. First, we contacted user groups associated with web tools (e.g., Macromedia, FrontPage); second we searched the web for other organizations that seemed to be oriented towards web use or even computer use in general. We particularly sought out organizations that might rely on informal developers (e.g., clubs or community organizations), but our survey invitation was aimed at both professional and casual developers.

We initiated contact with 591 organizations: approximately 30% product-centered groups (Coldfusion, Frontpage, etc.), 20% platform-centered (Mac, Linux, etc.), 38% hobby or ‘computer club’ type groups, and the remaining groups falling into language-oriented (e.g., ASP), professional/networking organizations and specific websites. We sent our email invitation to the listserv contacts, asking them to forward it to their members; the email summarized the study, data security/privacy, and the drawing for cash prizes (10 prizes of \$50) used as an incentive for participation.

3.2.2 Results

We received 334 responses to the survey. In the following, question numbers refer to the actual position in the survey, so that the interested reader may integrate the results reported here with the full survey and summary results available in Appendix C.2. Note that percentages reported in this paper are the percentage of respondents who answered a particular question, not the percentage of the entire survey population with missing responses. Many respondents skipped one or more questions, so we follow the norm of including the relevant sample size as each percentage result is reported.

Interestingly, the answer to whether or not a respondent self-identified as a “programmer” was not often a useful grouping variable for the web activities and problems summarized here. For this reason the results discussed use the entire dataset.

3.2.2.1 Participants

The survey population included both men and women (70% and 30% respectively); most respondents (86%) reported their race or ethnicity as white/Caucasian. This sample of web developers was relatively highly educated: 29% of

respondents reported that they had completed an undergraduate degree and an even larger proportion (35%) reported completing at least some post-graduate education.

There was considerable age diversity in our sample (likely due to the survey's bias towards informal web developers). Interestingly, the single largest group of respondents age-wise was those who identified as age 60 or older (21%). In combing for computer related groups to whom we wanted to promote the survey, we discovered many groups oriented towards or run by senior citizens; this may explain the large proportion of older respondents. Other respondents were spread relatively evenly across age categories of 26-30, 31-35, and so on up through the age group 56-59. Only 6% of the sample reported their age as 25 or younger.

A small majority of respondents (54.7%) reported that “work” was the most common reason for them to develop and maintain websites. This is interesting as it emphasizes that, although considerable web development is being carried out in professional contexts, there is a sizable number of projects underway for other purposes. The two next most common motives were “special interest/hobby” (16.6%) and “civic, volunteer, or community work” (12.4%).

3.2.2.2 Perceived Value of Web Functionality

One question aimed at understanding web developers' current needs asked them to rate the perceived value of a number of predefined features (Figure 8; these items were developed through our pilot studies). As indicated in Figure 8, access restrictions, online databases, member registration systems, and online surveys/forms are seen as particularly valuable to our respondents, all being well above the mid-point on a range from 1 (not valuable) to 5 (extremely valuable). Communication-oriented features like discussions and chat are seen as relatively less valuable.

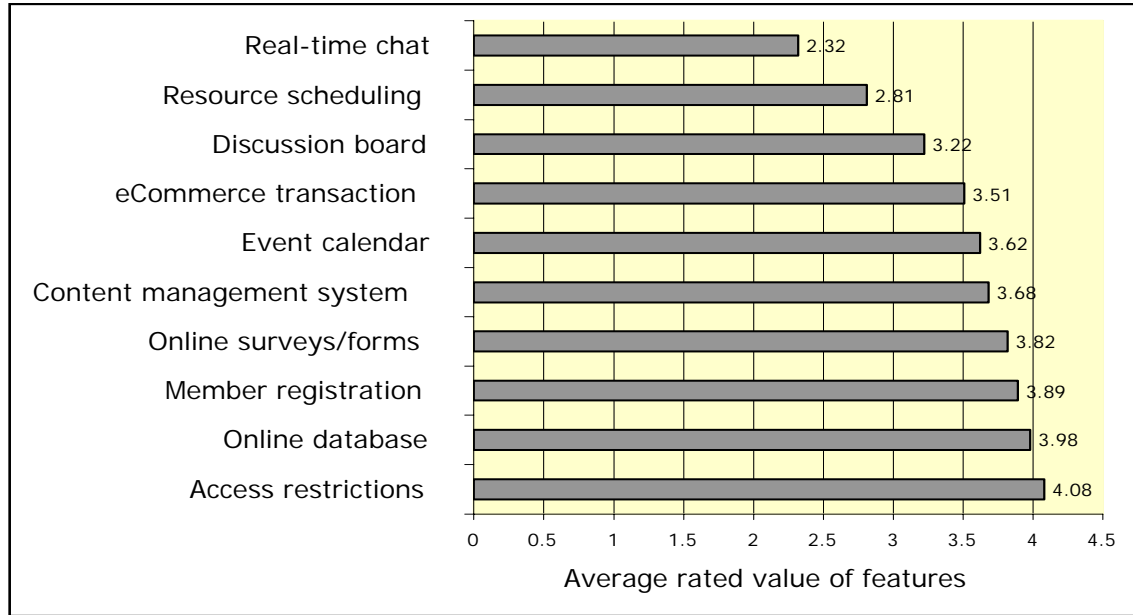


Figure 8: Results from question 5: “The following question asks you to judge the value of these same 10 features in your web development projects, regardless of whether you have worked with them yet or not.” (N=314 to 318)

3.2.2.3 Characterizing the Web Development Process

To gain insight into the typical web development process and attitudes towards web development, question 16 asked the respondents to rate their agreement with a series of statements (1=strongly disagree, 5=strongly agree).

Our respondents tended to agree with the statement: “*I spend a lot of time making sure my site's layout, formatting, content, and interactive elements are just right before I "go live"*” (mean=4.18, SD=0.93, n=274). They voice similarly strong agreement with: “*After my websites "go live", I check back frequently to make sure that everything works like it should (links, images, forms, etc.)*” (mean=4.11, SD=1.0, n=274). These responses suggest that attention to the details of a web page is high on these developers’ list of concerns.

Respondents tended to disagree with the statement: “*When taking on a new web project, I immediately start constructing pages*” (mean=2.43, SD=1.25, n=272), implying that they take steps to plan their project before jumping into building web pages.

However the statement: “*When working on a web site, I have a systematic process I follow*” evoked a rather neutral response, only slightly biased towards “agree” (mean=3.56, SD=1.12, n=273). This is an area we hope to further explore in later research.

Most respondents also agreed with the statement: “*As I work on a web project, I think about how I might come back later to change or expand it*” (mean=4.16, SD=0.96, n=274). This is a promising result as it implies that they are planning for enhancement or other maintenance activities. See Table 8 for details.

Table 8: Question 16: statements ranked from 1 (strongly disagree) to 5 (strongly agree)

						Average
I spend a lot of time making sure my site's layout, formatting, content, and interactive elements are just right before I "go live"	1%	4%	17%	32%	46%	4.18
	(3)	(11)	(47)	(87)	(126)	(n=274)
After my websites "go live", I check back frequently to make sure that everything works like it should (links, images, forms, etc.)	0%	7%	20%	24%	47%	4.11
	(1)	(20)	(56)	(67)	(130)	(n=274)
When taking on a new web project, I immediately start constructing pages	29%	28%	20%	15%	7%	2.43
	(79)	(77)	(55)	(41)	(20)	(n=272)
When working on a web site, I have a systematic process I follow	2%	18%	28%	26%	26%	3.56
	(5)	(50)	(77)	(70)	(71)	(n=273)
As I work on a web project, I think about how I might come back later to change or expand it	1%	5%	16%	31%	46%	4.16
	(4)	(13)	(45)	(86)	(126)	(n=274)

Question 15 was, in part, targeted at the issue of code reuse and participants were asked to rate how often particular statements are true (1=hardly ever; 5=quite often). The statement “*I consult and reuse/copy code I have previously written myself*” received a relatively high rating (mean=3.90, SD=1.36, n=273). This can be contrasted to their ratings for reusing others’ code: “*I search the web for snippets of code that I can directly copy, paste and edit*” (mean=3.01, SD=1.33, n=273).

3.2.2.4 Web Development Tools

Question 6 asked: “*What is the primary development tool you use for working on your site(s)?*” 42.1% of the respondents cited Macromedia Dreamweaver. Microsoft FrontPage tied with HTML editors (BBEdit, Homesite etc.) at 12-13% each, followed by Text editors such as notepad or vi with 9.7%. No other tool exceeded 3%. Note that the relatively high proportion of Dreamweaver users is likely biased by our recruiting strategy (the Macromedia user groups were large and had good response rate). Of course, this predilection for Dreamweaver should also be considered when interpreting responses to questions concerning tool likes and dislikes.

Question 8 asked: “*What are the three things you like MOST about your primary web development tool?*” Three open response fields were provided and we received 286 responses for the first, 272 for the second, and 246 the third—a total of 804 individual responses, typically just a few words long.

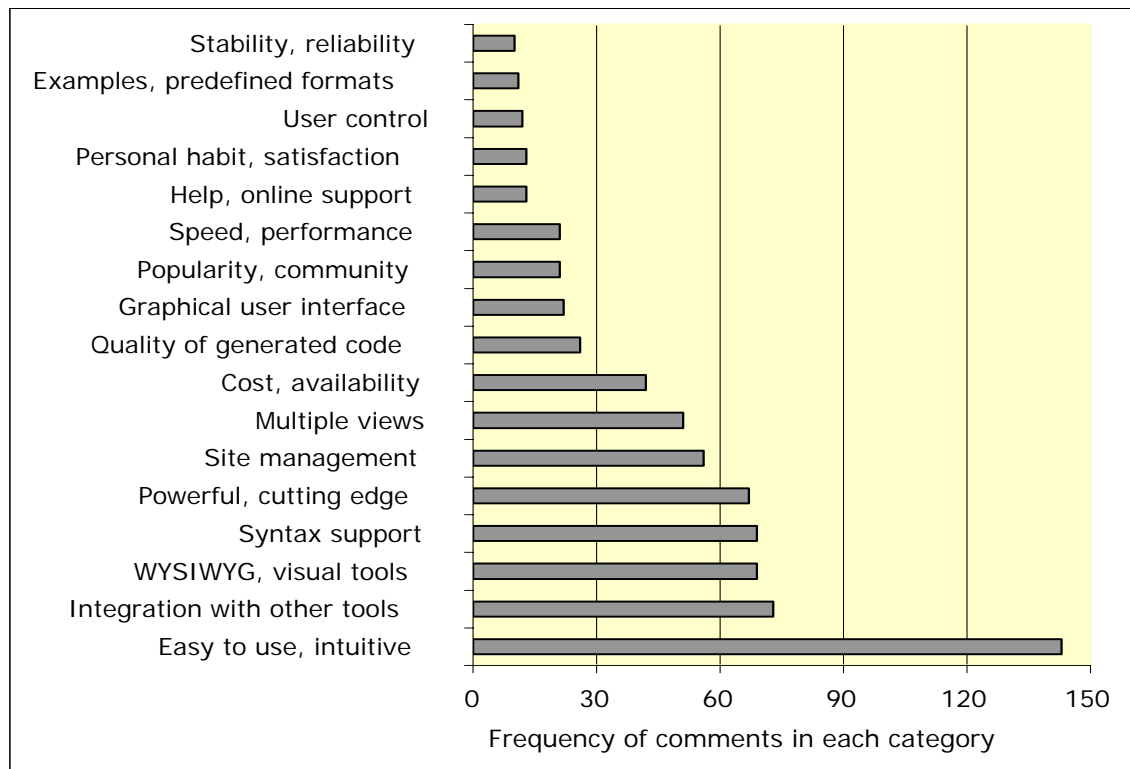


Figure 9: 90% of responses to question 8 “What are the three things you like MOST about your primary web development tool?” were coded into 17 categories

We coded the results by first scanning all responses and establishing categories. Next, we coded all comments according to the previously established categories. Figure 9 visualizes about 90% of grouped comments (719 responses). 10% of developers' comments were coded as "other" because they were too diverse to be grouped in a meaningful fashion.

Question 9 asked: "What are the three things you like *LEAST* about your primary web development tool?" Again, three open response fields were provided and we received 259 responses for the first, 193 for the second, and 143 the third, for a total of 547 individual responses (excluding 48 responses such as "nothing" or "n/a").

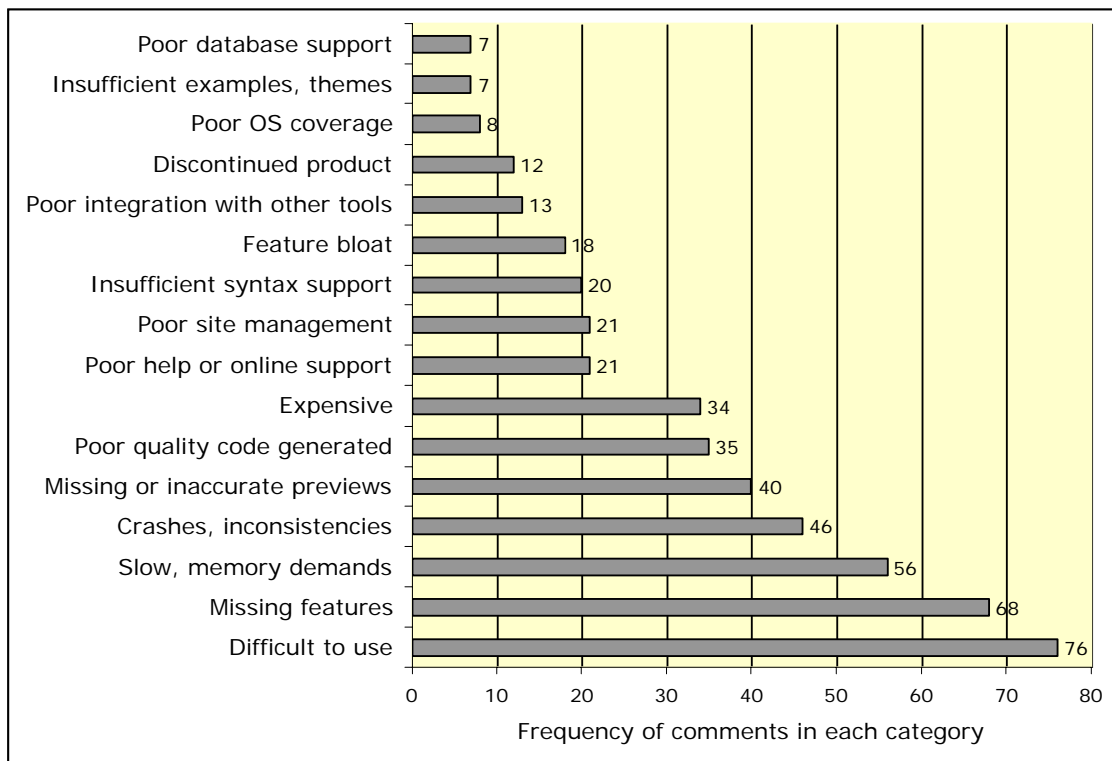


Figure 10: 88% of responses to question 9 "What are the three things you like *LEAST* about your primary web development tool?" were coded into 16 categories.

We used a similar coding strategy as in the previous question, resulting in 16 categories. Not surprisingly, many of the comments made in response to things liked least (see Figure 10) can be seen as the inverse versions of things liked most (e.g., the number

one group in both cases is related to the rather general evaluation of ease of use). Interestingly however, while feature coverage was rarely mentioned as a reason to *like* a tool, it was the second most common category for *disliking* a tool.

3.2.2.5 Problematic Development Situations

To explore the problems that web developers may encounter we asked our respondents to rate eleven problems according to how frequently they occur. As with the features probed in Question 5, this list of issues was based on our earlier surveys and interviews that probed problems in web development.

Figure 11 shows the results. None of the issues stands out as a particularly frequent problem, except perhaps of “*getting content in a timely manner from others...*” (mean=3.32, SD=1.41, n=272). This is interesting in that it is the one issue that is very much related to the developers’ collaborative context—that is, to their dependencies on others.

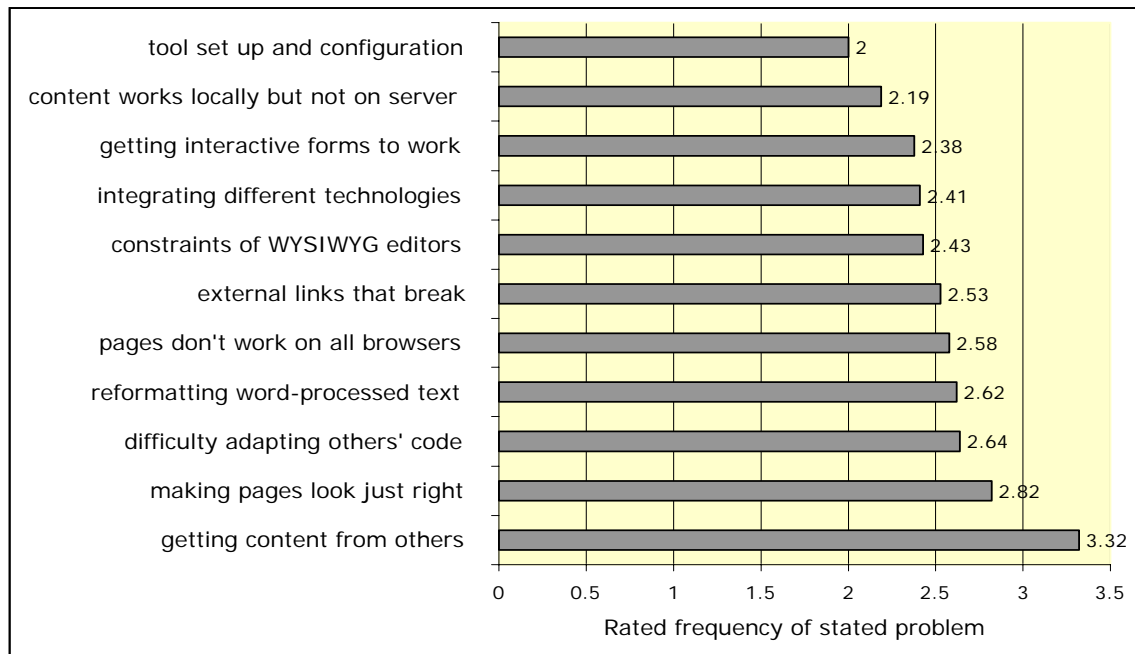


Figure 11: Responses to Question 14 “How often do you experience problems with the following kinds of issues that sometimes arise in web development work? Please use a scale from 1 (one) to 5 (five) where 1 means hardly ever, and 5 means quite often.” (n=267 to 276)

Rosson et al.'s (2004) interviews with community webmasters had indicated that this was a particularly vexing problem for these relatively informal web developers; it appears that it is a similar problem for a much more diverse population.

3.2.2.6 Attention Directed to Quality Control

To understand the extent to which quality control is a concern for our sample of web developers, we asked respondents to tell us how often they performed certain testing tasks (1=never, 5=always; “*When working on websites, how often do you test to make sure...*”; Question 12). An overwhelming majority of respondents agreed that they evaluate the general usability of their websites always or almost always: “*...it is easy for users to do what they want to do on the site and to find what they might be looking for (usability)*” (mean=4.33, SD=0.93, n=276). However, they seem to be much less likely to worry about universal access: “*Users who might have disabilities will be able to use your site (ADA compliance, section 508, Equal Access, etc.)*” (mean=2.75, SD=1.41, n=276). Although most developers appear to test for platform and browser compatibility, not all of them do so routinely (“*It will work across different operating systems and different web browsers such as Internet Explorer, Netscape Navigator, Safari, etc.*”; mean=3.75, SD=1.26, n=276).

The three items analyzed above represent a relatively superficial assessment of developers' testing processes. A deeper analysis of an open-ended question related to the respondents' testing strategies is given in our conference paper (Rosson et al. 2005).

3.2.2.7 Learning New Web Development Skills

We asked participants to rate how likely they would be to consult particular resources for assistance in case they needed to learn something new (Question 11; 1=not likely; 5=very likely). “*FAQs, books, or tutorials*” were rated most highly (mean=4.53, SD=0.88, n=257), followed by “*Examples of similar sites from which you can get ideas and copy code*” (mean=3.97, SD=1.18, n=259), and “*A friend or coworker who knows how to do it*” (mean=3.76, SD=1.26, n=259). Respondents indicated that they would be less likely to consult sources such as interactive software wizards, software agents, seminars, or support hotlines.

3.2.3 Discussion and Conclusions

Our survey yielded a diverse sample of respondents—a mixture of professional and casual developers, representing a wide range of ages, who seem to be pursuing projects in rather different web development contexts. However, despite the variation among the respondents, there are a number of implications that we see in our results.

For example, with respect to perceived value of different web functionality, most developers rated access restrictions, online databases, survey and forms as valuable elements for their web presence. Unfortunately, many of the features and applications that developers see as valuable are not easily implemented. For casual or informal web developers, providing access restrictions may be conceptually simple and obvious, but current tools make its implementation quite challenging. One of the other highly valued features—online databases—seems to be even more difficult to implement than access restrictions. Again, although the interactions with databases may be conceptually simple (e.g., consisting of overview and detail pages, a search function and some data input and edit forms), they are typically beyond the implementations skills for casual web developers. Current web development tools do not sufficiently abstract technical concepts such as session management, input validation or URL parameter passing. This requirement underscores an opportunity to develop more powerful web development tools designed for end users, tools that would raise the ceiling on what is achievable for nonprofessionals.

Our analysis of questions about respondents' web development process suggests that—at least in our sample—the prototypical web developer is meticulous and particular about the quality of the web sites she produces and maintains. Also, generally our web developers seem to invest some thought before embarking on a new project rather than implementing web pages ad-hoc, although they may or may not follow a strict process. Web developers also appear to frequently reuse code they wrote earlier but only occasionally search the web for example code to copy and use. These general findings are an encouraging indication that even an increasingly diverse web developer population is attuned to the “traditional” concerns of software engineering such as design and quality assurance.

The responses to the question about features most liked in web development tools show that this sample of web developers value ease of use as the most important property of a web development tool. They also clearly appreciate a tool that integrates well with other tools and provides frequently needed site management features such as integrated file upload. While they highly regard powerful WYSIWYG visual design and code generation features, they also demand support for viewing and editing, testing, and previewing the code behind the scenes. They appreciate code auto formatting and tag completion but at the same time expect to have full control over the layout of hand-written code.

At the same time, the responses to the question about what developers least like about their web development tool(s) show that many web developers are still not satisfied with usability aspects of their tools. While many respondents request more powerful features, such as more extensive WYSIWYG support, others complain about feature bloat. Across all comments, concerns about performance problems and faulty behaviors take the lead in complaints about tools. Another common complaint refers to automatically generated code that appears “messy”, “bloated”, and non-compliant to standards.

Regarding the typical problems that web developers encounter we were not able to detect any major distinctions in developer’s experiences. Only the issue of “getting content in a timely manner from others...” was rated above the mid-point on a frequency rating scale. This concern is interesting, as it is much more social in nature (being dependent on a colleague for input) than most of the other concerns. It may be that social problems of this nature plague everyone, whereas the other listed problems are much more dependent on the types of applications or work contexts in which developers operate. Our future research might investigate these problematic aspects of web development more carefully, for example also probing perceived severity of individual problems, connecting problems to developers’ working context, and providing an opportunity to describe problems in an open-format question.

Questions about the quality control process show that the vast majority of developers from our sample routinely validate website usability (although the procedures

followed are generally ad-hoc and informal in nature) and sometimes check for cross-platform issues but rarely for accessibility problems. These accessibility checks may be omitted because of lack of awareness and concern, but it may be at least partly due to the relatively tedious and time-consuming tool support for such checks (too verbose, reporting many false positives; lack of automation).

3.3 Concepts and Components of Typical Web Applications

The first survey showed that basic data collection, storage, and retrieval application are an important subset of webmasters' needs (see 3.1.1.1). As a further step in the requirements analysis and in order to scope the functional requirements for EUDWeb tools which target these web applications, I surveyed and analyzed existing web applications of this kind. The purpose of this work was to determine the components, concepts, and functionality needed to implement simple data collection and management applications.

This assessment is important in determining the features needed to make an EUDWeb tool sufficiently powerful. Rather than limiting myself to applications reported by the survey respondents (see 3.1.1.1), I analyzed existing web applications. I recognize that neither this analysis of existing applications nor survey and interview data will provide a full picture of the applications nonprogrammers might want to develop in the future. However, I believe we can obtain a reasonable approximation by looking at what has been done in the past.

Obviously, it is impossible to review all applications on the public World Wide Web and all private Intranets. Therefore, I restricted the analysis to a sample of web applications available at Virginia Tech. Google and its filtering capabilities (e.g. "filetype:asp site:mysite.edu") were used to find applications in use at Virginia Tech. Using file extensions that indicate dynamic content (.asp, .aspx, .php, .php3, .cfm, .jsp, .pl, .cgi) yielded a large number of cases. I disregarded simple dynamic websites (scripting only used for navigation, header & footers, no database) and focused on those applications that were close to the needs expressed by the survey respondents, ending up

with a set of 61 example applications. These included databases for people, news items, publications, job offers, policies, conference sessions, plants, service providers and so on.

I reviewed the applications that were publicly accessible and constructed a list of concepts and components found within these basic web applications (see Table 9). The components, concepts and functions derived can be viewed as high-level equivalents to low-level language constructs, predefined functions, objects and methods in classical text-based programming languages (e.g. for-loop, while-loop, if, print).

Again, I do not see this as an exhaustive list of features, but rather a pragmatic technique for scoping initial prototyping efforts. I expect the list of elements to change and grow along with our knowledge about web applications and the progress of technology.

Table 9: High-level components, concepts & functionality of typical basic web applications

Concept/ Component	Description
Basic concepts and services	
Page	Set of components and data that is visible at one point in time
Database	Persistent data store
Data	Unformatted or formatted text, images, files stored in a database or file system
Record	A set of related data; in its simplest form it corresponds to one row in a database table; in the more general form it may span multiple database tables
Recordset	The sum of all records pertaining to a certain concept; a more complex application may host more than one recordset
Data persistence	Data entered on different screens can persist for the duration of a user session
Input validation	Constraints on the valid options for input components
Authentication	Users can be identified via user-id and password; also user-id/name mapping
Authorization	Parts of (or entire application) restricted through definition of user authorization rules
Conditions	Can be applied to components to modify their behavior according to the context
Data manipulation	Provides operators for calculations and string manipulation

Layout and visual design components	
Decorative elements	Static text, images, separator lines, boxes and other elements for layout
Input components	
Checkbox, Radio button, Listbox, Text field, Link	Basic input, output and action components
Button	Invokes an action such as <i>go to page</i> , or <i>save to database</i> , <i>send email</i> etc.
Navigation menu	Offers a flat or hierarchical set of choices and invokes an action
Output components	
Dynamic output	Output of variable content; text or image (e.g. current value of a text field or database field)
Dynamic table	Output of data in tabular format including the following options: <ul style="list-style-type: none"> • Sort: Sorts table by ordering records according to one column • Paging/Browsing: Splits long table and offers record navigation • Summary: Displays summary information for one or more columns • Edit/Delete: Displays links to modify the displayed data record • Nesting: Table may be nested within a cell of another table displaying all records from recordset B within one record of recordset A
Repeating section	Output of data, layout, and input component in a repeating fashion including the option of nested repeating sections
General concepts and features	
Overview-Detail-Relationship	Overview page presents overview/summary information; Detail screen presents one particular record
Recordset Filter & Search	User specifies filter; Results are displayed on the same or a different screen; Includes Boolean operations; Related to the concept of a dynamic table
Add record	Adds a record to one or more database tables
Update record	Updates a record in one or more database tables
Delete record	Deletes a record from one or more database tables
Messaging	Send e-mail notification messages to administrators or confirmation messages to users
File upload	Upload of images, PDF documents and other file-based resources

3.4 Summary and Conclusions

Table 10 summarizes the key findings concerning the behavior of (semi-professional) web developers, current barriers to web application development, and finally lists developers' requirements for their "dream" web development tool.

Table 10: Web developers' behaviors, barriers to development, and a "wish list for the dream tool"

Semi-professional Web Application Developers...
<ul style="list-style-type: none"> • Start new projects with a planning phase, even though it is likely to be informal • Use an evolutionary prototyping approach up until web application reaches production quality • Are meticulous about the quality of applications they develop and like to have control over code • Often check for cross-browser compatibility and usability but in an unsystematic/informal fashion • Often avoid the use of sophisticated technologies because of cross-platform compatibility concerns • Rarely test for accessibility concerns, do not usually use accessibility tools • Frequently reuse their own code (e.g. HTML snippets, JavaScript functions, database code) • Prefer learning from examples over learning from general, verbose descriptions • Value tool features: ease-of-use, integration, WYSIWYG, code editing, flexibility, stability, speed
Major Barriers to (End-User) Web Application Development include...
<ul style="list-style-type: none"> • <i>Social</i> issues: <ul style="list-style-type: none"> • Needs analysis • Getting content from others in a timely manner • Getting feedback from users • User education • <i>Technical</i> challenges: <ul style="list-style-type: none"> • Overall complexity • Ensuring cross-platform compatibility • Integrating different technologies • Ensuring security • Debugging • User interface and graphics design

- Database design and connectivity
- Availability & setup of development environment and production servers

The “Dream” Web Application Development Tool...

- Is easy-to-use, “reads my mind”, “intelligent”
- Works and acts similar to standard productivity applications (desktop-based, WIMP metaphor)
- Offers powerful layout and graphics functionality and asset management
- Includes pre-build scripts, widgets, components
- Automatically generates clean, standard-compliant, cross-browser compatible code
- Automates tedious tasks such as HTML validation, cross-platform testing, and accessibility checks
- Integrates all tools needed for web development (layout, graphics, code, DB, publishing)
- Offers good, context-sensitive help and tips
- Facilitates debugging
- Is fast

Web application development poses a number of challenges. Semi-professional developers particularly emphasize the problems of ensuring and validating application security, cross-platform compatibility, debugging, and the integration between diverse technologies. From the perspective of EUDWeb the complexity/integration problem is likely to be the most important issue and highest entry barrier for nonprogrammers. A web application developer must know and be able to combine a considerable number of languages with different syntax (e.g., HTML, CSS, JavaScript, Java, SQL), tools (e.g., web editor, DBMS, web server) and concepts (e.g., client-server, session management, publishing). Furthermore, although a novice developer may not be aware of the requirements for and problems involving security and cross-platform compatibility these issues are important nonetheless and either need to be taught or hidden (the approach advocated by Click, see Chapter 5). Moral and legal obligations underline the necessity for universally usable and accessible applications – again a requirement an end-user developer may not even be aware of.

Apart from the technical challenges web development also exposes a number of social issues. The surveys and interviews have shown that web development is a highly

collaborative process. The main barriers here are communication barriers such as getting requirements, feedback, or contents from stakeholders. These issues may not be as problematic for EUDWeb since it seems likely that projects developed by end users are of smaller scale and involve fewer external stakeholders. Nevertheless, my experiences with an end-user survey development tool deployed since 2002 at Virginia Tech (Rode 2002) have shown that collaborative development is very common and should be supported by EUDWeb tools.

Finally, novices and semi-professional developers alike ask for tools that are easy to use, offer libraries of high-level components without reducing flexibility, and tightly integrate all aspects of development. Last but not least, they enjoy the speed of web development, a fact exploited by the concept of “Design-at-Runtime” described in Chapter 5.1.

This chapter has identified the main entry barriers to web development in general and EUDWeb in particular. The following Chapter 4 analyzes the expectations and “natural” mental models of nonprogrammer developers. The mismatches between their mental models and current web technology highlight further entry barriers to EUDWeb.



Chapter 4

Mental Models of End-User Web Developers

We can build better end-user development tools if we know how end-user developers think. If a tool works in the way that a tool user expects and it feels “natural” from the beginning it is likely to be easy to learn and use. Alternatively, a tool can be designed to reshape the way that end-user developers think about a problem. In either way, it is beneficial to know the starting *mental model* of the tool user. In this context, mental model is meant to characterize the way that people visualize the inner workings of a web application, the cognitive representations they hold of the entities and workflows comprising a system. A person’s mental model is shaped by his or her education and experience and will evolve as he or she continues to learn. The concept of “natural” or “naturalness” (Miller 1974; Pane, Ratanamahatana et al. 2001) as applied to software development technology refers to the mental model that users hold before they start learning to use a tool or programming language.

What are the mental models of my target audience and how detailed are they? I report two studies carried out to answer this question. The studies adapt the methods of Pane, Ratanamahatana, and Myers (2001), who considered the same question of naturalness in the context of a programming language for children; they began by studying how children and adults use *natural language* to solve programming problems. They used the results of these studies to design a programming environment that offers concepts closer to the *natural* mental model of end-user developers (see 2.3.3). Following this general approach, I investigated how nonprogrammers describe the behavior of web

applications in natural language. The findings from this work have guided the design of Click, the prototype EUDWeb tool, as is discussed in Chapter 5.

4.1 Exploring End Users' Concepts and Language Use

Our first efforts at exploring end users' mental models [MMODELS-1] (Rode and Rosson 2003) were aimed at investigating the language, concepts, and the general level of problem-solving that end users employ when solving web programming problems. We wanted to find out how – under ideal circumstances – end-user developers would specify and implement a web application; that is what development techniques would feel most natural to them. We were concerned with their use of language for the specification of common user interface elements (such as text fields, links, data grids) as well as their use of language and diagrams for the specification of the application's behavior. In particular we were interested in how these users would describe web-specific data processing—e.g., client-server interaction, HTML generation, the web's stateless nature, and so on.

4.1.1 Participants and Methods

Ten participants were sampled from a population of university webmasters who had reported in a previous survey that they had significant experience in web authoring but none or little in programming. Five were female, and five male. Pre- and post-study interviews revealed that one person had more programming experience than initially reported (use of Macromedia ColdFusion for a simple web application).

We recruited these participants for a two-part paper and pencil study. First, they labeled screen elements in a series of screenshots (which helped us study their language used to refer to common UI elements), and later they specified the application behavior (which helped us study their natural mental model). I created a simple web application (member registration and management) for the study (see Figure 12; see Appendix D.4 for all screenshots).

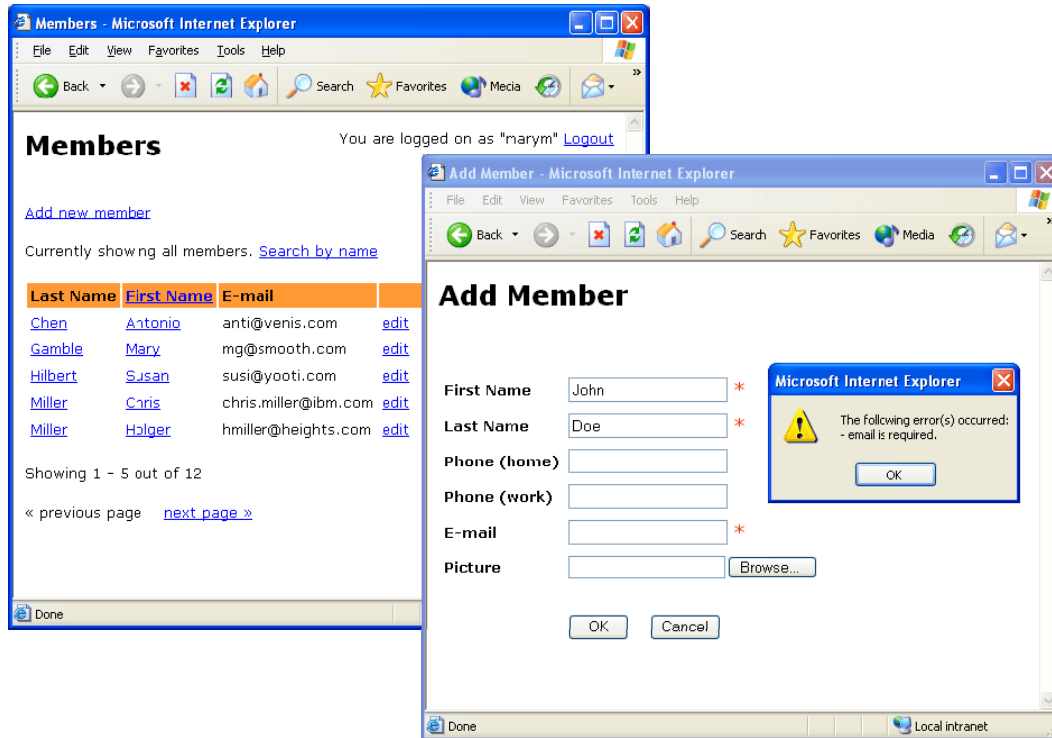


Figure 12: Two screenshots of example application used for MMODEL-1

Participants were given a general introduction to the goals of the study, then asked to view and label all elements of three screenshots from the application (login, member list, add member). The labeling instructions (see Appendix D.2) included a sample labeled image (a room with objects), including nested items (see Appendix D.3). This first phase of the study was intended to inform us about the language our audience uses to reference visible screen elements. Figure 13 shows an example of a labeled screenshot.

Next, participants were asked to explore the application until they were comfortable with how it worked. After this familiarization phase, participants were given seven user tasks (login, paging, user-specific listing, add member, sort, search, delete) and asked to “teach” these behaviors to a “magical machine”; the machine was said to understand screenshots but not know which elements are static and which respond to users’ actions. A paragraph of text within the written instructions explained this scenario to the participants (see Appendix D.2).

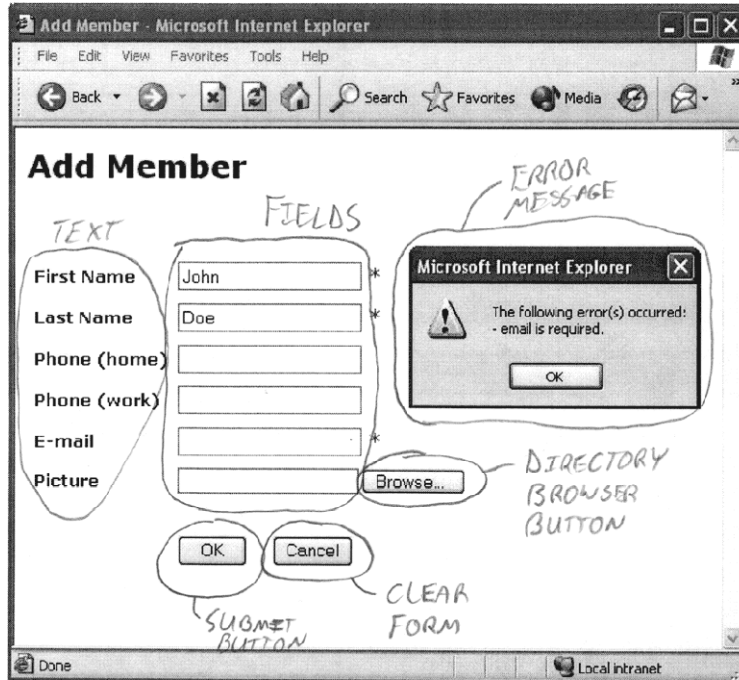


Figure 13: Example of an annotated screenshot of the “Add Member” dialog from the member registration application (MMODELS-1)

Each of the seven tasks (see Appendix D.2) was illustrated by concise instructions that were designed to guide the user without biasing their response – for example, task 4 had the following description:

*Add a new member (just make up some data). Assume you do **not** have an e-mail address. Continue with “OK”. Now enter an e-mail address. Continue with “OK”. Describe how the web application behaves.*

The interactive application was always available for further exploration or reference. Participants wrote responses using screenshots and blank paper (see example in Figure 14). I emphasized that they were free to choose how to communicate with the magical machine (using written words or sketches), but also that they should fully specify the application’s behavior. We wanted to see what end users consider sufficient as a behavior specification.

4. The fields for first & last name, and email are required fields. If one of those fields is left blank the programming in the page stops the information from being sent to the server when you hit "OK", the submit button. The page then produces an error message telling you which missing fields are required. Then clicking "OK" in the error box allows you to function again within the page. Once all required fields are entered and you click "OK", the info is sent to the server and the page is redirected to a new one displaying some of your entered information.

Figure 14: Example of a participant's description of the behavior of the "Add Member" dialog from the member registration application (MMODELS-1)

4.1.2 Results

Participants spent an average of about 90 minutes total on both parts of the study. The participants' annotated screenshots and written notes showed a general familiarity with "visible" elements of web applications (e.g., page, link, data table). Given these users' background in web authoring, it was not surprising to find that they used terms common in WYSIWIG web editors to label screen elements. A sample of these common elements appears in Table 11.

Table 11: Examples of labels chosen by the participants of MMODELS-1. Numbers in brackets denote the number of participants who chose the particular label.

Screen element	Labels provided by the participants
Web page	page [4], page and screen interchangeably [4]
Text input field	field [3], input field [3], box or text box [2]
Member list	table [3], dynamic table [1], data table [1]
Member add, etc.	link [4], action [3], option [3], function [3]

When describing the application's behavior, participants tended to combine procedural steps and declarative statements. They used declarative statements to specify constraints on behavior (e.g., "certain fields are required"). Procedural statements often conveyed a test and result (e.g., "If the password is incorrect, that field is cleared") or a page transaction (e.g., "Type the correct password into the field and Enter; this action opens the Members page"). With the exception of one participant, no one mentioned conventional programming constructs such as variables and loops in the behavior specifications. Where looping constructs are required (e.g., when authenticating a user), the participants specified one iteration, seeming to expect that it would apply (i.e., be repeated) as necessary.

Only three participants included any description of what happens "behind the scenes" in a web application (e.g., mentioning interactions with a server). Even these participants made no effort to describe page transactions in detail (e.g., no one discussed how information is forwarded between pages). Most participants (7 of 10) referred to application data as a database; another talked about a file. This is consistent with their general use of a "technical" vocabulary. However, only one included communication between the application and database ("sends command to the database on the server telling it to query"). Though comfortable with the concept of a database, the others seem to see it as a placeholder for a background resource.

In a similar fashion, users often referred to a "member list" or a "member" as if these abstractions are simply available for use as needed; no one worried about how an application obtains, stores, or manages data. We thought that the search and sort tasks might evoke informal descriptions of algorithms, but most participants focused on the desired result (e.g., what the user sees next in a table) rather than on how a data listing would be obtained. Six users seemed to assume that the "magical machine" manages user authentication; four offered as a detail that user data must be checked against a list or table of valid IDs.

4.2 Mental Models of Typical Web Development Concerns

One problem with the first study of concepts and language for web programming (MMODELS-1) was the generality of the problem-solving it required: I asked participants how web programming tasks would take place but did not direct their attention to specific constructs (e.g., iteration, input validation). Thus the results pointed to a few general (and often rather predictable) tendencies in end users' mental models. For example, the participants in MMODELS-1 tended to understand and describe web programming at a high level of abstraction. Concepts like session management and database were assumed to work "out-of-the-box". This first study showed the level of abstraction at which end users are likely to think at but did not provide insight into how particular web engineering aspects are understood. I wanted to probe more deeply, to explore how end users might conceptualize the specific components and features I had catalogued in my analysis of existing database-centric web applications (e.g., input validation, database lookups, overview-detail relationships; see 3.3). We carried out a second study (MMODELS-2) to explore these issues (Rode and Rosson 2004). Our goal was to determine how end users *naturally think* about *typical concerns* in web application development.

The rationale for using this methodology is that by studying the natural language procedures (Miller 1974; 1981) that nonprogrammer webmasters can generate about how a specific feature or concept works, we can develop approaches for supporting this feature that will be intuitive to this target user population.

We wanted to begin our investigation with programming concerns that are commonly addressed by web developers when creating a web application (particularly basic data collection, storage, and retrieval applications). Thus we selected a set of concerns that appeared frequently in an earlier analysis of 61 existing web applications (as discussed in 3.3). As an experienced web developer, I also relied on my personal experiences to judge what programming concerns are most important for applications within my target domain.

We selected 12 web development concerns to focus on in our second study:

- Session management (i.e. save data as the user moves from one page to another)
- Input validation (e.g., verifying the format of an e-mail address or ZIP code)
- Conditional output (e.g., only display a “logout” link if the user is logged in)
- Authentication and authorization (e.g., restricting access to data)
- Database schema (i.e. internal format used to store data)
- Database lookup (e.g., resolve a user-ID to a user name)
- Overview-detail relationships (i.e. show a listing of all records on one page and details of one particular record on another)
- Normalization and use of foreign keys (i.e. how to store data non-redundantly)
- Uniqueness of data records (i.e. the use of keys)
- Calculating database statistics (e.g., total number of records)
- Search (e.g., find a person by name)
- Timer (e.g., send notification emails)

4.2.1 Participants

We recruited 13 participants (8 female, 5 male) who, in a screening survey, identified themselves as having at least some knowledge of HTML and/or of a WYSIWIG web editor (≥ 2 out of 5 on a rating scale) but very little or no programming background (an essay-type question asked respondents to detail *any* programming experience). We later eliminated the data from 1 male and 1 female participant as our conversion revealed that they had more programming experience than originally indicated. In the following only the data from the 11 remaining participants is reported.

The screening survey did not question participants for their experience with databases. However, during the welcoming phase of the study the participants were explicitly asked about their level of database knowledge. All but one participant indicated that they had at least some experience with databases (9 with Microsoft Access, 1 with FileMaker Pro). Although our sample size is too small to draw strong conclusions, this seems to indicate that casual web developers (my target audience) are very likely to have database experience. Assuming that this finding can be replicated in a more diverse

sample, EUDWeb tools may be able to expose database concepts without overwhelming their users. Note though that the interviews that followed the study suggest that the level of database understanding is novice to intermediate rather than expert.

4.2.2 Methods


The goal of this study was to better understand how webmasters with no programming experience are able to imagine how a range of computational processes might be carried out by an interactive web application. Probing naïve expectations of this sort is a challenge, because the facilitator must provide enough information so that a nonprogrammer can understand what aspect of the application is being called out for attention, but not so much that the inner workings of the application are revealed. So as to describe the application feature of interest in as concrete a fashion as possible, I presented and asked questions about nine scenarios (for full list of scenarios see Appendix E.2), each describing one or more programming concerns related to a fictional web application – an online video library system.

Each scenario consisted of a mock screen shot, a short paragraph explaining what the mock screen depicts, and a series of questions. As an example, Figure 15 shows the first of the nine scenarios. This particular scenario was designed to probe end users' mental models regarding session management (1a), database lookup (1b), and conditional output (1c). Some of the questions in the nine scenarios are targeted at the same concerns, but approach them from a different perspective (see Appendix E.2). Most of the questions begin with the words: “*What do you think the web site must do to...*”; we hoped that this probe would prompt the webmasters to direct their attention “inside” to the inner workings of the hypothetical application. Participants were asked to provide as many details as they could when answering the questions; as the facilitator I often prompted them for details if it seemed that the scenario had not been completely analyzed. Participants were also encouraged to use sketches to clarify their thoughts. The study took place in a one-on-one setting in a private atmosphere. Verbal responses were voice recorded for later analysis.

After participants finished analyzing the nine scenarios I asked them two general questions which contribute to the needs analysis for an EUDWeb tool:

1. *What web applications do you currently use or would like to use in the future on your website?*
2. *How would you describe your ideal web application development tool?*

The first question was intended to help us expand our understanding of the kinds of web applications we should support with an EUDWeb tool. The second question was intended to gather informal requirements for the design of such a tool. For each of the two questions we conducted an unstructured interview encouraging the participants to elaborate on and clarify their responses.



1) After logging in with your user-ID the web site always shows your full name and a logout button in the upper right corner.

a) What do you think the web site must do to keep track of the fact that you are logged in even though you go from page to page?

b) What do you think the web site must do to show your full name, although you only entered a short user-ID? Take the user-ID “jsmith” as an example and show step-by-step how the web site determines the name “John Smith”.

c) Note that the library home page only displays your name when you are logged in. If you are not logged in, it shows a login box instead. How do you think this feature works behind the scene?

Figure 15: Scenario 1 of 9 as shown to each participant (MMODELS-2)

The study was analyzed in the following manner. First, I transcribed the recorded verbal descriptions for each participant (focusing on analysis questions, and excluding unrelated remarks). If participants had made sketches I used those to understand and annotate their remarks. Second, in a separate document I listed the 12 web development concerns of interest, and inserted pieces of the transcribed interview under the aspects

they referred to. Each remark was coded with a reference to the participant to enable later quantitative analysis. Often, I combined across answers from different scenarios or questions to give us a better understanding about a particular aspect of a webmaster's mental model. Finally, the results for each development concern were summarized by referring back to this document, and when necessary the transcribed interviews or even the original recordings.

Not all users answered all questions. Sometimes a participant responded simply that "*I have no idea*" rather than attempting an explanation. In such cases I encouraged participants to give a "best guess", but occasionally I was forced to continue without an explanation. In general I was sensitive to participants' comfort level, and if a participant conveyed or said that s/he simply did not have an answer or even that s/he was feeling stupid I moved on to another question. Unfortunately, one consequence is that answers regarding some of the more complex and unfamiliar programming goals (e.g., implementing an overview-detail relationship) were quite sparse.

4.2.3 Results

In the following I summarize the findings, clustered by web development concern. Implications of the findings are discussed in the final section of the chapter.

4.2.3.1 Session management

One of the test scenarios asked a question related to session management: "*What do you think the web site must do to keep track of the fact that you are logged in even though you go from page to page?*"

Overall, seven participants indicated that they would assume that the application's state is preserved while a user navigates the website. One participants' statement exemplifies this view: "*...is the status quo, it's like an on/off thing, a toggle type of situation*". Three participants did not understand the question or, even after explanation did not answer it. One participant exhibited a more explicit notion of state maintenance by saying: "*It just keeps verifying at each page again*".

4.2.3.2 Input validation

With respect to the programming concern of input validation we primarily wanted to explore the language and procedures participants would use to specify input constraints, because this is what an EUDWeb tool would need to know in order to construct a validation routine.

None of the participants seemed to have difficulties in specifying rules for valid input (in our case a phone number). All eleven participants used the concept of a pattern-matching process that is related to a particular input field, although different words were often used to describe it, e.g. “symbols”, “placeholders”, “slots”, “pattern”. In the simplest case, the number of digits alone was proposed as a way to validate the input.

4.2.3.3 Conditional output

The following question probed the participants’ intuitive model of conditional output: *“Note that the library home page only displays your name when you are logged in. If you are not logged in, it shows a login box instead. How do you think this feature works behind the scenes?”*

Three participants imagined “some coding within a template page”; two participants imagined two separate pages, and the remaining participants did not answer the question. Many participants informally used the phrasing of “if-then” rules in applying the condition. However, there seemed to be no clear sense about when and how these rules should be applied.

4.2.3.4 Authentication and authorization

Two of the questions relating to authentication and authorization were: *“How do you think the web site checks whether or not your user-ID and password are correct?”* and *“How do you think the web site keeps track of which user is allowed to see which part of the web site?”* We analyzed responses to these probes along two distinct dimensions. First, we wanted to know whether our participants would recruit the concept of *user groups*, or rather would consider the goal to be one of explicit permission values for each individual. Second, we wanted to know whether our participants would allocate to each user exactly one permission or group-related attribute (such as “user class”) or

more than one (such as three fields representing “manager”, “librarian”, “patron”). The latter shows an appreciation of the possibility that the application would serve multiple user groups with different needs.

With respect to the first concern, five participants imagined the concept of user classes, five assigned permissions to individual users and one participant offered both solutions as alternatives.

Regarding the second dimension of analysis, six participants assigned exactly one permission/group-related attribute to each user; only two participants described the possibility of assigning more than one attribute.

4.2.3.5 Database schema

As mentioned previously, ten of the eleven participants had at least some database experience. As a result, most of the database-related questions (e.g., “*In what form and format do you think the web site keeps record of the checked-out videos?*”) turned out to be a test of their knowledge of relational database concepts rather than an indicator for the naturalness of these concepts.

With this caveat in mind, we observed that nine participants appeared to rely on the mental image of a spreadsheet or table when thinking about a database. One participant imagined that one page would store one record and that a large set of pages would constitute “the database”. The participant without prior database experience imagined “pages of code” which somewhat resembled an XML-data store (e.g. <firstname>John</firstname>), although she did not explicitly mention XML. This mental image may have resulted from her prior knowledge of HTML.

4.2.3.6 Database lookup

We wanted to know how our participants visualize the process of looking up and retrieving a particular data record given a key field (for example, how the application finds a name given a user-ID). Eight participants seemed to have only an abstract mental model of this process (e.g. “...searches for your information”), one participant provided a more detailed algorithm of how to select the data, and one participant merely stated: “*I don’t really think about these things*” (even encouragement did not produce additional

insight). This was surprising to us given these users' prior experience with databases; presumably these are webmasters who have retrieved data often from a database, but have never reflected on how the look-up takes place.

4.2.3.7 Overview-detail relationships

One of the scenarios asked the participants to describe what the implementation would look like for a feature that provides a listing of movies, where each movie is linked to a separate web page that displays the movie's details. In particular, we wanted to know how the participants imagined the link between "overview" and "details" pages, and the information carried by this link.

Four participants answered that the link would carry the movie's ID information (which might be the best possible implementation). Three participants imagined a more naïve model in which the link carries the movie's title (a problematic implementation if two movies have the same title). The remaining four participants struggled to find an answer (or even understand the question)—two of them imagined fixed web pages that would be linked on the basis of pre-assigned file names.

4.2.3.8 Normalization and use of foreign keys

We wanted to know if and how our participants deal with the problem of data redundancy. Since most non-trivial web applications need to store multi-dimensional data (e.g., movie information *and* patron information) end-user developers are sooner or later confronted with the problem of separating data into more than one table or dealing with the problems resulting from data redundancy. Thus one scenario asked them to describe how they thought movie-checkout information is stored within the database.

Four participants described a model that would store patron attributes (first name, last name, phone, checkout date etc.) directly in each movie record (either disregarding or implicitly accepting the problem of data redundancy, their comments did not clearly distinguish between these alternatives). Two participants imagined that the patron information would be stored in a separate table and linked via a user-ID (the classical "normalized" solution). Two participants mentioned both models as alternative implementations. Three participants did not answer this question.

4.2.3.9 Uniqueness of data records

When asked specifically how they thought the application handles the problem of having two copies of the same movie, nine participants proposed the existence of a unique identifier (call number, index number + subscript for each copy etc.). Only two participants did not give a clear answer.

4.2.3.10 Calculating database statistics

We did not ask many questions that would involve calculations, but we did include one specifically designed to probe this aspect of programming—we asked how the web application provides a sum of all checked-out movies, or more generally, how any calculation of database statistics would be implemented. Seven participants imagined that the web application would simply count the respective records on request. Three participants imagined a self-updating row counter similar to the automatic recalculation and sum features of spreadsheet applications. One participant simply stated: “*no idea*” (again I tried to no avail to receive a more satisfying answer).

4.2.3.11 Search

As with answers to questions about database lookup, all participants used a relatively high-level description to explain how a two-parameter search might be implemented (we only tested the Boolean conjunction, i.e. the logical AND). Five participants spontaneously used Boolean logic (although not in a formal way) to specify the query (e.g. the keyword contains the word “wind” *and* the movie release date is greater than 1998). Four participants imagined that two queries would be performed consecutively in order to handle the two parameters. Two participants were not able to answer the question beyond giving a high-level analogy, for example “...*like ‘Find’ in Word*”. In most cases, participants avoided, failed at, or gave up when trying to describe the details of how the search might be implemented (i.e., specifying a pattern matching process).

4.2.3.12 Timer

The final scenario asked participants to explain the inner workings of a timer that automatically emails patrons when their movies were over-due. Eight participants imagined some form of system clock that every night initiates a search on the movie table. Interestingly, three participants imagined that the timer would somehow be handled *within* the database table (e.g. “*every night the table would refresh itself and put in the calendar day for every row*”, “*the due date column is a function of the check out date...there is another column that is the overdue trigger*”).

4.2.3.13 Use of and need for web applications

After the participants explored all the scenarios, I asked them to describe their current use of and needs for web applications. This question was intended to help us continue to explore and refine the classes of web applications that an EUDWeb tool would need to support for this user population.

One general observation is that few participants distinguished between static websites and interactive web applications. When I enquired about their needs for features, they often asked for simple static features such as a consistent navigation scheme, breadcrumb trail, or drop-down menus. Furthermore, three participants asked for simple search functionality limited to their website as provided by commercial search engines; two mentioned the need for web usage statistics or as they phrased it “hit counters”; two were interested in restricting access to certain pages within their site. These comments indicate that at least some of our participants see web development as a single activity, regardless of whether functionality is implemented on the client or on the server, and with or without the help of external tools (like web log processors or search engines) – this a view that seems logical in hindsight but was actually quite revealing to me as an experienced web developer.

In order of frequency (the number in parenthesis indicates frequency of mention), the following classes of web applications were requested by the webmasters:

- Registration forms (5),
- Surveys (4),
- Databases (4),
- Reports (3),
- Service request forms (2),
- E-commerce applications (2).

Although most envisioned systems fit nicely into one of the categories above, the actual web applications were quite diverse in purpose, ranging from a simple “need more info” email form to a rather elaborate database of stock donors. This underlines the need for tools that allow the creation of *custom* web applications. I do not see such lists as an exhaustive account of all possible end-user-developed web applications but rather an indication for the kind of power required from a EUDWeb tool.

4.2.3.14 *The “dream” EUDWeb tool*

The final post-test interview question encouraged the participants to imagine their ideal EUDWeb tool and in particular how it should operate to serve their needs.

Leading the “wish list” is a set of templates that are provided for the developer (six participants). At the same time, three participants commented that they would also like to develop manually and not be confined by templates. Five participants mentioned that a wizard-approach may be an appropriate tool feature, presumably reasoning from their experience with wizard assistants in spreadsheets and other common applications. Three participants wanted a tool that would assist them in the layout of their site including pre-defined components for header, footer, sidebar, navigation etc. Two participants stated that they would want a direct manipulation user interface. Many of the other participants showed their preference for windows, icons, menus, and pointing device actions (WIMP) like drag-and-drop more implicitly. Two participants wanted the tool to support collaborative development.

Other ideas were unique to individuals but convey something of their attitudes about tools and programming more generally. For example, one participant stated: *“I’m scared of experimenting...I have lost a whole computer before”* (presumably showing a desire for simplicity, stability, and undo functionality). Another remarked: *“I don’t like programs to think for me, I like to make decisions myself”* and added: *“I don’t like just seeing the screen and the program doing all the thinking behind it and me not having any view of what that thinking is. Because I think I can figure out the thinking if you teach me the language”* (presumably showing a desire for detailed control).

4.2.4 Summary and Conclusions

From a methodological point of view I learned a number of lessons about studying webmasters’ (or other end users’) mental models. Extracting the participants’ mental models was difficult and required a very involved interview. Participants frequently expressed that they simply did not know or had never thought about the implementation of a particular aspect. A possible refinement would be an approach that has a more “graduated” set of scenarios and questions. For example, one might start out with a very straightforward question about database structure and follow that up with more explicit probes about how retrieval or filtering might be done.

I noted that in many cases participants had very sparse models of the programming functions we presented. Although a sort of “non-result”, this observation is interesting in itself because it underscores the need for tools that provide transparent support of certain frequently-used functionality (e.g., session management, search). Note that participants often used appropriate language to refer to technical concepts even when they did not understand how they worked (e.g. key fields). Therefore, it seems plausible that casual web developers will be able to understand a toolkit that employs constructs like key fields or foreign-key relationships.

The following section summarizes the general findings obtained through the second mental models study and how these findings have influenced my thoughts about the design of future EUDWeb tools.

Session management. The majority of our participants assumed that session management is implicitly performed, and thus is not something that a developer would have to consciously consider. This suggests that an EUDWeb tool should automatically maintain the state of an application, perhaps even without exposing this fact to the developer. For novice web application developers this concept may introduce unnecessary complexity. In subsequent evaluations of our EUDWeb prototype tool Click we found some incidents where developers expected a reset of the application's state (or part thereof) (see 6.2.5.4). This is consistent with a default belief that the background processes will manage any needed state information in an appropriate fashion. (It is also consistent with the intuitions we observed in our first mental models study).

Input validation. The typical approach of defining an input mask using patterns or placeholders (as used by many existing tools, e.g. Microsoft Access) seems to be an appropriate abstraction for end users. Certainly, this result is not surprising in light of the fact that ten participants had previous database experience and were familiar with this notion.

Conditional output. Although "if-then" phrasing was frequently used, the exact implementation (in particular when and where these if-then rules should be applied) did not appear trivial to most participants. This suggests that while an EUDWeb tool may use the notion of "if-then" at a high level of abstraction, it may need to automatically develop an implementation or guide the developer as to where to place these rules.

Authentication and authorization. Overall, the problems involved in permission management did not appear too taxing for our participants. However, the proposed implementations were rather variable and almost always incomplete, and were not powerful enough for a real-world application. We believe that our participants would not have many difficulties in *understanding* a good permission scheme; however they may not be able to create a sufficiently powerful and secure one on their own. Therefore, an easy-to-use EUDWeb tool should offer permission management as a built-in feature and make it customizable by the developer.

Database schema. Overall, the table paradigm seems to be the prevalent mental model among our participants. This suggests that an EUDWeb tool may safely use the

table metaphor for managing data. However, the management of more than one related data table may not be a trivial problem, as discussed further under the aspect of “Normalization and use of foreign keys.”

Database lookup. Although the concept of database lookup (or select) did not seem difficult to the participants, the majority did not provide a detailed algorithm. This suggests that an EUDWeb tool should offer database lookup as predefined functionality that is customizable by the developer.

Overview-detail relationships. Overall, imagining how the linkage between overview page (list of all movies) and detail page (movie details) is implemented was quite a challenge for our participants. Almost all of the participants immediately stated that the information was “linked”, “associated”, “connected,” or “referenced;” but the details of this linkage were quite unclear. This suggests that although an EUDWeb tool may be able to use words like “linking” to describe a relationship between two views, it will likely need to guide the developer as to what kind of information the link will carry (or abstract this detail completely).

Normalization and use of foreign keys. The results suggested that most of our participants would not design a normalized database representation but rather some redundant form of data storage such as that familiar from spreadsheet applications (which lack the concept of foreign key relationships). Therefore, if non-redundant data storage is required (note though this may not be important for small or ad hoc applications), an EUDWeb tool may have to make the developer aware of data redundancy problems and propose potential solutions and perhaps (semi-) automatically implement these solutions.

Uniqueness of data records. Our participants had no difficulties imagining the utility of a unique record identifier. However, as the results from the “Overview-detail relationships” aspect show, the correct use of this unique identifier was often unclear. Therefore, an EUDWeb tool may either automatically introduce a unique identifier as a data field or guide the developer towards defining one.

Calculating database statistics. Participants were asked to describe how the web application calculates the total number of checked-out movies. Most participants naturally selected the most likely implementation (application counts records on request).

For the others, their prior knowledge of the workings of spreadsheet programs seemed to influence their mental models (self-updating counter). Overall, this question was not perceived as a stumbling block. I suggest that an EUDWeb tool should offer familiar predefined statistics such as column sums, averages etc. to aid the developer.

Search. The concept of searching appears to be well understood at a high-level of abstraction, including the possibility of multiple search parameters. However, the implementation of a search function was beyond the mental models of most of our participants. Therefore, EUD tools should offer a built-in query mechanism that lets developers specify parameters and connecting operators but does not necessarily expose the details of the implementation.

Timer. Overall, our participants did not seem to have major difficulties imagining an implementation for a timer function, as long as the tool provides easy access to an internal clock of some sort.

The answers to our question about the “use of and need for web applications” indicate that our target audience (nonprogrammer webmasters) not only requires help regarding the implementation of database-driven web sites but also help regarding more mundane issues such as consistent navigation, site search or drop-down menus. From a tool that intends to support nonprogrammers in the development of dynamic web sites they expect a rounded feature set that addresses all facets of web development. They want the tool to be accessible by providing predefined templates, and wizards while still leaving the developer in full control of the details. Even if a tool’s sole purpose is to assist end users with the implementation of basic data collection, storage and retrieval applications (my research focus), the tool designers should consider the web development process as a whole and expect their users to look for features that are not directly related to database-driven websites.

4.3 Summary and Conclusions

The two mental models studies have shown that end-user developers frequently only have very sparse mental models of the inner workings of features commonly found in web applications. Although this represents a sort of “non-result”, this observation is interesting in itself because it underscores the need for tools that provide transparent support of certain frequently-used functionality (e.g., session management, search). Generalizing across the pattern of results reported here, I offer the following characterization of a “prototypical” end-user web developer (Table 12).

Table 12: The Mental Model of the “Prototypical” Novice Web Application Developer

The “Prototypical” End-User Web Developer...
<ul style="list-style-type: none">• Often uses technical terminology (e.g., fields, database) but without being specific and precise• Is capable of describing an application’s visible and tangible behavior to a nearly complete level (only if under-specification is pointed out to them)• Naturally uses a mix of declarative language (e.g., constraints, if-then rules) and procedural language (e.g., a few explicitly sequential steps) to describe behavior, while being unclear about where and when these constraints/rules/steps should be applied (lack of control flow)• Does not care about, and often is unable to describe exactly how functionality is implemented “behind the scenes” (e.g., search, overview-detail relationships)• Disregards intangible aspects of implementation technologies (e.g. session management, parameter passing, security issues) and only considers surface features (e.g., invisible link → page protected)• Understands the utility of advanced concepts (e.g., unique key fields, normalization) but is unlikely to implement them correctly without guidance• Imagines a spreadsheet table when reflecting on data storage and retrieval

I advocate that EUDWeb tools should expose their functionality in a way that is close to their users’ natural mental model. A tool is likely to be easy to use if it works according to the expectations of its users. For example, high-level components should be available for implementing frequently needed functionality such as searching or

generating lists of data, purely technical concepts such as session management should be abstracted, and difficult technical problems such as cross-platform compatibility and security hidden as much as possible (e.g. by automatically generating cross-browser compatible code and automatically performing security checks).

However, the mental models studies I conducted can only determine what end users “naturally” think. In order to determine whether or not certain design solutions are easy to understand and easy to use we need to create and evaluate prototype tools – the focus of the work described in the next chapter.



Chapter 5

Click – A Web Application Development Tool for End Users

Click is a prototype of a web application development tool targeted at end-user developers. This prototype embodies much of the findings from my studies of nonprogrammers (see Chapters 2, 3, 4) and has been developed as a proof-of-concept and to evaluate certain techniques (e.g., abstraction, integration) that may facilitate EUDWeb. Much of the implementation effort took place in close collaboration with Yogita Bhardwaj and Jonathan Howarth who worked with me as research assistants throughout significant parts of this work.

In the following section I will first introduce the paradigm of “Design-at-Runtime” – a basic concept I have developed and evaluated as part of the Click prototyping effort. Next, I will briefly describe the history of Click by discussing early prototyping efforts and lessons learned. The balance of the discussion will be dedicated to Click, its features, rationale, and architecture. An interactive demo of Click is available at <http://phpclick.sourceforge.net/>.

5.1 Design-at-Runtime

As a result of our surveys and interviews of web developers (see Chapter 3) it became apparent that one important requirement for a good development tool is its speed and in particular its capabilities for quickly iterating through the develop-test-cycle. For this reason – and although in general my research questions take a holistic perspective to

EUDWeb (see 1.4) – I have chosen to focus on the problem of facilitating the develop-test-cycle. The following articulates the basic idea and rationale for “Design-at-Runtime” – a concept that is referenced repeatedly in the discussion of Click’s design (see 5.3-6.2).

The active programming strategies observed in professional programmers (see 2.2) provide a scientific grounding for my work on nonprogrammer tools – indeed I expect end users to be even more active and result-oriented than experienced software developers. For example, a nonprogrammer is less likely than a programmer to worry about designing an elegant system architecture. Many studies of computer use have demonstrated that for the most part end users do not want to “learn” but rather to “produce”, and will use whatever information or resources is available to help them make sense of a task just enough to make progress (Carroll 1990).

Given this view of programmers as active users, I propose an alternative to the mode-based programming paradigm of typical visual web development tools (e.g., Macromedia Dreamweaver or Visual Studio), in which developers need to explicitly switch between development and runtime mode. I call the paradigm “*Design-at-Runtime*”. As an application of Tanimoto’s (1990) general concept of “*liveness*” (see 2.3.2) to the domain of web engineering, the design-at-runtime concept builds from the ideas of direct manipulation (Shneiderman 1983) and the “debugging into existence” behavior (Rosson and Carroll 1996) documented for professional programmers. In its core it is similar to the automatic recalculation aspect in spreadsheet programs. A critical piece of the concept is that the user is able to both develop and use the application without switching back and forth between design and runtime modes. That is, the application is always usable to the fullest extent that it has been programmed. The end-user developer alternates between constructing and “using” the application until he or she tries to use an object with a not-yet-defined behavior. At this point the system prompts the user with a dialog that can be used to define the missing behavior. This interleaving of development and use continues until the entire application has been defined and tested.

The applicability of design-at-runtime reaches beyond just web development – the paradigm could be used by programmers and nonprogrammers alike, in many domains. However, my focus is on EUDWeb and my discussion addresses this particular

application of the general concept. Of course, the usefulness of working with live data instead of placeholders at design-time has been realized before. In Macromedia Dreamweaver, developers can switch to the so-called “Live Data View”. In this mode live web pages are shown and some adjustments can be made. However, Dreamweaver does not allow developers to actually use their developing applications – for example, hyperlinks do not work in this mode. Therefore, the developer still must repeatedly switch between different interaction modes.

Although I have not conducted any formal experiments to compare design-at-runtime against classical mode-based programming I argue that it has a number of advantages. The paradigm embraces the naturally occurring tendency for “debugging into existence”. The programming environment gives immediate feedback to any actions and changes by the developer. Design-at-runtime delivers true What-You-See-Is-What-You-Get (WYSIWIG), because the developer always works with a running application operating on live data. Finally, the application under development is implicitly subject to continuous testing. This may help to improve the reliability of the resulting application.

However, there are still a few unresolved issues regarding the realization of the design-at-runtime paradigm. For example, developers need a means to distinguish whether they intend to interact with or edit an already-defined button action – therefore, at least a minimal notion of a mode (execute vs. edit) is still needed. Click addresses this problem by providing small handles that are displayed next to each component that when clicked invoke the “edit” operation. Another example challenge for design-at-runtime is a component that outputs a value that momentarily is empty. It may be tedious for the developer to determine the role of the output component – some concept of role-expressive placeholders or handles for empty values may still be needed.

The summative evaluation of Click shows that end-user developers quickly embrace and highly appreciate the advantages of design-at-runtime (see 6.2.8).

5.2 Early Prototyping Efforts and Lessons Learned

5.2.1 FlashLight

As a first attempt and proof-of-concept for a web development tool that employs the design-at-runtime paradigm (see 5.1), and that may be suitable for nonprogrammers, I created a system called “FlashLight” (Rode and Rosson 2003). Figure 16 shows a screenshot.

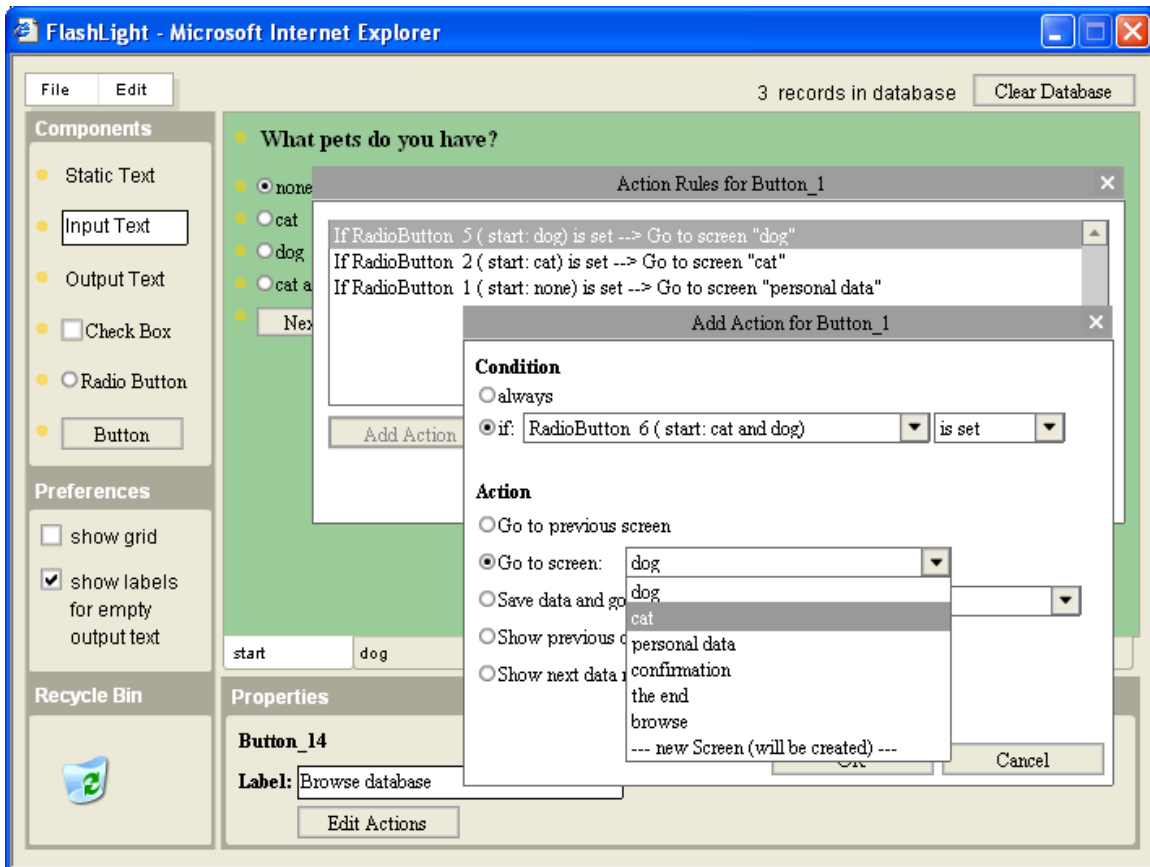


Figure 16: Defining button actions in FlashLight

FlashLight is a prototype tool using a combination of Flash MX (Macromedia 2002b), the server-side programming language PHP (Lerdorf, Gutmans et al. 1995), and XML. This early prototype implements a subset of the concepts and components that may appear in a typical web application (see 3.3) namely data input and storage, session

management, branching login, and basic data output. The tool works at a high level of abstraction in order to hide the complexities of web application development and allows users to create multi-screen web applications for data collection.

5.2.1.1 Programming in FlashLight

In FlashLight, developers design a series of screens by dragging components like checkboxes, radio-buttons, or text input fields onto the workspace. Screens can be selected via tabs (see Figure 16). Developers can edit the component properties at any time by clicking on a small yellow dot that visualizes a “handle” for the component.

The components are fully functional as soon as they are dragged onto the workspace – text input fields allow user input, buttons can be clicked and so-called “output text components” display live data. Application-specific functionality is programmed by dragging buttons onto a screen and clicking them. If a button already has an action (e.g. go to screen xyz, save data record) associated with it, the action is immediately performed; otherwise a dialog with the user is initiated to specify an action.

The dynamic behavior of the application is controlled by defining “action rules” which are pairs of conditions and associated actions. Figure 16 shows a screenshot of FlashLight that depicts the process of defining an action rule. The screenshot shows that three rules have already been associated with the button to define what it should do depending on the user’s inputs. In a similar manner the user can define the behavior of the “output text” component. So-called output rules determine what kind of output is shown under certain conditions.

FlashLight also simplifies application deployment. Once a developer saves an application (File/Save), the properties inspector of each screen displays a web-address that can be pasted into a browser to access the working application. Every screen is given a unique web address. Thus an application may offer different functionalities to different users (e.g. data input for general public, data browsing for authorized users only).

Many of FlashLight’s concepts (e.g., drag-and-drop, concept of handles, action rules, ease of publishing) were used in Click.

5.2.1.2 Database Model

FlashLight’s underlying database model is very simple: the development of a powerful and scalable database layer was not a priority for this prototype, because we were more interested in the overall programming paradigm and user experience. In FlashLight, the database model is represented by a set of data records, each containing the values that correspond to the user inputs from checkboxes, radio-buttons, and input text components during one user session. The data entered by the user is automatically kept persistent throughout the application allowing the user to jump back and forth between different screens.

On one hand, the implementation of the database model simplifies development by hiding the database layer from the developer. On the other hand, it has severe limitations. In FlashLight, there is always a one-to-one mapping between an input component and a database field. It is currently not possible to have two input components correspond to the same value in the database. This would be needed to implement “add record” together with “edit record” functionality in a web application – in fact a rather basic requirement that was addressed in our second phase of prototyping. A fully functional EUD tool would need to address the problem by decoupling input components from database fields. Furthermore, FlashLight only handles databases with exactly one type of record (or table), although web applications often contain more than one type (e.g. a library application would contain a data table for books, one for patrons etc.). Finding a good way to represent the entity-relationship-model (multiple database tables and relationships via keys) to a nonprogrammer is a challenge for further research. Following a commonly-used approach (e.g., Turau 2002; Zdun 2002; Laszlo Systems Inc. 2005; Macromedia 2005c), FlashLight stores metadata describing the application in a custom XML format. User data is stored on the server in a similar fashion.

5.2.1.3 Platform and Implementation

I developed FlashLight using Flash MX and the integrated programming language ActionScript. FlashLight implements a small subset of what is possible with Flash—hence its name. FlashLight components running on a server use PHP to save application

metadata and data into XML files for persistent storage. I chose Flash mainly because of its flexibility, rapid prototyping support, and web delivery capability. Nevertheless, I do not endorse Flash as an ideal platform for comprehensive web development tools (potential alternatives are a desktop application written in C++, Java, Visual Basic, C# etc. or a DHTML-based web application written in ASP, PHP, ColdFusion or Java). Indeed, my experiences with Flash have been mixed. A frequent complaint—that Flash’s movie metaphor gets in the way of application programming—turned out to have a simple solution. I ignored the movie metaphor and placed all ActionScript code within one movie frame. On the downside, although ActionScript is object-oriented, it seems limited in terms of scalability. I externalized all of the ActionScript code using Flash’s “#include” directive; nonetheless, I found it difficult to enforce a maintainable code structure and avoid unwanted side-effects.

Since my development of FlashLight in 2002, Macromedia has released Flex (see 2.1.3.5), a Flash-based web programming language targeted at application development rather than simulation and animation that addresses many of Flash’s shortcomings. Because Flex has only become available recently, I could not use it for prototyping although it is a promising technology.

5.2.2 Custom Extensions to Existing Tools

After the attempt to implement a full-featured web development tool in Flash had proven too difficult (see previous section), I explored the option of creating a custom extension for Macromedia Dreamweaver (Macromedia 2005b). Dreamweaver exposes an Application Programming Interface (API) that allows developers to extend Dreamweaver’s built-in functionality with custom features. Using only standard HTML and semi-standard JavaScript (a proprietary library is required), Dreamweaver’s user interface can be modified. I had envisioned that by developing a Dreamweaver extension end users would be able to reuse the WYSIWYG web editing functionality already offered by Dreamweaver. However, although adding simple code-generation and code-replacement to Dreamweaver functionality proved to be fairly straightforward, I soon encountered a major drawback to this approach: Dreamweaver’s API is powerful, but still

quite limited. Dreamweaver has many predefined UI concepts which cannot be modified programmatically. Finally, Dreamweaver's API proved to be too difficult and inflexible to implement the integrated and seamless workflow I had envisioned.

After abandoning the idea of a custom Dreamweaver extension I explored the possibility of extending Eclipse (2005). Eclipse is an open-source universal tool platform and extensible IDE. After a brief investigation of Eclipse's features and existing libraries to support WYSIWYG web development, I concluded that too much low-level programming would be required to implement the functionality needed by even a basic EUDWeb tool. Note that since the review in 2003, Eclipse's libraries have improved considerably. IBM has even chosen Eclipse as the underlying platform for their web development IDE: Rational Web Developer for WebSphere Software (IBM 2005a).

5.2.3 Click Prototype #1 and #2

The attempts to implement a EUDWeb tool as an extension to an existing tool were abandoned because of inflexibility. In order to better control the user experience and workflow of the tool, I decided to create a separate tool.

The first prototype of Click was a PHP/MySQL-based web application that could be used to setup and manage the database and define the *behavior* of a web application. The screen *layout* still needed to be designed outside of Click using an external WYSIWYG web editor, such as Dreamweaver. The advantages of this approach were twofold. First, the developers could continue to use the layout tools they were familiar with; and second, Click did not have to include its own WYSIWYG web editor, which significantly reduced the scope of the prototyping effort. In order to develop an application using this prototype, the end-user developer would open and alternate between his or her favorite web editor and Click. The WYSIWYG editor was set up to save files to a network drive that could be accessed by Click. Once a web page had been saved, Click noticed the change and allowed the developer to define the behavior for particular elements such as text fields or buttons (small icons were displayed inline as handles). After the developer had changed certain properties, Click would rewrite the page's code, embed appropriate PHP code that would implement the functionality, and

finally save the file. The WYSIWYG web editor would automatically reload the externally modified file and the developer could make further changes to the layout. This alternation between modes of development and tools is shown in Figure 17.

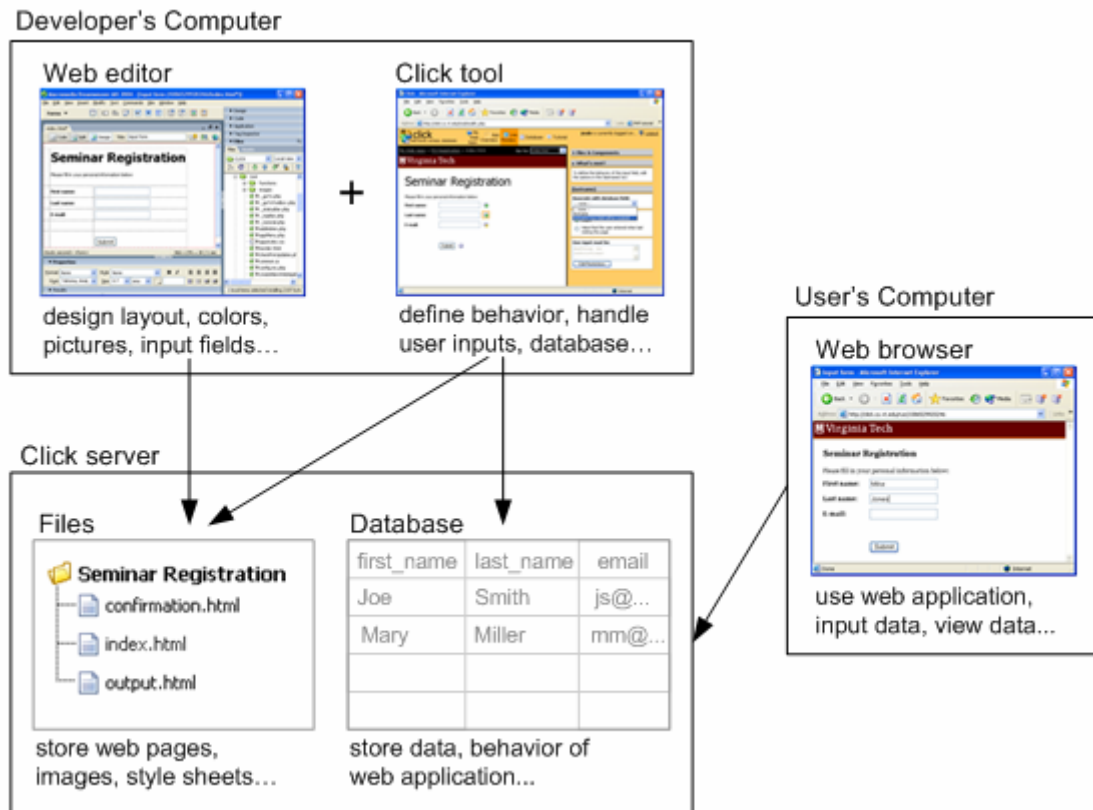


Figure 17: Click prototype #1: an external WYSIWYG editor is used in conjunction with Click

The main shortcoming of the first Click prototype was the need for an external WYSIWYG editor. This proved to be a major impediment to usability. In a series of formative usability studies, many developers were demonstrably confused about when to use one tool and when the other. Furthermore, the user interfaces of the two tools were not consistent, causing more confusion.

My vision of a stand-alone EUDWeb tool became more realistic with the availability of a JavaScript drag-and-drop library (Zorn 2004). Using this library we implemented Click's second prototype, a web-based WYSIWYG web editor that allowed developers to specify, both, the behavior *and* the layout all within Click. This prototype

no longer used code-replacement strategies to update a page's HTML/PHP code but rather stored the application's definition in an XML format that at runtime was interpreted by Click. Storing the layout and behavior definition in XML, similar to what is done in OpenLaszlo (Laszlo Systems Inc. 2005) and Flex (Macromedia 2005c), was more straightforward than writing and rewriting multiple pages that contained low-level HTML and PHP code (as used in prototype #1). After several rounds of refinement through formative usability studies, this second prototype of Click appeared to be much closer to the goal for an integrated EUDWeb tool. The drawback to relying on a predefined set of components (in this case they were mapped to XML) was once again lack of flexibility and scalability. End-user developers could implement basic form-based web applications as long as Click's pre-defined components offered the needed functionality, but custom extensions were not possible. The third and final prototype of Click addressed this flexibility problem by switching the predefined monolithic components for a flexible, extensible, and layered component framework as described in Section 5.4.25. For example, when the final Click tool was used to implement a production conference paper review system, certain features that went beyond the pre-defined functionality (e.g. display only papers belonging to the currently logged-in reviewer) could be addressed with custom code – something that would not have been possible in the previous prototypes.

5.3 Click’s Development Paradigm and Key Features

We are developing Click (Rode, Bhardwaj et al. 2005) as an EUDWeb prototype that is specifically targeted at end users who want to develop web-based data collection, storage and retrieval applications. A canonical example would be an online seminar registration application. Before I discuss Click’s features and their rationale in detail, I will briefly illustrate how an end-user developer might use it to create a web application. To construct an application, a developer starts with a blank page or a predefined application template (e.g., service request form, online registration, staff database). The construction process is not predetermined; the developer can begin either by placing components on the screen (using drag-and-drop) or by defining a database structure. Figure 18 shows Click being used to define a button that will save user-entered data into a database and display another web page (see Appendix F.1 for color screenshots).

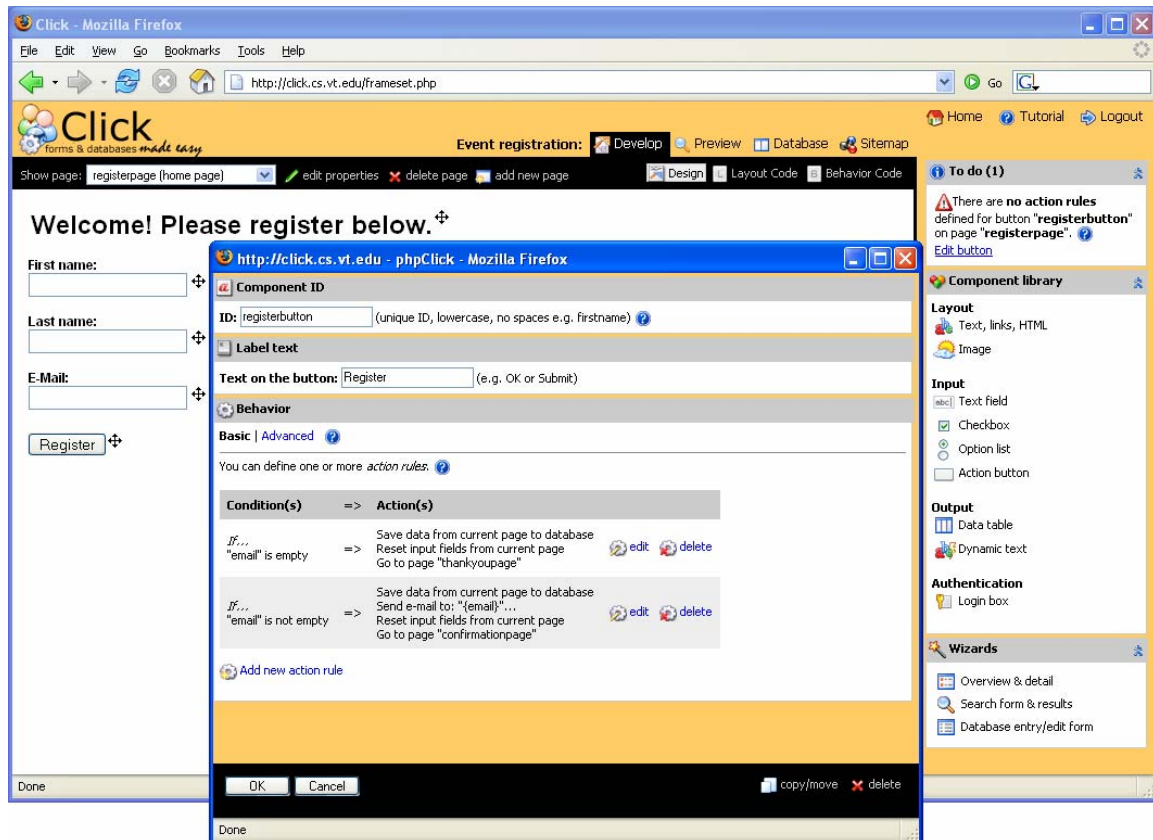


Figure 18: Defining a “Register” button and associated action using the form-based UI of Click

Click applications are developed iteratively, with user input mechanisms added and their behavior specified as the developer needs them. Deployment is as easy as “declaring” a web application as public (in response, Click generates a URL that can be used to access the working application).

Click is an integrated web-based environment that contains visual development tools, code editing features, a preview mode, and a database management interface. No installation or configuration is required by the end-user developer. When the developer instantiates and positions components for a page under construction, Click generates corresponding HTML and component template code (see 5.4.19, Table 15 and Table 16).

In order to convey an overview of Click’s concepts and simultaneously illustrate the level of introductory help provided, Table 13 shows the entire beginner’s tutorial available for developers within Click.

Table 13: Click - Beginner's tutorial

Tutorial








Watch the intro [Video](#) (14 minutes, WMV format with sound, 19MB)

What is Click?

Click is a tool for creating registration forms, e-mail forms, surveys, and database-driven web applications such as online staff databases or service request database. Click **does not require programming** knowledge of any kind but yet allows advanced developers to make custom modifications by writing HTML/PHP/PRADO code.





Click has a built-in What-You-See-Is-What-You-Get **web editor** you can use to design the look & feel as well as the behavior of your web application. It also includes a **database engine and management tool** that allows you to collect, update, and report data. Finally, Click is not only a development tool but also a production hosting environment including a web server and database server. Therefore, once you have finished developing your application you simply declare it as "published".

Click's core concepts

As a developer Click allows you to create  **web applications**. A web application consists of one or more  **pages** as well as a database that can contain one or more  **database tables**. Each page can contain  **components** as basic as  *Text*, *links*, *HTML*,  *Text fields*,  *Checkboxes* or, more advanced,  *Login boxes*,  *Dynamic text* etc.






Click's views

Click offers four main **views** which are displayed as tabs across the top of the screen. The four main views are:

1.  **Develop**: You will spend most of your time in this view as it allows you to create pages, place components on pages and define their behavior. The *Develop* view has three sub-views which are:
 - **Design**: Allows you to position components, view/change the properties of components, and test the web application
 - **Layout Code**: (for advanced developers) Shows the HTML/PRADO template code that implements the layout of your web application
 - **Behavior Code**: (for advanced developers) Shows the PHP/PRADO code that implements the behavior of your web application
2.  **Preview**: Although the *Develop* view already shows you how the web application will look at runtime, this view will give a more accurate picture.
3.  **Database**: This view allows you to create database tables, change their structure and add, edit, and export data during the time of application development.
4.  **Sitemap**: The Sitemap view gives you a graphical overview of the application. It can help you to understand dependencies and uncover missing functionality.

Click's toolbox


Click offers a number of tools that facilitate application development:






-  **Application templates**: When you create a new web application you can choose to start with one of the pre-defined templates.
-  **Wizards**: Wizards help you to create a set of related components to implement commonly needed functionality such as a search form&results pages, or a database entry/edit form. Anything wizards can do for you, you can also do yourself manually. They just help to save work.
-  **"To-do list"**: In upper right corner of the *Develop* view's toolbox you will find a message box informing you about potentially missing functionality, and certain dependencies within the web application you are constructing.
-  **Context-sensitive help**: The help feature is available throughout Click and provides explanations about certain options and concepts. The availability of help information is indicated by the **clickable help icon** .

Click's development process



In the process of creating a web application you will define the **layout** and **behavior** of the application and the underlying **database structure** (unless you do not need database storage because you are implementing an e-mail-only form). There are different ways you can go about constructing a web application. **Whatever you do in Click can always be changed again at a later point in time.** Therefore, feel free to play with the tool.


Getting started: Defining the database structure vs. creating pages and components

If you already know what kind of data you will be collecting you might want to start in the  *Database* view by creating fields within a database table. For example, if you are creating a staff database you may want to create a database table called "staff" containing the fields firstname, lastname, email, and phone. Working with Click's built-in database is not much different from working with a spreadsheet application like Microsoft's Excel. In fact, individual database tables can be compared to sheets within Excel. **Database fields** can be compared to columns on a sheet, **Data records** can be compared to rows.

However, if you are just playing with ideas you may also start in the  *Develop* view by creating pages and laying out components such as  *Text fields*,  *Checkboxes* etc. You can always quickly switch between the  *Develop* view and  *Database* view and make changes as the become necessary.



Laying out components

To place a new component on a page simply **click once** on one of the items in the  **Component library** which is part of the toolbox in the  *Develop* view. A dialog window will open and allow you to define the component's properties. Once you close the properties dialog by pressing the OK button, a new component will be dropped in the middle of the page and you can now **move it around using drag-and-drop**. To change the properties of a component at a later point in time simply **click once** on the component and the properties dialog will open again.


Each component is uniquely identified by the **component ID** - a short name consisting only of lowercase letters without spaces or special characters. This ID may be repeatedly used throughout the application to refer to this particular component. Although it is possible to change the component ID at any point in time, we recommend that you **assign component ID's that are meaningful to you**. For example, instead of accepting the default ID of "inputtext1" you may want to assign the ID "firstname". Assigning meaningful ID's is especially helpful for components that collect input like the  *Text field* because these are often referred to in other places of the application.

By default, components are only shown on the page that you position them on. However, you **can choose to have a component displayed on all pages**. This allows you to implement for example a common **header and footer** without having to re-define the same components on every page.



Defining the page navigation



Any page can be designated as the **home page** of your web application (use  *edit settings for page...*). This is the first page users will see when coming to your application. In order to establish links that allow your users to navigate between pages you can place one or more  *Text, links, HTML* components onto your page and within these components define  hyperlinks that point to other pages. Also, certain components already integrate navigation to other pages such as the *edit* link of the *Data table* component.

Using buttons for initiating actions

Much of an application's **interactivity is initiated by**  **Action button components**. For each button you can define so-called **action rules**. Action rules are **pairs of conditions and actions** that are executed when a button is pressed. You can have one or more action rules associated with a button. The actions of an action rule (for example: *Go to page...*, *Send email...*, *Save to database...*) are only executed if the condition is true. If the condition is false, nothing happens. A condition compares the values that a user entered into certain input fields with known values.

Click's security features

You can define any page as login-protected by using  *edit settings for page...* If a page requires login users will need to provide a user-id and password which they enter into a  *Login box* component (you will need to place such a component on one of the publicly-accessible pages). Click offers two levels of security. First, you can require login and allow all users access who pass the login process. Second, you can require login but only allow those users access who are **administrators** (the field "admin" in the database table "users" determines who is considered an administrator).

Click offers two methods for authenticating a user's user-id and password. Read more about this by clicking on the help icon in the *Authentication* dialog located under  *Application settings* in the toolbox of the  *Develop* view.

Click's publishing and production hosting features

Once you are finished constructing your web application and want to make it available for production use, simply return to Click's main menu by clicking on the 🏠 *Home* link (in the upper right of the screen), and then click on the application's 📄 *publish* link. Click will immediately display a web address similar to this one: <http://click.cs.vt.edu/apps/1109889745071/>. Now all you need to do is either send this link by e-mail to the people you want to use your web application, or create a link on another website pointing to this web address.

You can find more information and give us your feedback on Click at <http://phpclick.sourceforge.net/>.

5.4 Design Rationale

Click has been developed as a proof-of-concept tool to explore one approach to overcoming the barriers to EUDWeb. Below, I will discuss how Click’s design has been driven by the problem analysis (Chapters 2, 3, 4) by showing how specific problems identified in our earlier work have been used to motivate particular design decisions. I begin with an overview of problems we addressed in the design (see Table 14); thereafter each issue is examined in more detail.

The problems and observations (shown in the left table column) represent the current state of knowledge (Chapter 2), the results from survey and interviews studies (Chapter 3), my findings from the mental models studies reported in Chapter 4, as well as findings from the three formative evaluation studies of Click (6.1). The problems and observations are grouped into problem areas. The ones marked in bold font point forward to the summative evaluation of Click (see 6.2) in that we identified them as particularly important or interesting with respect to success in EUDWeb.

Table 14: Mapping from problems to design solutions (the numbers in parenthesis represent the sections discussing the issue in detail; issues marked in bold are the focus of the summative evaluation)

Problems and Observations	Click design solutions
Workflow	
<ul style="list-style-type: none"> • End-user developers need integrated tools that take holistic approach (2.4.5) • End-user developers struggle to notice under-specification and find missing/faulty behavior (4, 6.2.9) • End-user developers do not know how to get started (2.4.5, 6.1) • End-user developers prefer to work in a iterative and opportunistic fashion (2.2, 6.1, 	<ul style="list-style-type: none"> • Introduction video & tutorial (5.4.1) • Application templates (5.4.2) • Support for opportunistic development & Design-at-runtime (5.4.3) • Support for continuous workflow (5.4.4) • To-do list (5.4.5) • Sensible defaults and strong affordances (5.4.6) • Context-sensitive help (5.4.7) • Auto-generated sitemap (5.4.8)

<p>6.2.9)</p> <ul style="list-style-type: none"> • Many tools require premature commitment (2.4.5) 	<ul style="list-style-type: none"> • Integrated development & runtime environment (5.4.27)
<p>Abstraction</p>	
<ul style="list-style-type: none"> • End-user developers cannot implement applications using low-level constructs (6.1) • End-user developers do not understand the stateless nature of the web (4.2.3.1) • End-user developers disregard intangible aspects, e.g. parameter passing (4.1.2, 4.2.4) • Lack of abstraction; early exposure of low-level concepts, e.g. session management (2.4.5) • Integration of diverse technologies is difficult and error-prone, i.e. HTML, JavaScript, CSS server-side code etc. (3.1.2.3) 	<ul style="list-style-type: none"> • Domain specificity, i.e. database-centric apps (5.4.9) • Session layer (5.4.10) • Database layer (5.4.11) • Input validation layer (5.4.13) • Authentication & authorization layer (5.4.14, 5.4.15) • Parameter passing & “Current data record” (5.4.21) • High-level components (5.4.16) • Button action rules (5.4.17) • Templating (5.4.20) • Event-based web programming (5.4.18) • Application templates (5.4.2) • Wizards (5.4.22)
<p>Layout specification</p>	
<ul style="list-style-type: none"> • HTML layout is non-trivial and time-consuming, i.e. requiring nested tables or complicated CSS (6.1) 	<ul style="list-style-type: none"> • Pixel-based positioning (5.4.23) • “Global” components (5.4.24)
<p>Behavior specification</p>	
<ul style="list-style-type: none"> • End-user developers naturally specify input constraints declaratively (4.2.3.2) • Lack of explicit control flow (4.2.3.3) 	<ul style="list-style-type: none"> • High-level components (5.4.16) • Input validation layer (5.4.13) • Event-based web programming (5.4.18) • Separation of layout and behavior (5.4.19) • Button action rules (5.4.17) • Templating (5.4.20)

Database interaction	
<ul style="list-style-type: none"> • End-user developers understand but cannot correctly implement crucial database concepts, e.g. unique key fields (4.2.3.9) • Connecting to databases is non-trivial (3.1.1.2) • End-user developers imagine database as spreadsheet (4.2.4) 	<ul style="list-style-type: none"> • Database layer (5.4.11) • Integrated development & runtime environment (5.4.27) • Authorization layer (5.4.15)
Testing and Debugging	
<ul style="list-style-type: none"> • End-user developers struggle to notice under-specification and find missing/faulty behavior (4, 6.2.9) • Discovering cause for faulty behavior (3.1.2.4) • Bad error messages, i.e. too technical, not clearly related to error condition (3.1.1.1, 2.4.5) 	<ul style="list-style-type: none"> • To-do list (5.4.5) • Auto-generated sitemap (5.4.8) • Support for opportunistic development & Design-at-runtime (5.4.3)
Security	
<ul style="list-style-type: none"> • End-user developers think about security just in terms of surface features, e.g. hidden “edit” link (4.1.2, 6.1) • Even experienced developers are unsure about the security of their applications (3.1.2.1) • Web apps in general have many vulnerabilities and are exposed to high risks 	<ul style="list-style-type: none"> • Security layer (Input validation, Authentication, Authorization layer) (5.4.13, 5.4.14, 5.4.15) • Integrated development & runtime environment (5.4.27)

Compatibility	
<ul style="list-style-type: none"> • Many cross-platform differences, particularly regarding CSS and JavaScript (3.1.2.2) • Good cross-platform testing difficult for end-user developers • Authoring and debugging JavaScript is difficult; few developers use it fully (3.1.1.2) 	<ul style="list-style-type: none"> • High-level components (5.4.16)
Scaling up	
<ul style="list-style-type: none"> • The central problem of EUD is the tradeoff between ease-of-use and expressiveness (2.3.1) • Ideal is a “gentle slope of complexity” (2.3.1) 	<ul style="list-style-type: none"> • Layers of programming support & Gentle slope of complexity (5.4.25)
Collaboration	
<ul style="list-style-type: none"> • End-user development is a collaborative process (2.1.1, 3.4) 	<ul style="list-style-type: none"> • Collaboration support (5.4.26)
Configuration & Deployment	
<ul style="list-style-type: none"> • End-user developers lack knowledge of server setup and configuration 	<ul style="list-style-type: none"> • Integrated development & runtime environment (5.4.27)

The following sections discuss the design rationale for each feature in detail.

5.4.1 Introduction Video & Tutorial

As a tool that integrates most aspects of web development, Click is different from state-of-the-art web development tools such as Macromedia Dreamweaver or Microsoft FrontPage. Usability testing (see 6.1) has shown that novice Click users often do not expect the level of support offered by the tool and therefore start their development by

using suboptimal but familiar strategies such as hand-editing HTML code. To help developers get started quickly, Click contains a 10 minute introduction video and also a written tutorial. The video and tutorial give a brief overview of the general development process and introduce Click’s main features.

5.4.2 Application Templates

As discussed in Section 3.1.1.1, web developers’ needs regarding particular web application features are often very similar and basic. For example, simple online forms or databases are often needed to help collect or report data. Click provides a set of commonly-used applications as a starting point for new development. Developers then have the opportunity to modify or expand an application or just deconstruct and investigate the example to discover how particular functionality was implemented. The pre-defined applications currently include the following:

- Event registration database
- Service request database
- Staff database
- Ride board application
- Multi-page survey

These application templates are solely built using Click’s pre-defined high-level components and do not use any custom code. However, future templates may contain more advanced features that are implemented using custom behavior code (see 5.4.19).

5.4.3 Support for Opportunistic Development and Design-at-Runtime

Supporting iterative and opportunistic development is a key design requirement for Click. To support the general tendency of web developers to work in a personalized and opportunistic fashion (see 2.2, 3.1.1.2, 6.1), Click does not enforce a pre-determined workflow. The developer can either begin by defining the page layout or by creating the database schema or switch between these two approaches at any point in time. Contrary to common code-generation approaches that make late changes to the user interface or

behavior expensive to implement (see 2.1.2), Click allows modifications to the layout, behavior, and database schema at any point in time. Virtually no functionality requires decisions or *premature commitments* (Green and Petre 1996) that cannot be modified easily at a later point in time.

Moreover, changes take effect immediately, thereby facilitating a rapid build-test cycle. Click implements the “design-at-runtime” concept as previously discussed in Section 5.1, allowing the developers to design and run (or test) the application without switching back and forth between design and runtime modes. However, Click additionally provides an explicit preview mode; this requirement was discovered through formative usability evaluation which is discussed later (see 6.1).

5.4.4 Support for Continuous Workflow

In many situations a particular task or workflow requires a sub-task to be finished before the main workflow can be completed. For example, defining a “go to page” action may require the sub-task of creating the target page if it does not already exist. Whenever these kinds of dependencies occur, Click offers the developer a means to accomplish the sub-task without interrupting the main task. In the aforementioned example, Click may offer to create a page named “untitled1” as a choice for the target page.

This feature was refined in response to our observations during the formative evaluation (6.1) of Click. In early prototypes participants often had to interrupt a workflow (e.g., defining a “save to database” action) in order to set up the necessary preconditions (e.g., creating a database field), which sometimes even led them to forget their original intent.


5.4.5 To-Do List

Click does not attempt to predict and interrupt a developer’s workflow in the way an “intelligent” software agent might do, because the risks and costs of false guesses would likely be high (Robertson, Prabhakararao et al. 2004). However, Click does maintain a non-intrusive “To-do” list that keeps track of the developer’s progress and gives recommendations about possible or required future tasks. This feature was designed

in response to our earlier finding that most current tools do not provide sufficient overall guidance for developers (2.4.5.2).

The messages in the to-do list notify the developer about such undesirable or faulty states as for example:

- pages or input components with generic names (e.g., recalling whether “inputtext4” or “inputtext5” was the input field for the user’s first name may be difficult when the developer wants to make references elsewhere),
- a data table component that links to a details page that contains no components to display the details,
- a missing login page in an application that contains pages requiring authentication.

A warning icon  indicates tasks that are necessary for the completion of a valid application.


5.4.6 Sensible Defaults and Strong Affordances

Our formative evaluations of Click (6.1) have shown that *defaults matter*. Not surprisingly, participants performed consistently better when the options they had to configure had sensible defaults. During the design of Click we have paid much attention to the default settings of the user interface in general and property dialogs for components and pages in particular. For each property we have chosen the most likely value as the default in order to minimize the effort required for web developers who want to quickly construct an application. Furthermore, wherever we found that the default value should cover most cases, we reduced the prominence of the particular option. For example, in the property dialogs for components we show important or likely-to-be-modified properties as “expanded” while others are displayed as “collapsed” to reduce visual clutter.

Click’s user interface provides strong affordances (Norman 1988) to indicate important or required settings. For example, the property dialog for a text field

component hides the *width* property which defaults to 20 characters but prominently features the *data connection* property that determines where user input will be saved to.

5.4.7 Context-sensitive Help

The help icon  indicates the availability of a detailed explanation for a particular option or concept. Because many of Click’s programming concepts and procedures are non-trivial and different from state-of-the-art approaches (2.4.5; though perhaps superior), explanation is required. For example, in the dialog that allows the developer to require login for a particular page, a reference is made to the two basic authentication methods (central user database or application-specific user database).

5.4.8 Sitemap

During the mental models studies (4.3), I observed that participants often under-specified the behavior of the feature in question. In an attempt to provide a visual overview of the entire application that would reveal under-specification (such as unconnected pages) and faulty behavior, we designed the sitemap feature.

When the developer selects the *Sitemap* tab, Click generates a graphical representation of the application as it has been defined so far. Figure 19 shows an example of a sitemap for a “ride board” application (see also Appendix F.1 for screenshots in color).

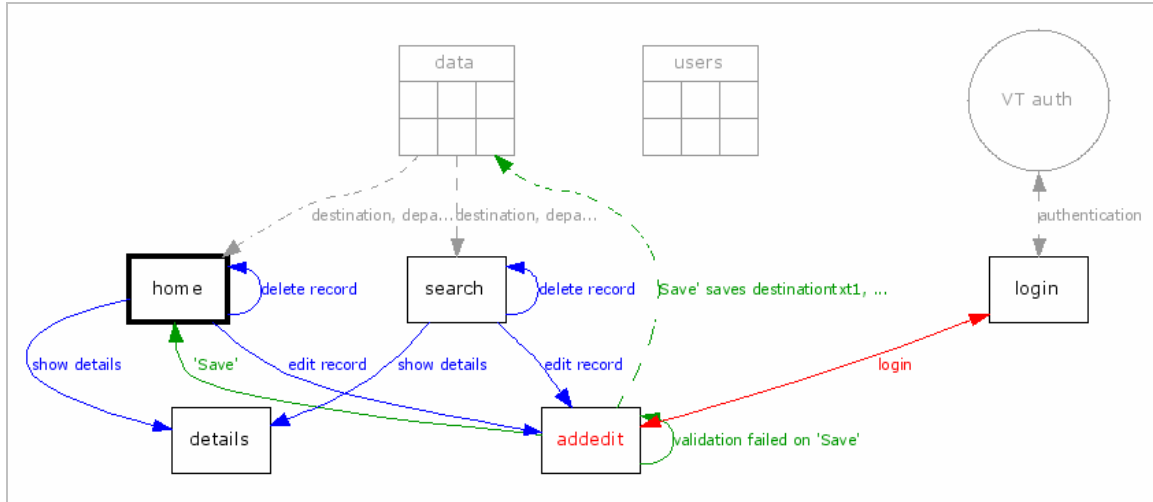


Figure 19: A sitemap automatically generated by Click

The sitemap has been implemented to a large extent by Yogita Bhardwaj who has advocated its use to aid comprehension of functionality while reverse engineering web applications (Bhardwaj 2005). Although similar in nature to the activity diagram and collaboration diagram types of the Unified Modeling Language (UML 2005), the sitemap distinguishes itself in two important ways. First, it has been designed for use by nonprogrammer end users as opposed to UML which targets mainly professional developers. Second, the sitemap is automatically generated by reverse-engineering the application's behavior definition, while UML diagrams are typically manually generated as first steps of the design process.

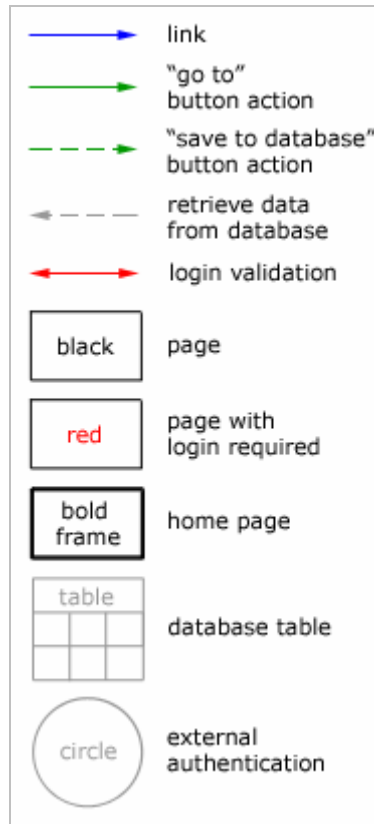


Figure 20: Legend for Click's sitemap as shown in Click's user interface

The sitemap is intended to provide an overview of the dynamic relationships between pages, database tables and the authentication system. Color coding is used to differentiate simple hyperlinks (blue) from page transitions or actions initiated by a button (green) or automatic page redirects for pages that require authentication (red) or data flows from and to the database tables (gray). The legend shown in Figure 19 summarizes the basic constructs. See Appendix F.1 for a color plate.

Besides providing a general overview, the sitemap helps developers to discover under-specification such as unreferenced pages or database tables. During the summative evaluation of Click the sitemap was not as frequently used and useful as we had hoped. A discussion of the evaluation details and conclusions is given in Section 6.2.6.3.

5.4.9 Domain Specificity

Although potentially unlimited in power (through the use of the code editing features and levels of programming support), Click is optimized for the particular domain of basic data input, storage & retrieval applications (see 1.3). This focus has simplified many design decisions and helped to determine the exact set of high-level components that should be made available to the user.

5.4.10 Session Layer

My studies of mental models of nonprogrammers (Chapter 4) have shown that most novice developers assume that web applications “remember” their state, particularly the state of input fields by default, the exact opposite of what is provided by HTTP. Click’s session layer provides short-term data persistence by default. The values entered into any input field on any page persist until they are explicitly reset due to a button action specified by the developer. The value for each input field is stored in the user’s session and retrieved for a particular input field every time the user returns to the page containing the input field component. The session layer is transparent to the developer.

5.4.11 Database Layer

One of the main shortcomings we observed in current web development tools is the limited level of integration with external services such as the database layer (2.4.5.2). Although from the point of view of the developer it is seamlessly integrated into Click, a full-featured database management system (MySQL) stores data collected via forms or holds data used for browsing, searching and reporting purposes. The user interface provided for editing the database schema and data is designed to resemble the prototypical spreadsheet application showing tabs for individual spreadsheets and the data in columns and rows. In my studies of nonprogrammers’ mental models, the spreadsheet was the most often cited metaphor for a database (see 4.2.3.5).

The need for and process of establishing a database connection is fully transparent to the developer. If a component should send data to or retrieve data from the database, the developer needs to specify only the source/target database table and in certain cases

identify a particular record or database field. The screenshot of the property dialog of the *Text field* component (Figure 41 in Appendix F.1) shows an option that asks the developer to specify a “Data Connection” for the field.

Click offers two pre-defined database tables named *data* and *users*. The first table is intended to store application data while the second is used by the optional application-specific authentication system. New database tables can be created and existing ones can be renamed (with the exception of the *users* table). The developer can add any number of custom database fields to any table.

By default, every database table contains three pre-defined fields which cannot be deleted or renamed. These fields are *id* which is used as a primary key to uniquely identify data records, *timestamp* which is used to mark the last modified time and the *lastmodifiedbyuserid* field which is used to store data record “ownership” information. The latter field can be used by the authorization layer to determine who owns a particular data record since this field stores the user-ID of the currently logged-in user whenever a data record is saved to the database. The *users* table additionally has the predefined fields “userid”, “password” (which stores the password in encrypted form), and “admin” (which is “1” if the corresponding user is an administrator and otherwise “0”).

5.4.12 Security Layer

Building from our conclusions about end users’ mental models of security (4.3), we designed a security layer that consists of a *visible* and an *invisible* part. The *visible* part is what the developer uses to configure input constraints (input validation) or to implement access control (authentication & authorization). The *invisible* part of the security layer are the facilities that Click (or an application created with Click) provides “under the hood”, mainly consisting of functions that perform internal input validation (e.g., URL parameter checking) and data escaping (e.g., escaping special characters for use in SQL statements) to guard against exploits and attacks.

Furthermore, many web applications are vulnerable due to a misconfiguration of the web server, the database, or operating system. If a novice developer has to configure security parameters the risk for mistakes is high. Click integrates the development tool

with the testing and hosting environment trying to minimize the security-related decisions a developer will have to make and thereby reduces the risks of misconfiguration (or more accurately places this burden on the IT experts that set up the Click tool).

5.4.13 Input Validation Layer

Almost any application that accepts user input can benefit from input validation. On one hand it can increase usability, promote clean data, and catch user input errors, and on the other hand increases security by blocking malicious inputs like SQL injection attacks (e.g., Loureiro 2002). Current web programming languages (e.g. PHP, ASP, JSP, ColdFusion) require the developer to manually code input validation routines. Only recently languages like ASP.NET (Microsoft 2002) offer abstractions like “validator controls” which allow the programmer to specify input constraints declaratively. Click extends this approach by allowing the developer to specify input constraints declaratively as properties of input components – similar to the way that nonprogrammers naturally think about the concept of input validation (see 4.2.3.2). Click has built-in options for frequently needed validation rules such as e-mail, date, number of characters but also allows the developer to specify custom regular expressions (context-sensitive help explains the nontrivial concept of regular expressions using examples).

5.4.14 Authentication Layer

End-user developers understand the need for authentication but are unlikely to be able to implement a secure authentication feature themselves (see 4.3). Furthermore, during the surveys (3.1.1.1), and post-study interviews (4.2.3.13, 6.1) participants often mentioned the need for their applications to integrate with the organization’s central authentication service (i.e. Virginia Tech’s central PID/password system).

In applications built with Click, users are authenticated in one of two ways. First, an application can verify the entered user-id and password against a user directory (using LDAP) which centrally holds all user account information. This method may be used for applications that have to integrate with an organization’s security context – a feature frequently requested by developers. Second, an application can verify the entered user-ID

and password against the database table *users*. If developers select this option, they will need to create a record within the database table *users* for each of the users who should be able to authenticate (a message in the “To do” list will remind them to do so).

If the developer enables both authentication methods, user-ID and password are first checked against the central user directory. If the user-ID exists but the password does not match, the authentication fails. If the user-ID does not exist, the application will attempt to authenticate the user against the database table *users*.

Regardless of which authentication methods are enabled developers will need to add a user record to the database table *users* if they want to define administrators (an administrator is a user with full privileges). However, if central authentication is enabled no password is needed in the database table *users*. The application will validate the password against the central user directory and validate the role of being an administrator against the data table *users*.

5.4.15 Authorization Layer

Nonprogrammers think about authorization checks in terms of surface features such as a non-appearing *edit* or *delete* link (see 4.3); they do not realize that the absence of a link does not necessarily mean that a page or feature is inaccessible (e.g. by directly entering the URL). Click presents authorization issues similar to the way novice developers think, but at the same time addresses security concerns more in-depth.







For example, within the property dialog of the *Data table* component, the developer is presented with an option to “show the *edit* link only for data records *owned* by the *currently logged in user*” (context-sensitive help explains this concept in more detail). If the developer has defined a login page (a page that has a *Login box* component), users will be able to authenticate themselves before using all or particular pages of the web application. The *Data table* component can compare a data record's *lastmodifieduserid* field with the user-ID of the user who is currently logged in. If these two user-ID's match, the record is said to be *owned* by the user and the *edit* link is shown, otherwise not. The *edit* link is always shown if the currently logged in user is an




administrator or if the record's *lastmodifieduserid* field is empty (which will be the case for anonymous data entries).

Pages can be designated as public or as requiring login. If a page requires login, it may allow all authenticated users to view the page or only administrators (who are identified by a “1” in the “admin” field of the “users” table; see 5.4.11) depending on the setting chosen by the developer.

5.4.16 High-level Components

Click offers components that are frequently needed for the creation of data input, storage and retrieval applications (3.3). The components have been designed to work at the level of abstraction that I observed nonprogrammers refer to during the study of mental models (see Chapter 4). The following components are implemented in the current prototype:

-  *Text, links, HTML*. Allows the display of arbitrary HTML such as formatted text, tables or hyperlinks. The developer uses a WYSIWYG editor to edit the text.
-  *Image*. Allows the display of any image. The component handles file upload and file management within the built-in image library.
-  *Text field*. Allows single or multi-line input of text and includes an optional label and input validation.
-  *Checkbox*. Enables on/off input. The component includes an optional label.
-  *Option list*. Options can be displayed as a horizontal or vertical row of radio buttons or as a drop-down select box. For each option the developer specifies the visible text as well as the value that is saved into the database.
-  *Action button*. The button component drives most of the dynamic aspects of an application. Buttons employ the concept of *action rules* which are pairs of conditions and actions that are executed when a button is pressed (more details below).

-  *Dynamic table*. The data table is the most complex and feature-rich component within Click. It is used to display data in a tabular format and includes sorting features, record set paging, linking to a details page (for the implementation of the overview/detail pattern), linking to an edit page, and record delete. The data displayed by the dynamic table component is determined by a database query (database table, fields, filters, sort order) which can either be specified in a forms-based UI or directly in SQL (in the advanced view). A preview function assists with the query definition.
-  *Dynamic text*. Enables the output of textual data. Similar to the Text component the developer specifies a formatted text using a WYSIWYG editor. Place-holders can be inserted and represent either the current value of input fields (e.g., {*phone*}; note the use of curly braces) or the value of a particular field from a database record (e.g., [*phone*]; note the use of brackets). In the latter case, a *Dynamic table* component needs to pass the name of the database table and the record id in order to uniquely identify the data to be displayed.
-  *Login box*. Handles input of user-ID and password and authentication/authorization/login/logout procedures. The component transparently handles the communication with the authentication and authorization layer.

Each input component (*Text field*, *Checkbox*, *Option list*) has a property that determines whether or not the component is linked to a particular data field in a particular database table. If enabled, input components will automatically read data from the linked field (if a record ID is passed from a *Data table* component) or save data to the linked field (if an *Action button* component defines the *Save to database* action). Regardless of this setting every input component by default maintains its value throughout the use of the application (see *Session layer*). Although not implemented in the current prototype, Click has been designed to offer the following additional components:

- *Repeating region*. An extension of the *Dynamic text* component would repeatedly display a template for each record (or a defined subset of records) within a

database table. Macromedia Dreamweaver (Macromedia 2005b) already offers a good implementation of this concept,

- *Dynamic image*. We are planning to handle images as “first-class” data-types within the database. Once this concept is implemented a *Dynamic output* component would be able to output either text or images or a mix of both,
- *Navigation/Menu*. Virtually every website has some kind of navigation (links on top or left of the contents). This component would abstract the linking, highlighting of the currently active page and the handling of drop-down sub-menus and breadcrumb-style navigation.

Most of the properties of a component can be modified within the properties dialog. However, some properties are only editable by directly modifying the layout code. We have designed Click so that all the frequently needed options can be specified through an easy-to-use UI while more advanced or obscure options can be customized by editing code. The developer can modify all component properties within the *Layout code* view (see Table 15 on page 131 for a layout code example).

Certain components (such as *Dynamic table*, *Action button*, and *Dynamic text*) are not only represented by layout code but also have behavior code associated with them (see Table 16 on page 131 for an example of the behavior code of an *Action button* component). Layout and behavior code are automatically generated and updated when the developer uses the form-based component property dialogs. Click has been designed to create easy-to-read, well-documented, and easy-to-extend code. Currently, the layout and behavior code can only be viewed and edited along with the code of all other components on a particular page. However, as a future extension we plan to enable viewing code on a per-component level in order to facilitate readability.

At runtime, all components are rendered into standard HTML and JavaScript by PRADO (Xue 2005), the component framework on which Click applications are build. This process is completely transparent to the developer who is thereby shielded from the complexity and pitfalls of client-side scripting. PRADO attempts to render standard-compliant, cross-browser HTML, CSS, and JavaScript code.

5.4.17 Button Action Rules

Action rules are pairs of conditions and actions that are executed when a button is pressed. The concept of if-then rules is a common paradigm for specifying user interface event handling in end user environments (e.g., see graphical rewrite rules: Repenning 1995; Pane 2002). They are also consistent with our general observation that end users are comfortable with simple logical patterns such as if-then (see 4.3). Figure 43 (on page 255) shows a screenshot of Click's action rule dialog.

One or more action rules can be associated with a button. The actions of an action rule are only executed if the condition is true. If the condition is false, nothing happens. A condition compares the values that a user entered into certain input fields with known values. Currently the following actions are pre-defined in Click:

- *Save data to the database...* (option 1: *All the input fields of the application*; option 2: *All the input fields from the current page*)
- *Send email...* (the developer will fill in a template email with place-holders for runtime values from input fields, e.g., *{firstname}*)
- *Reload current page* (this action may be used to refresh a search results in a data table on the current page.)
- *Reset/Clear input fields...* (option 1: *All the input fields of the application*; option 2: *All the input fields from the current page*)
- *Go to page...* (provides a list of existing pages to choose from)

The developer can also define custom actions by directly modifying the behavior code (see 5.4.19) of the application under development (advanced view).

5.4.18 Event-based Web Programming

Before the advent of event-based web programming languages (such as ASP.NET), the underlying mechanism for communicating with the user has been the *page-submit-cycle*. A web page containing input fields would send its data to a server-side script when the user pressed the submit button. The script would receive the data and in turn reply with a new web page. This model not only limits the usability of web-based

user interfaces but also creates a challenge for the programmer who has to handle a potentially complex network of scripts that send data to each other and produce output. In terms of its poor code readability and maintainability the *page-submit-cycle* is comparable to the *goto* statement in early procedural languages such as BASIC (Dijkstra 1968).

Recently, the event-based approach, which has long been used for the development of desktop applications has found its way into the web programming arena. In this approach the *page-submit-cycle* is abstracted and callback functions or event handlers are attached to button components. Click builds on top of the PRADO framework (Xue 2005) which has introduced event-based programming for the otherwise comparatively novice-friendly web programming language PHP.

I argue that in comparison to the *page-submit-cycle* the event-based paradigm is easier to learn and use for aspiring web programmers because it is already close to their natural mental models (see 4.3). However, novice web developers can use Click's pre-defined actions without even having to be aware of the event-based paradigm.

5.4.19 Separation of Layout and Behavior

Traditional web programming languages (e.g., PHP, ASP) mix layout code and business logic (behavior code) which often results in difficult-to-read and difficult-to-maintain programs. More recently, event-based languages such as ASP.NET cleanly separate the layout code from the business logic. Being built on top of the event-based PRADO framework (Xue 2005), Click places the layout code into a different file from the behavior code. Table 15 and Table 16 exemplify this separation by showing the layout and behavior code for one web page of a simple conference registration application. As shown in Table 16 Click generates behavioral code that expresses the selected actions via high-level functions (e.g., *sendEmail*, *saveToDatabase*, *goToPage*) that are implemented on top of PHP. These functions are designed to be understandable by novice programmers who want to go beyond the dialog/form-based facilities.

Table 15: Layout code for one web page of a simple conference registration application

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Event registration</title>
  <link rel="stylesheet" type="text/css" href="styles/default.css">
</head>
<body>
<com:Form>
<%include Pages.showOnEveryPage %>

<com:HtmlText ID="htmltext1" X="90" Y="46" Z="52">
  <prop:Text><h1>Welcome! Please register below.</h1></prop:Text>
</com:HtmlText>

<com:InputText ID="firstname" X="93" Y="104" Z="54" Columns="20" Rows="1"
  TextMode="SingleLine" DbFieldName="data:firstname"
  InputRequired="false" ValueType="Characters" MinValue="1" MaxValue="30">
  <prop:Label><b>First name:</b><br /></prop:Label>
  <prop:ErrorMessage>Please enter between 1 and 30 characters.</prop:ErrorMessage>
</com:InputText>

<com:InputText ID="lastname" X="92" Y="156" Z="56" Columns="20" Rows="1"
  TextMode="SingleLine" DbFieldName="data:lastname"
  InputRequired="true" ValueType="Characters" MinValue="1" MaxValue="50">
  <prop:Label><b>Last name:</b><br /></prop:Label>
  <prop:ErrorMessage>Please enter between 1 and 50 characters.</prop:ErrorMessage>
</com:InputText>

<com:InputText ID="email" X="93" Y="210" Z="58" Columns="20" Rows="1"
  TextMode="SingleLine" DbFieldName="data:email"
  InputRequired="false">
  <prop:Label><b>E-Mail:</b><br /></prop:Label>
  <prop:ErrorMessage>Please enter a valid e-mail address.</prop:ErrorMessage>
  <prop:RegularExpression>\w+([-+.]|w+)*@w+([-.]|w+)*\.|w+([-
.]|w+)*</prop:RegularExpression>
</com:InputText>

<com:Button ID="registerbutton" Text="Register" X="92" Y="268" Z="60"
  OnClick="registerbutton runActions" />

</com:Form>
</body>
</html>

```

Table 16: Behavior code for one web page of a simple conference registration application

```

function registerbutton runActions($button, $parameter) {
  $condition1 = $this->newCondition('{email}', 'empty');
  if ($condition1->isTrue())
  {
    $this->runAction('saveToDatabase', 'homepage');
    $this->runAction('resetInputFields', 'homepage');
    $this->runAction('goToPage', 'thankyoupage');
  }
  $condition2 = $this->newCondition('{email}', 'notEmpty');
  if ($condition2->isTrue())
  {
    $this->runAction('saveToDatabase', 'homepage');
    $this->runAction('sendEmail', 'conference@vt.edu', '{email}',
      'Conference registration',
      'Dear {firstname} {lastname},
      Thank you for your registration!');
    $this->runAction('resetInputFields', 'homepage');
    $this->runAction('goToPage', 'confirmationpage');
  }
}

```

5.4.20 Templating

Classical programming languages allow the specification of dynamic output through the use of variables. Click adopts but simplifies this approach by exposing input fields and database fields as pre-defined variables. For the specification of the output format of a *Dynamic text* component and for the specification of e-mail text for a *SendEmail* action Click offers the concept of templating. With the proliferation of the model-view-controller pattern the templating approach (e.g., Velocity template engine by Apache Software Foundation 2005; Smarty 2005) has become commonplace in web development. The developer can write static text but can also use place-holders that are substituted at runtime for the current values of input fields or database fields. Input field place-holders use curly braces (e.g., {firstname}). See Table 16 for an example. Database field place-holders use square brackets (e.g., [firstname]). Database field place-holders can only be used within *Dynamic text* components that are linked to from a *Data table* component which identifies the database table and data record whose data is to be displayed.

5.4.21 Parameter Passing and “Current Data Record”

The mental models studies have shown that while end users can easily imagine the mechanism of the overview/detail pattern at a high-level, they frequently do not know how it can be implemented (4.2.3.7).




In order to implement an overview/detail (or overview/edit) pattern using a *Data table* and *Dynamic text* component information about the selected data record (the database table and the id of the record) needs to be passed from the *Data table* to the *Dynamic text* component. Click transparently passes this information via the URL (in two URL parameters called “dbTable” and “id”) to the *Dynamic text* component. This technical concept is abstracted for the developer and the *Dynamic text* component simply refers to the “current data record”.

The “current data record” is a concept in Click that identifies the record (or row) within a database table that is considered *active* at the moment. If the developer uses database field place-holders within a *Dynamic text* component the component will

substitute actual values for these place-holders. The values are determined based on the “current data record”.

5.4.22 Wizards

Much of the commonly needed functionality that can be implemented with Click requires the interaction of a number of components. For example, to implement a search function, the developer needs to create at least one *Text field*, one *Action button* and a *Data table* component. These components need to be configured correctly to realize the search functionality. As our summative study of Click (6.2.9f) has shown, this is often no trivial task for a novice developer. To facilitate the implementation of common functions Click includes wizards which present the developer with a series of dialogs and create a set of related components automatically. The current prototype of Click provides the following wizards:

-  *Overview & detail*. Creates a *Data table* component that serves as the overview by listing a subset of all fields of all data records and a *Dynamic text* component that displays the details of a selected data record.
-  *Search form & results*. Creates a *Text field*, *Action button* and *Data table* component as described in the example above.
-  *Database entry/edit form*. Creates a number of *Text field* components, one for each database field selected by the developer as well as an *Action button* that is configured to save the data into the database.

Future wizards may offer functionality to create navigation menus or other commonly needed functions.

5.4.23 Pixel-based Positioning

Our observations of novice web developers have shown that even the task of positioning HTML elements on screen is nontrivial. For example, when using Click prototype #1 (5.2.3) in conjunction with Dreamweaver as a WYSIWYG editor some participants struggled to properly align input fields. Graphics software and word

processors have long had options to position any element with high accuracy through the use of absolute positioning and snap-to-grid features. Just recently WYSIWYG editors have started offering this notion but it is not yet fully embraced by the web development community for a variety of reasons including cross-platform or backwards compatibility. The state of the art is still the use (or misuse) of HTML tables to implement a particular layout and to align components such as text input fields.

Click does not offer a “silver bullet” that solves all aspects of this problem. The tool favors usability as experienced by the developer over other concerns by using CSS2 (World Wide Web Consortium 1998) for absolute positioning. Components are placed on the screen via a simple click on one of the items in the component library and can be moved with pixel-level accuracy via drag-and-drop. Although not yet implemented, a snap-to-grid function and snapping guides would further improve ease-of-use.

The two major downsides of absolute positioning are cross-platform compatibility issues due to differing screen resolutions and HTML rendering, and the fact that the pixel-based positioning does not handle variable-length elements. Currently, the two variable-length components supported by Click are the *Data table* and *Dynamic Text*. The *Data Table* has a maximum length which can be specified by the developer and if too many data records exist for the limited space, the *Data table* components uses paging. The length of *Dynamic text* is theoretically unlimited but practically often pre-conceivable by the developer. The cross-platform compatibility problem should not be a major problem for applications of low visual complexity and if components are positioned with some padding space between them that can buffer the effects of platform differences. A more robust but also more complicated approach is the use of layout managers that follow the box-model, like Flex (Macromedia 2005c). Nevertheless, I still consider layout to be a largely unsolved problem for novice web developers.

5.4.24 “Global” Components

In order to eliminate the need for the redundant definition of layout elements that are repeated on different pages (e.g. header, footer, sidebar) Click offers the concept of “global components”. Previous Click prototypes (5.2.3) had implemented the concept of

global headers and footers which proved to be somewhat limited in terms of expressiveness (e.g., a shared sidebar could not be realized).

Elements that are shared among all pages of an application such as header, footer, or sidebar can be implemented in Click by enabling the property “show this component on every page”. “Global” components are contained in a “global file” that is included into the code of each page. If the developer removes the include statement by editing the code in the *Layout code* view, the particular page will not show the global components. This adds to the flexibility of the otherwise quite basic concept.

More flexible but also more difficult to use is the concept of layout templates as offered by Dreamweaver (Macromedia 2005b) and FrontPage (Microsoft 2005a) which allows users to define templates and successively pages that are based on these templates.

5.4.25 Layers of Programming Support and Gentle Slope of Complexity

A critical tradeoff for any end-user development tool is the relationship of usability and expressiveness. Ideally a tool’s complexity is proportional to the problem to be solved: If a developer wants to take the next small step, the learning and effort required should be small as well. In practice however, most tools’ learning curve exhibits large discontinuities (e.g. having to learn many new concepts such as session management, database communication, and encryption before being able to implement a basic authentication feature). One of my EUDWeb design goals is to make the effort required more proportional to the complexity of the problem at hand. I have adopted the concept of a “gentle slope of complexity” (MacLean, Carter et al. 1990), a principle that proposes that tools should adapt and grow with users’ needs in a layered fashion. For the Agentsheets simulation tool, Repenning and Ioannidou (1997) show how an end-user development tool can offer different layers of functionality that require different degrees of sophistication, in this case ranging from direct manipulation visual construction to a full-fledged programming language. I recommend a similar approach for EUDWeb. Click implements several layers of programming support (see Figure 21).

<i>Layer 1</i>	Customizing template web applications
<i>Layer 2</i>	Using Wizards to create related sets of components
<i>Layer 3</i>	Designing via WYSIWYG, direct manipulation, parameter forms
<i>Layer 4</i>	Editing layout code (similar to HTML, ASP.NET, JSF)
<i>Layer 5</i>	Editing high-level behavior code
<i>Layer 6</i>	Modifying and extending the underlying component framework
<i>Layer 7</i>	Editing PHP code

Figure 21: Layers of Click's programming support that illustrate a “gentle slope of complexity”

At Layer 1, developers may customize existing web applications (see *Application templates* in 5.4.2); ease-of-use is high but trades off with flexibility. At Layer 2, developers may use Click's *wizards* (e.g. overview-detail page wizard, search form wizard) to create a related set of components. At the next layer, developers can use Click's form-based user interface to insert *new* components, customizing the component behavior through parameterization. If the visual layout tools are too inflexible, at Layer 4 the developer can manually edit the layout code (Table 15 on page 131); this is comparable to hand-editing HTML). The predefined high-level functions may be modified by editing the behavioral code (Layer 5; see Table 16 on page 131). At this level, developers have the flexibility to define Boolean conditions of nearly unlimited complexity but are not required to write low-level PHP code. At Layer 6 (not yet implemented in Click), developers may access the component-based PRADO framework (Xue 2005), which like ASP.NET or JSF, abstracts many of the details of web programming. Using PRADO, advanced developers can define new components (by composing existing components or creating new ones from scratch) similar to that supported by WCML (Gaedke, Schempf et al. 2000). At this level developers can also modify Click's high-level functions (e.g., change `saveToDatabase`) or create a new high-level function (e.g., `receiveRssData`) for use by themselves or other Click users. At the final and most powerful layer, experienced developers have full access to the capabilities of PHP (Click does not yet offer a form-based user interface for this or the previous layer). To gain ultimate flexibility, Click can export the full application code so that it may be used stand-alone on a separate web server.

I do not expect all users to take advantage of all layers. Concluding from our observations during the formative studies (6.1) and summative study (6.2) of Click, I anticipate that novice developers will start with the visual tools, and only explore more advanced features when they become necessary for their work. Indeed many end users may never reach the state of hand-writing code. I also do not see these layers as a "natural progression" for developers as they gain experience. It is likely that the use of these features will be quite opportunistic and vary on an individual basis.

The layers summarized in Figure 21 are specific to Click but future web development tools may implement similar facilities, perhaps leaving out, changing or introducing new layers. My intention is for Click to have a *gentle slope of complexity*: offering features and flexibility that grow proportionally with the developer's needs.

5.4.26 Collaboration Support

My design of Click recognizes that EUDWeb will rarely occur solely on an individual level but rather that it is a collaborative process (Nardi 1993). As a web-based system, Click enables a general level of collaboration among developers, in that any web application developed in Click can have one or more developers. Each of these developers can log into Click and modify the application as well as grant this right to other Click users (which will make them *developers* for this application). Because Click offers different layers of complexity and power, one possible scenario is that a novice developer asks a colleague with more advanced web development skills to extend an application by writing a custom component or behavior.

Furthermore, the Click model assumes shared responsibilities between IT personnel who maintain the tool and underlying server infrastructure (web server, database server) and developers who build applications on top of this infrastructure without being exposed to the details.

5.4.27 Integrated Development and Runtime Environment

The state of the art in web development requires the developer to use a number of tools and servers, most often at least a WYSIWYG web editor, an image editor (not yet part of Click), a file management and transfer software, a web server, and a database server. Often these tools are not all related or poorly integrated which can pose a substantial hurdle for non-technical developers (3.1.1.1). Click is an integrated environment that supports prototyping and testing and also includes a production hosting environment. Click abstracts the interfaces to the web server and database server. Finally, since Click is a web-based tool, it does not require setup on the developer's computer.

5.5 System Architecture and Implementation

Click is a web application built using the programming language PHP (Lerdorf, Gutmans et al. 1995) and the database management system MySQL (2005). The user interface is implemented in DHTML (a combination of HTML, JavaScript, and CSS). Click displays its user interface within multiple HTML frames (see Figure 22), one for the header, one for the status bar, one for the main workspace, one for the toolbox shown on the right side of the screen, one smaller frame used for layout purposes, and one hidden “action” frame used for communication with the backend.

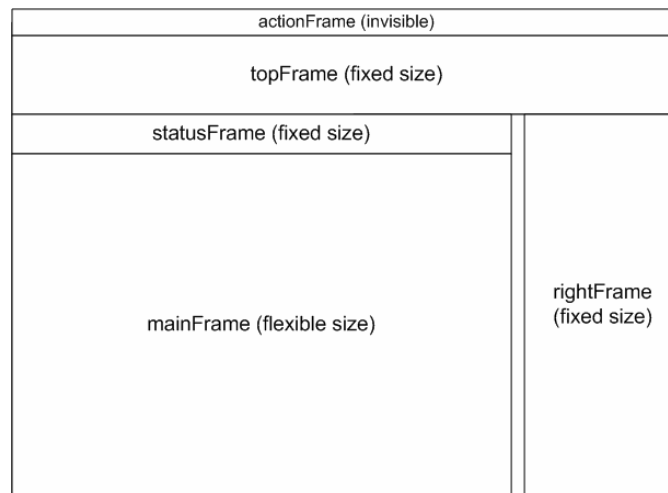


Figure 22: Click's HTML frames setup

Figure 23 illustrates Click's system architecture which follows to a large extent the Model-View-Controller (MVC) pattern originally conceived by Reenskaug (1979). Every action initiated by the developer is sent to the front controller (“do.php”) via the hidden “action” frame. The front controller executes the actions, updates the model, and returns a new view (a.k.a. user interface). The model consists of the template files which control the layout and behavior of the application (which is currently being developed) and the database which holds the application's data schema and data.

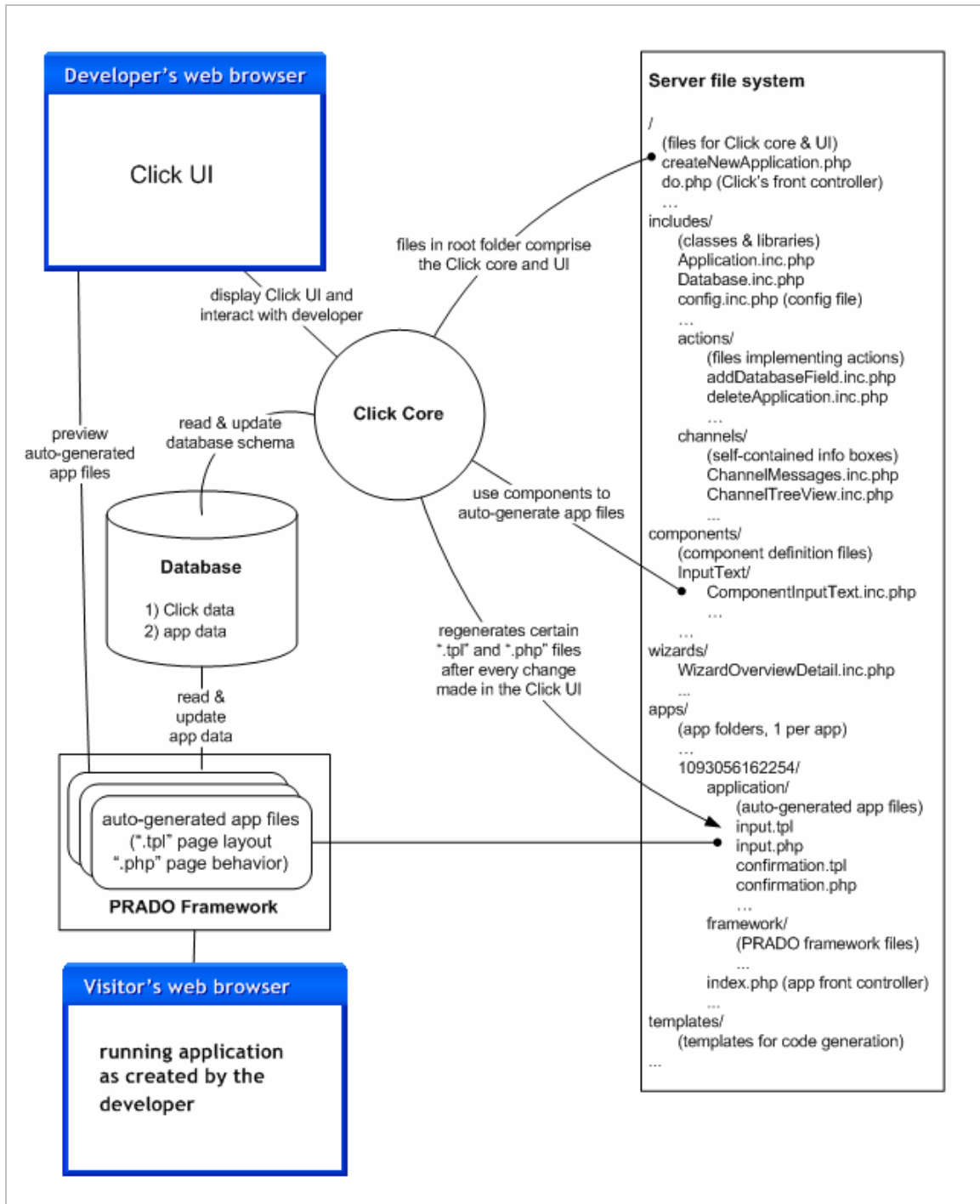


Figure 23: Click's system architecture and file system layout

Applications developed with Click are built on top of the PRADO framework (Xue 2005) which is a component-based and event-based system that abstracts many functions of web development similar to ASP.NET (see 2.1.3.3). PRADO separates the code for each page of the application into a layout template file (for an example see Table 15 on page 131) and a behavior template file (see Table 16 on page 131). With the help of Click's UI the developer can place and configure components which are automatically translated into corresponding layout or behavior code. Alternatively, a more advanced developer can directly edit the code produced by Click.

Each Click application is uniquely identified by a 12-digit ID which is also used to generate the URL for a finished application when it is made available to the public. When the developer “publishes” an application it runs directly from the Click server. There is no separate production deployment step, although future version of Click may support the deployment to remote servers.

Click is a multi-user system. An application can have one or more developers. The developer accounts and associations between developers and applications are stored in the MySQL DBMS (or alternatively in an external LDAP server). The database server also has another purpose. For each application, it contains exactly one database which holds the application's data. This database can contain one or more tables whose structure can be modified directly from within Click's *Database* view.

5.6 Summary and Conclusions

Click is a prototype EUDWeb tool and is comprised of many individual features. Table 17 shows an overview of the features and concepts that I believe to be *novel* in comparison to state-of-the-art web development tools. Some of the concepts and features listed in Table 17 are starting to appear in tools targeted at professional developers but for the most part have not yet been made accessible and usable to the end-user developer.

Table 17: Click’s Novel Concepts and Features

Click’s Novel Concepts and Features
<ul style="list-style-type: none">• Design-at-Runtime concept• IDE integrates all tools (layout, behavior, DB, testing, runtime environment, image editor not yet)• Requires no programming knowledge for developing basic database-driven web applications• Supports opportunistic development (any aspect can be easily modified at a later time)• Supports continuous workflow (developer should never have to interrupt to setup preconditions)• Offers customizable high-level components (such as dynamic tables, action buttons, input fields)• Implements concepts close to developers’ “natural” mental model (such as button action rules, input validation as input field properties, “persistence-by-default”-style session management)• Concept of “current data record” (partly) replaces parameter passing concept• Tightly integrated database management system• Auto-generated sitemap (not frequently used in evaluation study, see 6.2.6.3)• To-do list (evaluation showed mixed results, see 6.2.6.2)• Integrated page-level authorization system (evaluation showed need for improvements, 6.2.7.1)• Exposes a “gentle slope of complexity” through layers of programming support ranging from high-level pre-defined functions to allowing custom PHP code (has not yet been evaluated)• Integrated runtime/production hosting environment

The next chapter describes how Click’s features have been shaped by formative evaluation (6.1) and, how and to what extent they enable EUDWeb (see 6.2).



Chapter 6

Evaluation of Click

6.1 Formative Evaluations

Click's design has emerged over an 18-month iterative cycle of prototyping and formative evaluation. In addition to obtaining informal feedback from colleagues, professors, and external users (Click is available as open-source software), we conducted three scheduled formative usability evaluations. During each of the three sessions, 4-6 users were asked to develop a basic event registration application with slightly differing requirements. Figure 24 shows an example of a specification used during the formative evaluation sessions. Other examples were used to evaluate different parts of Click's user interface. The participants were given diagrammatic specifications and asked to create the application from start to finish. In order to guarantee that participants matched our target audience, we pre-selected them based on their (lack of) web programming knowledge.

During the studies the participants were asked to think aloud (Lewis 1982) and we collected information about all usability incidents encountered. After each study the recorded usability problems were summarized across all participants and ranked according to severity which helped us guide and prioritize the continued development of Click. See Table 18 (page 145) for an example. Full results are available online (Rode and Bhardwaj 2004a). In the later phases of development we used the collaborative bug tracking and feature tracking capabilities of Click's project website on SourceForge (Rode, Bhardwaj et al. 2005) for recording software faults and usability problems.

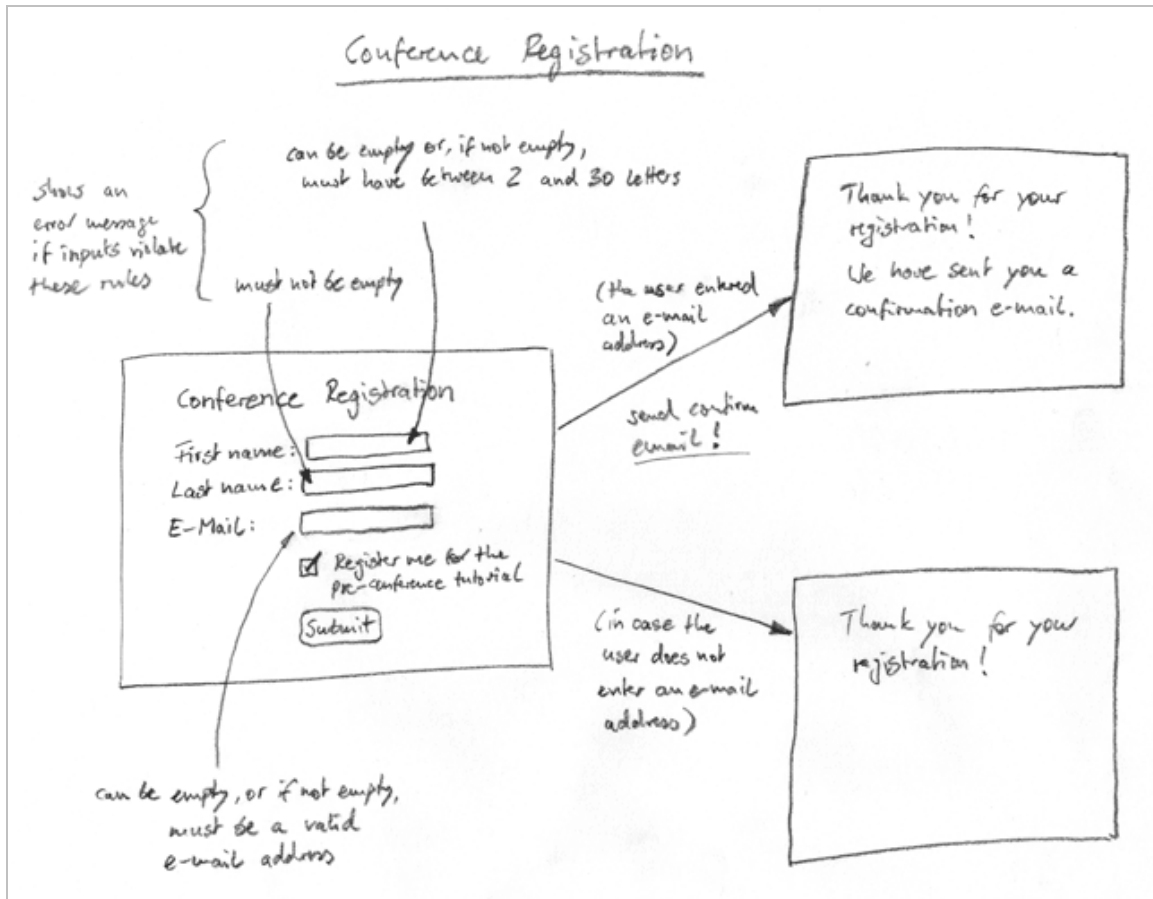


Figure 24: Example of a specification used during the formative evaluation sessions of Click

At the end of each usability testing session, participants were asked to respond to a questionnaire that was designed to gauge their subjective impression of particular Click features and elicit ideas for improvements. The full results from of the post-study questionnaire are available online (Rode and Bhardwaj 2004b; Rode and Bhardwaj 2004c; Rode and Howarth 2004).

6.1.1 Evaluation of Prototype #1

We conducted one formative evaluation for each Click prototype. In the first study, conducted in May 2004, we evaluated prototype #1 (5.2.3) which was a web application that worked in conjunction with an external WYSIWYG editor. Dreamweaver MX (Macromedia 2005b) was used during the study. The most important

usability issue uncovered during this study was that participants frequently were unsure about whether to implement a certain feature within the WYSIWYG editor or within the web application. For example, when asked to implement input length restrictions for a text field, most participants gravitated to editing the HTML “maxlength” property (exposed in Dreamweaver’s property dialog as “Max chars”) rather than using the more advanced and secure input validation features provided by Click. Due to the poor workflow, which was caused by switching between WYSIWYG editor and Click and the inconsistent user interfaces, participants were often confused and lost track of the task at hand. These problems motivated the implementation of Click prototype #2 as a stand-alone web application that includes features for layout definition.

Table 18: Example from a usability problem list as used during formative evaluation

Criticality	Problem	Participant	Potential solutions
critical	users did not understand how branching in action rules work; some expected to have to select one "if statement" and then specify the positive and negative branch (if-then-*else*) .	1,2,3,4,6	<ul style="list-style-type: none"> • Redesign to consider the else branch; perhaps more graphical • Provide an example
important	users had difficulties getting started (after creating a web app); were stumped by blank page after login; did not find "What's next" message very helpful	1,2	<ul style="list-style-type: none"> • Have a better “To-do” list message. More concise, direct • Put a notice directly onto the blank page, e.g. “To get started...” • Have a tutorial video (screen capture with sound)
normal	users noticed the preview in the query editor too late since it was below the fold	1,2,3	<ul style="list-style-type: none"> • collapse the filter and sort by default which will bring the preview above the fold and make the selection more understandable
...

6.1.2 Evaluation of Prototype #2

In the second study, conducted in July 2004, we evaluated prototype #2 (5.2.3) – a monolithic web application which provided a set of pre-defined components but was not extensible beyond the pre-defined functionality. Overall, in comparison to prototype #1 participants encountered fewer critical usability problems. However, a number of critical problems remained that had not yet been addressed in the redesign. Most notably, the implementation of branching behavior (see Figure 24) using the conditional guard of button action rules was unclear for two reasons. First, the majority of participants did not immediately recognize that branching behavior could be implemented using button rules. Rather, some participants expected the branching behavior to be specified within the properties dialog of the email text field (as shown in Figure 24, the condition involved the email field) while others looked for a global “behavior editor”. Second, several participants expected to be able to specify an “else” branch within the “if-then” action rules – a concept which is not yet available in Click (see example in Table 18). Many less critical, yet important usability problems were uncovered such as participants’ desire for an explicit preview-only function which prototype #3 provided in addition to the design-at-runtime (5.1) functionality. A major shortcoming of Click’s monolithic architecture became apparent when participants indicated their need for advanced custom functionality during the studies’ debriefing session. Prototype #2 was not extensible by the end user but rather required expert programmers to create new pre-defined components. The final prototype #3 (which was very close to the current implementation of Click as discussed in 5.3 and 5.4) features a complete redesign of the architecture making Click extensible at different levels of power and complexity (5.4.25).

6.1.3 Evaluation of Prototype #3

In the third study, conducted in November 2004, we evaluated prototype #3. This last formative study replicated many of the findings of the previous studies for yet unsolved usability problems but did not uncover any significant new problems. Again, the full results are available online (Rode and Bhardwaj 2004a).

Finally, the *summative* study of Click's final design which we conducted in April 2005, also revealed problems that point out remaining weaknesses of Click's design and implementation. These findings are discussed in the Section 6.2.9.

6.2 Summative Evaluation

As a final step in this EUDWeb project, we conducted a summative evaluation study to test the overarching proposition embodied by Click: *Using Click, nonprogrammer developers are able to create a basic database-driven web site within a short amount of time.*

Click implements a large set of features (see Section 5.4) to address many problems that nonprogrammer web developers are confronted with when developing web applications (see Chapter 3). Instead of evaluating every feature provided by Click (which would require a substantial number of experimental studies), we focused on evaluating how Click helps to address a subset of seven concerns that capture some of the most prominent issues uncovered during the requirement analysis (see Table 14 on page 112). Each of these concerns can be grouped under the general problem areas of complexity, integration, security, and feedback. The concerns point out, that end-user developers:

- cannot implement applications using low-level constructs (complexity),
- do not understand the stateless nature of the web (complexity),
- naturally specify input constraints declaratively (complexity),
- understand but cannot implement crucial database concepts (complexity),
- lack “holistic guidance” and struggle to find missing/faulty behavior (integration),
- think about security just in terms of surface features (security),
- test their work frequently during development (feedback).

I have adopted Scriven’s (1967) approach of *mediated evaluation* to investigate how certain features within Click address the aforementioned problems. Scriven uses two dimensions to classify evaluation efforts. The first dimension distinguishes the purpose of the evaluation. *Formative* evaluation is meant to provide information that helps to shape and prioritize the design process during successive prototyping iterations, while

summative evaluation is meant to assess how well a particular design matches particular design goals (e.g., task can be accomplished in less than 1 hour).

The second dimension distinguishes the process of collecting evaluation data. In a *pay-off* evaluation “hard” data is being collected (such as times, error rates, or think-aloud protocols). In an *intrinsic* (or theory-based) evaluation particular features of an artifact are analyzed and rationalized. The advantage of pay-off evaluation is that it results in “hard facts”; the disadvantage often is the lack of interpretation. Intrinsic evaluation provides the rationale for particular features and can therefore complement the facts discovered in a pay-off evaluation. *Mediated* evaluation combines the advantages of these two types of evaluation. First, the rationale for particular features is given and then these features are subjected to a pay-off evaluation to see how well the rationale matches the actual usage. In the following section, I describe particular propositions which are derived from the aforementioned problems and observations. Part of the rationale for these propositions can be found in the discussion of Click’s design solutions (see 5.4).

6.2.1 Propositions

Aside from my main proposition which states that: *using Click, nonprogrammer developers are able to create a basic database-driven web site (Online ride board application) within a short amount of time*, I have developed 12 more specific propositions that each represent a feature of Click which was designed to address a particular problem within the general problem areas of complexity, integration, security, and feedback. These more targeted propositions are summarized in Table 19.

Table 19: Propositions for the summative evaluation of Click

Problem/Observation	Proposition
Complexity	
<ul style="list-style-type: none"> • End-user developers cannot implement applications using low-level constructs 	<ul style="list-style-type: none"> • End-user developers understand how to use the high-level button action rules • End-user developers understand how to use the high-level “Table” component

<ul style="list-style-type: none"> • End-user developers do not understand the stateless nature of the web 	<ul style="list-style-type: none"> • End-user developers expect “persistence-by-default” • End-user developers understand the “clear input fields” action
<ul style="list-style-type: none"> • End-user developers naturally specify input constraints declaratively 	<ul style="list-style-type: none"> • End-user developers understand how to set up input validation for text fields
<ul style="list-style-type: none"> • End-user developers understand but cannot implement crucial database concepts 	<ul style="list-style-type: none"> • End-user developers understand the concept of “current data record”
Integration	
<ul style="list-style-type: none"> • End-user developers lack “holistic guidance” and struggle to find missing/faulty behavior 	<ul style="list-style-type: none"> • End-user developers rarely feel “completely lost” • End-user developers successfully use the To-Do list when they feel “lost” or are unsure about how to proceed or what is left to be done • End-user developers successfully use the sitemap when they feel “lost” or are unsure about how to proceed or what is left to be done
Security	
<ul style="list-style-type: none"> • End-user developers think about security just in terms of surface features 	<ul style="list-style-type: none"> • End-user developers understand how to set up “page access restrictions” • End-user developers understand the concept of a data record that is “owned” by a user
Feedback	
<ul style="list-style-type: none"> • End-user developers test their work frequently during development 	<ul style="list-style-type: none"> • End-user developers are comfortable with and frequently use the runtime feature of the “design-at-runtime” concept

6.2.2 Participants

I recruited 6 participants (3 male, 3 female) representing my core target audience of nonprogrammer webmasters through an online screening questionnaire (see Appendix G.3). I had previously emailed the questionnaire in the form of a link to an online survey

to all administrative users of Virginia Tech's web hosting service. Participants were selected from all the survey respondents who had indicated having at least fundamental web master knowledge (a response of "3" or higher to at least one of the questions shown in Table 20) but no web programming knowledge (a response of "1" to the question shown in Table 21). Table 22 shows the self-reported experience of the selected participants.

Table 20: Two questions about "web master knowledge" from participant selection questionnaire

1) How do you rate your knowledge of your primary visual web development tool (Frontpage, or Dreamweaver, GoLive etc.)? (1=no knowledge, 5=expert knowledge)
2) How do you rate your knowledge of HTML? (1=no knowledge, 5=expert knowledge)

Table 21: Question about "web programming knowledge" from participant selection questionnaire

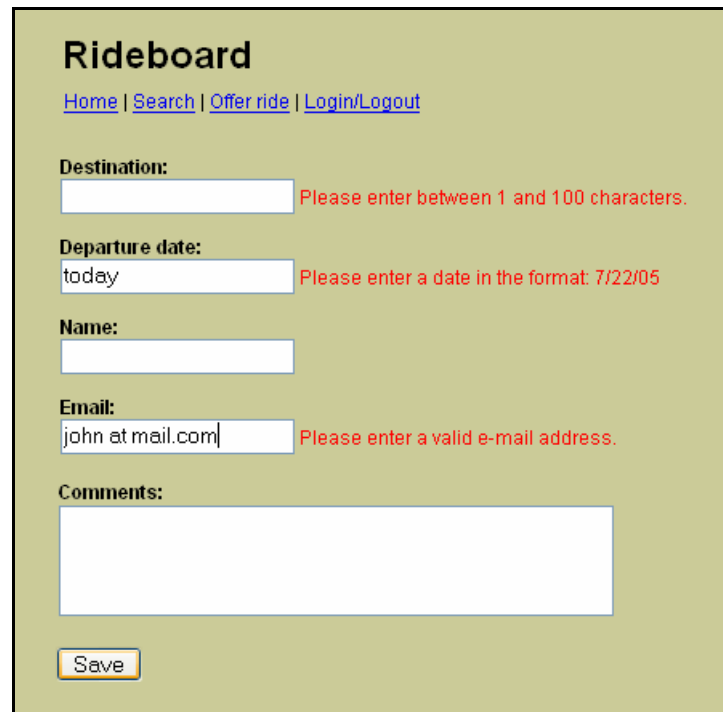
How do you rate your knowledge in web programming (use of Javascript, PHP, ASP, or Java etc.)? (1=no knowledge, 5=expert knowledge)

Table 22: Participants from summative evaluation and their self-reported experience on the online pre-study selection questionnaire (1=no knowledge, 5=expert knowledge)

Participant (gender)	Job	WYSIWYG editor	HTML	Web Programming	Database
P1 (m)	faculty	4	5	1	2
P2 (f)	staff	1	4	1	2
P3 (f)	faculty	4	2	1	2
P4 (m)	staff	3	4	1	2
P5 (f)	staff	2	3	1	3
P6 (m)	faculty	3	2	1	1
Mean (Std. dev)		2.83 (1.17)	3.33 (1.21)	1 (0)	2 (0.63)

6.2.3 Methods

Within a laboratory setting together a research assistant and I observed each participant individually while the participant was constructing a basic database-driven web site – an online ride-board application. As a way to specify the application’s requirements we provided a working example application that the participant was able to access throughout the study. Figure 25 shows one screenshot of the example application (see Appendix G.5 for screenshots of all five screens). Figure 19 (on page 120) shows the Click’s sitemap containing the five pages that comprise the application.



Rideboard

[Home](#) | [Search](#) | [Offer ride](#) | [Login/Logout](#)

Destination:
 Please enter between 1 and 100 characters.

Departure date:
 Please enter a date in the format: 7/22/05

Name:

Email:
 Please enter a valid e-mail address.

Comments:

Figure 25: Screenshot of the "Offer ride" page from the example application

After completing the Institutional review board procedures (participant read and signed the informed consent form; see Appendix G.2) I read aloud a short instruction sheet (see Appendix G.4) that introduced the goals and individual steps of the study. After finishing the instruction phase, the participant was given the opportunity to ask questions to clarify any misunderstandings about the study procedures. Next, the

participant watched a 14-minute video (Rode 2005) that introduced the basic concepts and features of Click. In order to promote concentration the participant watched the movie in private. Next, the participant was asked to explore the example ride board application in order to discover and understand the exact functionality s/he had to implement using Click. In addition a one-page sheet that guided the participant through all the functions of the application was provided (see Appendix G.6). Next, I read aloud a second short instruction sheet (see Appendix G.7) that asked the participant to work autonomously without asking for help unless s/he would feel “completely lost”. Finally, the participant began to replicate the ride board application using Click.

The research assistant and I quietly observed the participants’ actions only intervening when they asked for help. If participants asked for help without having spent sufficient effort on a problem, they were encouraged to keep trying. However, once participants “gave up”, the research assistant logged this event as a *critical incident* (using a custom-built logging application) and I helped them to solve the problem at hand by giving hints about the source of the encountered difficulties. Infrequently (about 1-3 times per study), I gave unsolicited comments to help the participants across minor but time-consuming usability issues such as where to find the link button, or how to delete a component. These small problems were not logged as critical incidents as they can be easily fixed and are easily understood and remembered. We did, however, log these problems as usability issues.

At the end of the study the participant responded to an online questionnaire (see Appendix G.9) that contained questions focusing on the subjective evaluation of particular problems and Click features. Furthermore, where necessary, the participant was asked to explain the reason or thinking behind critical incidents as they have occurred during the use of Click.

The laboratory setup consisted of a Windows XP PC which was set up running two monitors at the screen resolution of 1024x768. The left monitor displayed the example application, while the right monitor was used to (re-)create the application using Click. Mozilla Firefox 1.03 was used as the web browser. The PC was instrumented with Morae (TechSmith 2005) – a software used to record the participant’s screen actions and

voice. The research assistant and I observed each participant, sitting slightly behind and to the side, while taking notes (critical incidents, small usability problems, and other observations about the participant’s approach). After concluding the study, we combined our handwritten notes into one document per participant, organized into nine categories that matched the propositions discussed earlier. To complement the handwritten notes and the screen capture, and to facilitate data analysis Click had been instrumented to log every action performed by the participant. This log (see Table 23 for an example) was later used to extract information about time spent in particular aspects of development such as developing, testing, reading documentation, or handling a critical incident. Based on this log a visualization of the development timeline was created (see Figure 26 on page 169 for an example or Appendix G.10 for color plates of *all* timeline visualizations). The activity log shown in Table 23 also illustrates how the beginning and end of a critical incident was marked by the facilitator.

Table 23: Excerpt of participant's 6 activity log (facilitator’s logging of critical incidents in bold)

```

...
2005-05-13 09:29:21 128.173.41.22 participant6 1115990928494
openAddComponent action=openAddComponent&type=HtmlText
...
2005-05-13 09:54:56 128.173.144.114 facilitator noActiveApp
startEmergencyHelp noQueryString
...
2005-05-13 09:55:14 128.173.144.114 facilitator noActiveApp
stopEmergencyHelp noQueryString
...

```

The sessions lasted an average of about 2.5 hours. Each participant was paid \$30 for participating in the study.

The following sections assess the findings regarding the specific propositions that motivated this study (6.2.1). Finally, this summative study also revealed problems that point out remaining weaknesses of Click’s design and implementation. Therefore, at the end we also report the critical incidents encountered by the participants along with general observations of the participants’ development approaches. Although these observations are formative rather than summative data, they clarify the shortcomings of Click and point out where this prototype does not match end-users’ mental models.

6.2.4 Results on the Overall Success

At the onset of the summative evaluation, I hypothesized that: *Using Click, nonprogrammer developers are able to create a basic database-driven web site (Online ride board application) within a short amount of time.*

All participants finished replicating the full functionality of the example application within a time span ranging from 46 minutes (minimum) to 119 minutes (maximum). However, note that I helped participants to find the source of the *critical incidents* (see 6.2.9) they encountered. The number of critical incidents encountered varied from 0 to 11 per participant. Also, the severity of the critical incidents varied considerably.

Table 24 gives an overview of the time participants took to explore the example application, the development time, the number and type of critical incidents, and the participants' ratings of the statement "*Overall, Click is easy to use*" on a scale from 1 (strongly disagree) to 5 (strongly agree). The last column shows a "success" rating score assigned by the two facilitators. After each session both facilitators independently rated the statement: "*Overall, the participant's approach was successful*" on a scale from 1 (strongly disagree) to 5 (strongly agree).

Overall, 4 out of 6 participants (P1, P2, P5, P6) appeared to be mainly successful in replicating the example application's functionality. Interestingly, the participants' subjective rating of Click's ease-of-use did not always match our assessment of their success. In the extreme, participant P5 finished implementing the application in about half of the average time without encountering a single critical incident. However, her subjective rating was to our surprise only 3 out of 5.

Table 24: Times, critical incidents, participant’s and facilitators’ ratings from summative evaluation

Participant (gender)	Time to explore example app. [min]	Time to develop [min]	# Critical incidents	Types of critical incidents (see 6.2.9)	Participant’s rating of “Overall, Click is easy to use” [1-strongly disagree, 5-strongly agree]	Facilitators’ rating of participant’s success [F1: X out of 5; F2: X out of 5]
P1 (m)	10	99	4	a, f, j, v	4	4; 4
P2 (f)	6	73	2	e, p	4	4; 4
P3 (f)	6	118	11	b, c, d, h, j, o, p, r, s, t	3	2; 2
P4 (m)	6	119	8	a, f, k, l, m, q, u, v	4	3; 3
P5 (f)	3	46	0	-	3	5; 5
P6 (m)	6	103	5	a, g, i, n, v	4	3; 4
Mean (Std. dev)	6.17 (2.23)	93 (28.45)	5 (4)		3.67 (0.52)	3.5 (1.05); 3.67 (1.03)

6.2.5 Results on the Problem of Complexity

From the perspective of the end-user developer the web technologies currently needed to implement an average web application are simply too numerous and too complex, which creates the most critical entry barrier to EUDWeb (3.4). Selected important observations made during the study of mental models (see Chapter 4) and during the formative evaluations of the Click prototypes (6.1) are that end-user developers typically:

- cannot implement applications using low-level constructs,
- do not understand the stateless nature of the web,
- naturally specify input constraints declaratively,
- understand but cannot correctly implement crucial database concepts.

Click’s design addresses these problems in a number of ways. The following sections discuss to what extent *selected* Click features succeed in resolving the problem of complexity.

6.2.5.1 The High-level Button Action Rules

At the onset of the summative evaluation, I hypothesized that: *End-user developers understand how to use the high-level button action rules.* We observed few complications in the usage of button action rules. Furthermore, five of the six participants strongly agreed (and one participant agreed) with the statement: “*Now I understand how to define button action rules. (1=strongly disagree, 5=strongly agree)*”. Apparently, tying the actions (“save to database”, “go to page” etc.) to a button felt “natural” to our participants.

However, participants also looked for button actions, when in fact Click did not offer them. In particular, the implementation of the search function caused a number of critical incidents because most participants repeatedly tried to find a “search” button action (although a dynamic table component with correctly configured filter conditions was needed). In a seemingly desperate attempt P1 even tried to employ the button action rule conditions to implement a search; P3 tried to use the “Save to database” action because no other appropriate action seemed available. We concluded that the implementation of “search” functionality may be less problematic if it were provided as a predefined button action.

In conclusion, the concept of basic button action rules appears to be an intuitive concept for end-user developers. However, the summative study only tested unconditional action rules. Previous formative evaluations of Click had uncovered conceptual problems and resulting usability issues with the conditional guard of action rules (see 6.1.2). Further work is needed to conceive concepts that allow end-user developers to intuitively define actions depending on complex conditions. A promising alternative to the currently used set of independent rules are multi-way if-statements in the format `if-elseif-elseif...else`.

6.2.5.2 The High-level Dynamic Table Component

At the onset of the summative evaluation, I hypothesized that: *End-user developers understand how to use the high-level “Table” component.* When asked to rate the statement: “*Now I understand how to set up and use a Dynamic table component. (1=strongly disagree, 5=strongly agree)*”, two participants (P4, P6) chose “3” and the remaining four participants chose “4”. Overall, the participants did not appear to have problems using and customizing the dynamic table component to implement the list functionality on the *Home* page. Only participant 6 encountered a critical incident before starting to use the dynamic table component (see 6.2.9g).

Slightly more difficult than initially configuring the dynamic table component was the configuration required for setting up the overview/detail relationship between the data rows on the *Home* page and the dynamic text component on the *Details* page. However, participants did not perceive this as a major stumbling block either as the ratings for the following statement indicate: “*Now I understand the relationship between a Dynamic table component and the Dynamic text component. (1=strongly disagree, 5=strongly agree)*”. All but one participant, chose “4”; only participant 4 chose “3”.

In conclusion, the high-level configurable dynamic table component appeared to be an appropriate abstraction for listing data, providing edit and delete functionality as well as linking individual records to a details page. The dynamic table component is an example for components that work close to the natural mental model of end-user developers at a high level of abstractions.

6.2.5.3 The Input Validation Features of the Text Field Component

At the onset of the summative evaluation, I hypothesized that: *End-user developers understand how to set up input validation for text fields.* The implementation of input constraints for the *Offer ride* screen did not cause problems for any of the study participants. Declaring constraints as properties of the text input fields appeared to be natural. Again, five out of six participants strongly agreed (and one participant agreed) to the statement: “*Now I understand how to set up input validation for text fields (e.g. to require input of a valid e-mail address). (1=strongly disagree, 5=strongly agree)*”.

However, we observed that some participants initially tried to use the input validation UI to expand the visible size of the *Comments* field (a multi-line text input field). This was due to the fact, that these participants did not notice the collapsed “Visible width and height” property, likely a small usability problem rather than conceptual mismatch.

In conclusion, exposing input validation as properties of the associated input fields appears to be a close match to end-users’ expectations.

6.2.5.4 Persistence-by-Default and the “Clear Input Fields” Action

At the onset of the summative evaluation, I hypothesized that: *End-user developers expect “persistence-by-default”*. In my studies of nonprogrammers’ mental models of web programming concepts I have observed that end users expect that the application maintains and remembers the current state such as the value of input fields (see 4.2.3.1). For the most part this finding was reinforced by the summative evaluation of Click.

Furthermore, the post-study questionnaire asked a direct question to explore the expectations of the participants (see Table 25). Three participants strongly agreed with the statement (“5”), two participants chose “4”, and one participant chose “3”.

Table 25: Question targeted at exploring participants' expectations towards state persistence

(5c) *Think about building an application that has input fields (such as text fields and checkboxes) on a number of different pages.*

I would expect the input fields to automatically "remember" the data that the user has entered when s/he moves between pages (as opposed to automatically clearing the fields when the user goes to another page).

(1=strongly disagree, 5=strongly agree)

However, we also observed a few incidents where the “persistence-by-default” was not the intended or expected behavior. For example, when submitting a new data record from the *Offer ride* screen, participants expected that the fields would be cleared right after the data has been saved to the database. Since we had observed this

expectation in our pilot studies, the button action “Clear input fields” is now enabled by default but can be easily disabled by removing a checkmark.

Furthermore, I hypothesized that *End-user developers understand the “clear input fields” action*. Four out of the six participants strongly agreed, one participant agreed and one participant strongly disagreed with the statement: “*Now I understand why it was necessary to define a “Clear input fields” button action on some screens (e.g. the “Offer ride” screen). (1=strongly disagree, 5=strongly agree)*”.

Although the “Clear input fields” action did not appear to cause much confusion, (in hindsight) this study may not be a good measure since “Clear input fields” is enabled by default and did not have to be disabled for any of the functionality required by the example ride board application. Participant 3 who indicated in the questionnaire that she did not understand the action, may not have consciously noticed its presence.

In conclusion, I recommend that EUDWeb tools offer a transparent persistence layer that automatically maintains the values of all input fields. At the same time, the developer should have clearly visible options to reset input fields to their default values. Ideally, the tool is aware of situations that require the opposite default as discussed before on the example of the “submit new data record” screen.

6.2.5.5 Place-Holders and the Concept of “Current Data Record”

At the onset of the summative evaluation, I hypothesized that: *End-user developers understand the concept of “current data record”*. The implementation of the overview/details relationship between the table on the *Home* page and the *Details* page caused a number of problems and even critical incidents for all participants except P1 and P5. However, these problems did not seem to be directly related to the concept of “current data record” but more so to the general mechanism behind the dynamic text component, and especially the place-holder concept. In particular, participants often appeared to be confused about the type of place-holder to choose (input field place-holders or database-field place-holders). The distinction was unclear if at all noticed.

Once the participants had sorted out the problems with correctly defining the dynamic text component, they seemed to immediately grasp the nature of the relationship

between the dynamic table and dynamic text although they may not have noticed that Click referred to the relationship as the “current data record”. Five participants agreed with the following statement while one participant assumed a neutral position (“3”): *“Now I understand the relationship between a Dynamic table component and the Dynamic text component. (1=strongly disagree, 5=strongly agree)”*.

In conclusion, the templating/place-holder feature exposed by the dynamic text component has proven problematic for novice developer. Especially the exact role and difference between the two types of place-holders is non-intuitive. A possible extension to the concept of place-holders, which might improve its usability, is the displaying of the value that the place-holder currently stands for (e.g. `{editpage:firstname}` (currently: “Jochen”). Note that this idea has not yet been implemented.

The concept of “current data row” may be an appropriate abstraction as participants quickly understood the overview-detail relationship once they had overcome the hurdle of correctly configuring the dynamic text component.

6.2.6 Results on the Problem of Integration

Web application development involves many different technical activities such as graphical design, layout, business logic programming, database schema design, server configuration, cross-platform compatibility testing, and publishing. Current tools typically target only one or few of these activities and leave it up to the developer to assemble the right set of tools and to integrate the workflow. This lack of integration poses another critical entry barrier to EUDWeb (see 3.4). During the review of state-of-the-art development tools (2.4.5) we noted that end-user developers lack “holistic guidance”. Furthermore, the studies of end-users’ mental models (Chapter 4) and this summative evaluation (6.2.9) show that end-user developers often struggle to notice missing functionality or uncover faulty behavior.

Click’s design addresses these problems in a number of ways. The following sections discuss to what extent *selected* Click features succeed in resolving the problem of lack of integration and workflow.

6.2.6.1 Integrated Layout Tools, Database, Testing, and Publishing

During the development phase of the summative evaluation the participants appeared to switch between the different activities of layout, behavior definition, testing, and database design effortlessly with few exceptions. Only the final publishing step caused a number of critical incidents. This problem was a small usability issue – due to a poorly placed publish button – rather than a conceptual issue.

Overall, the level of integration seemed to be a good match to end-user developers' expectations.

At the onset of the summative evaluation, I hypothesized that using Click: *End-user developers rarely feel "completely lost"*. The number of critical incidents encountered by the participants (see Table 24) indicates that this proposition must be rejected to the most part. However, the participants' ratings of the question: *"While designing the application I felt "completely lost" and did not know how to proceed. (1=hardly ever, 5=very often)"* indicate that their *perception* is not too negative. Two participants rated the statement with "1" (hardly ever) and "2", and the remaining four participants chose the midpoint "3". Table 24 shows the critical incidents or episodes where participants felt "completely lost".

6.2.6.2 The To-Do List

At the onset of the summative evaluation, I hypothesized that: *End-user developers successfully use the "To-Do" list when they feel "lost" or are unsure about how to proceed or what is left to be done.* We observed, that the To-do list was not used as often as we had imagined, despite frequent encouragements during the video tutorial and study instructions. Table 26 shows the ratings for two statements that indicate the use and perceived usefulness of this feature. It is apparent that participants' opinions differ considerably. Participant 5 (she encountered no critical incidents) did not (or only rarely) look at the To-do list (or sitemap) and did not rate its usefulness. Within the group of participants, P3 and P4 rated the To-do list as most useful (4/5 and 5/5 respectively) and also both mentioned it when answering the question *"What are the top 3 aspects of the Click tool you like?"* Interestingly, those were the two participants who encountered the

highest number of critical incidents. Although the To-do list obviously did not prevent the critical incidents that they encountered, it may have prevented additional ones.

Table 26: Participants' ratings on frequency of use and usefulness of To-do list

Statement from questionnaire and scale	1	2	3	4	5	N/A
I looked at the "To do" list. (1=hardly ever, 5=very often)	1 (P5)	3 (P1,2,3)	1 (P6)	1 (P4)	-	-
When I did look at the "To do" list I found it useful. (1=strongly disagree, 5=strongly agree)	-	1 (P6)	2 (P1,2)	1 (P3)	1 (P4)	1 (P5)

We observed that when participants encountered difficulties they often considered messages in the To-do list, although this was not always their first instinct. Participants occasionally used the To-do list's action links (P3 and P6 praised this feature in the questionnaire), in particular to create the first custom fields in the database and to rename pages with generic names (e.g., untitled1). However, the To-do list also showed a number of disadvantages.

First, some participants did not understand the meaning of some messages. For example, P3 was confused about a message that indicated that "*The database table "data" has no custom-defined fields*"; she did not understand what "data" refers to since it is a predefined DB table that she did not know about. Also, in a few cases the wording of messages was too "technical" for the participant another indication that wording impacts usability to a great extent.

Second, in a few cases participants referred to the To-do list while approaching a critical incident; although the To-do list showed messages, none were relevant to the problem at hand. This was in many cases no more than a small disappointment for the participant, but in other cases severely misleading (in particular, when the participant did not notice that the To-do list message was unrelated). For example, participant 4, while intending to build the data entry form for the *Offer ride* page, noticed a message about a missing dynamic text field that was referenced from the table on the *Home* page. He

clicked on the To-do list's action link and started to create a dynamic text field, thinking he worked on the data entry form.

Third, occasionally participants could not re-create a particular feature because they had initially used the action link of a To-do message which was no longer available. In particular, this was the case with creating additional database fields. A possible solution to this problem would be to provide a “show me how” function that walks the developer through the steps for addressing the problem instead of simply providing a direct link.

Finally, the term “To-do” list itself may be inappropriate as a comment by P3 indicates: “There’s nothing on my To do list. That never happens. Does it mean I’m done? [giggles]”.

In conclusion, the To-do list feature with its current (limited) level of sophistication does not provide a clear advantage to end-user developers. However, the ratio of cost and benefits is likely to improve, if the messages displayed in the To-do list become easier to understand and more context-relevant – a non-trivial problem.

6.2.6.3 The Sitemap

At the onset of the summative evaluation, I hypothesized that: *End-user developers successfully use the Sitemap when they feel “lost” or are unsure about how to proceed or what is left to be done.* We observed, that the sitemap was rarely used, even in episodes that resulted in critical incidents. This was despite repeated encouragements during the video tutorial and study instructions. Table 27 shows the ratings for two statements that indicate the use and perceived usefulness of the sitemap. If participants viewed the sitemap, in most cases they did not spend the time to understand the visualization, quickly returning to actively building the application. However, occasionally participants expressed their esteem for a feature that “could potentially be useful in the future when building more complex applications” (comment paraphrased). In only one case, there was clear and immediate benefit to the use of the sitemap. Participant 6 noticed, while viewing the sitemap and legend, that login-protected pages should appear in red. Since his “offerride” page did *not* appear in red he correctly

concluded that it still needs to be declared as login-protected. Briefly thereafter, he set up the access restrictions.

Table 27: Participants' ratings on frequency of use and usefulness of sitemap

Statement from questionnaire and scale	1	2	3	4	5	N/A
I looked at the Sitemap (1=hardly ever, 5=very often)	2 (P3,5)	3 (P2,4,6)	1 (P1)	-	-	-
When I did look at the Sitemap I found it useful. (1=strongly disagree, 5=strongly agree)	2 (P2,3)	-	1 (P4)	1 (P6)	1 (P1)	1 (P5)

In conclusion, the sitemap does not appear to provide a clear benefit for the development of basic web applications.

However, I believe that the sitemap would become more useful after repeated use and for the development of applications with requirements that are less understood. Improvements to the tutorial may better explain the purpose and the meanings of the visualizations. Furthermore, the visualization could be refined in a number of ways; one being the use of miniature renditions of the actual page layouts instead of abstract rectangles. Also, it seems likely that developers benefit more from the sitemap when developing more complex applications that need to be verified for functional coverage. However, there is also a limit to the scalability of the sitemap, because the visualization quickly becomes cluttered with a growing number of pages and relationships (a selective on/off feature may mitigate this situation).

Finally, the sitemap may be used more often if it could be displayed along another view, such as the *Develop* view. If these two views can be displayed simultaneously (similar to the split view in Macromedia Dreamweaver), perhaps highlighting and animation could be used to visualize the current state of the application. Displaying the sitemap along with another view has the obvious downside of competition for screen real estate. Large displays or dual-monitor setups would certainly be a solution but (in the short term) seem to be unrealistic for nonprofessional web developers.

6.2.7 Results on the Problem of Security

The major web development problem identified by experienced web developers are the difficulties related to implementing secure applications (see 3.1.2.1). Whether or not an end-user developer is aware of the risks involving web applications, these risks are real and need to be mitigated if EUDWeb is to become a reality. During the studies of end-user developers' mental models I found that end-user developers think about security just in terms of surface features and are (not surprisingly) unaware of specific risks "behind the scenes" such as SQL injection or cross-site scripting.

Click's design abstracts security. The visible part of the security layer allows the developer to specify the authentication and authorization parameters on a high level of abstraction. The invisible part of the security address the risks "behinds the scenes".

The following sections discuss to what extent *selected* Click features succeed in resolving the problem of developing secure applications.

6.2.7.1 High-level Authentication Features

At the onset of the summative evaluation, I hypothesized that: *End-user developers understand how to set up "page access restrictions"*. Four of the six participants encountered one or more problems while defining access restrictions for the *Offer ride* page. Click has two basic concepts related to access restrictions. First, in its properties dialog a page can be declared as "requiring login." Only two out of the six participants immediately noticed this option.

Second, every application has at least one page that is defined as the "login page" for the application. Click will automatically redirect users to the login page if a page requires login. This login page contains a login box component. Two of the participants did not notice the login box component and began to manually create a login form by combining two text input fields and a button component. Participant 4 tried to place a login box component directly on the *Offer ride* page and could not recover from this problem, which lead to a critical incident. Finally, when a new application is created, Click automatically sets up a "homepage" and a "loginpage", a feature that three of the participants did not expect. Consequently they created redundant login pages.

In one case we observed a participant exploring the options in the property dialog of the *Text, Html, Links* component, apparently in an attempt to find an option to define login restrictions for the *Offer ride* hyperlink. This particular behavior indicates once again, that end-user developers primarily perceive the surface features of security functions.

Despite the numerous problems that the participants encountered, Click's access restriction functionality appeared to "make sense" once the participants had discovered them. The post-questionnaire supports this argument. Four participants strongly agreed (and two participants agreed) to the statement: "*Now I understand how to set up user-login controls (e.g. the login-protected "Offer rides" screen). (1=strongly disagree, 5=strongly agree)*".

In conclusion, although five out of the six participants were able to correctly implement login restrictions without assistance (but with considerable effort in most cases), our "page-level access restriction" concept needs to be refined. I believe that the overall approach of specifying restrictions as page properties is appropriate and easily understood. However, the specifics of the login box and its interaction with protected pages need to be better communicated to the user. One step in this direction would be to place help information right within the properties dialog of the login box component. Currently the login box properties provide no help at all and only display the component-ID of the login box which is of little consequence to the developer. Furthermore, an EUDWeb tool may offer a hint when the developer creates more than one page containing a login box component.

6.2.7.2 Authorization via Concept of "Record Owned By User"

At the onset of the summative evaluation, I hypothesized that: *End-user developers understand the concept of a data record that is "owned" by a user.* Participants appeared to understand the essence of the concept. While implementing the data table on the *Home* page, in almost all cases, they chose the appropriate option in the dynamic table dialog to only show "edit" and "delete" links for the records that belong to the currently logged in user. However, it is unclear if participants were aware of *how*

Click determines whether or not a particular record is owned by a particular user (Click automatically maintains a “lastmodifiedbyuserid” field for each data record). In fact, in a few cases participants were confused because, although they had chosen the option, “edit” and “delete” links would still appear even when nobody was logged in. The situation was established when they had added data as “anonymous” users either directly in the *Database* view or through the *Offer ride* screen while it did not yet require login. Click always displays “edit” and “delete” links for anonymous data records if these links are enabled at all. Perhaps an additional option for controlling the behavior for anonymous data records may improve Click’s usability.

Four participants strongly agreed (and two participants agreed) to the statement: “*Now I understand the following option of the Dynamic table component: Show edit link only for data records "owned" by "currently logged in user". (1=strongly disagree, 5=strongly agree)*”.

In conclusion, the concept “record owned by user” seems to be appropriate for end-user developers.

6.2.8 Results on the Problem of Feedback

The desire for speedy tools and the support for short develop-test cycles was a reoccurring theme within my survey and interview studies of experienced and novice developers (e.g., 3.1.1.2).

All of the EUDWeb prototype tools we have developed employed the concept of design-at-runtime (see 5.1) which was designed to accelerate the develop-test-cycle by minimizing mode switching. The sections below describe how this concept was used and perceived during the summative study.

At the onset of the summative evaluation, I hypothesized that: *End-user developers are comfortable with and frequently use the runtime feature of the “design-at-runtime” concept.* Most participants frequently used the testing feature of the design-at-runtime concept – particularly in the later stages of development. Figure 26 shows a visualization of P6’s behavior during the study as it is prototypical for the whole group of participants (see Appendix G.10 for color plates showing the visualizations of all

participants' behavior). The visualized timeline is derived from the automated activity log and displays how participant 6 spends time in particular “modes”, i.e. “Develop” (1st line, black), “Test in develop” (2nd line, green), “Test in preview” (3rd line, green), “Read help” (4th line, blue), “Critical incident” (5th line, red), “View example” (6th line, gray). Note that because of space economy the visualized timeline is split into multiple segments. Only participant 1 and P3 used the explicit *Preview* function more than testing directly in the *Develop* view. P1’s behavior can be explained with a bug in Click that made the design-at-runtime not work as expected (P1 repeatedly tried clicking on links but instead of being sent to the link target, the properties dialog opened; after a couple of attempts P1 reverted to using the *Preview* function). No other participant encountered this bug. However, we observed that participants occasionally opened the properties dialog when they had intended to perform a link or button action. This was due to targeting problems and may be improved by extending the distance between the actual component rendition and the “move” icon and perhaps by limiting the “open properties dialog” behavior to clicking on the “move” icon only.

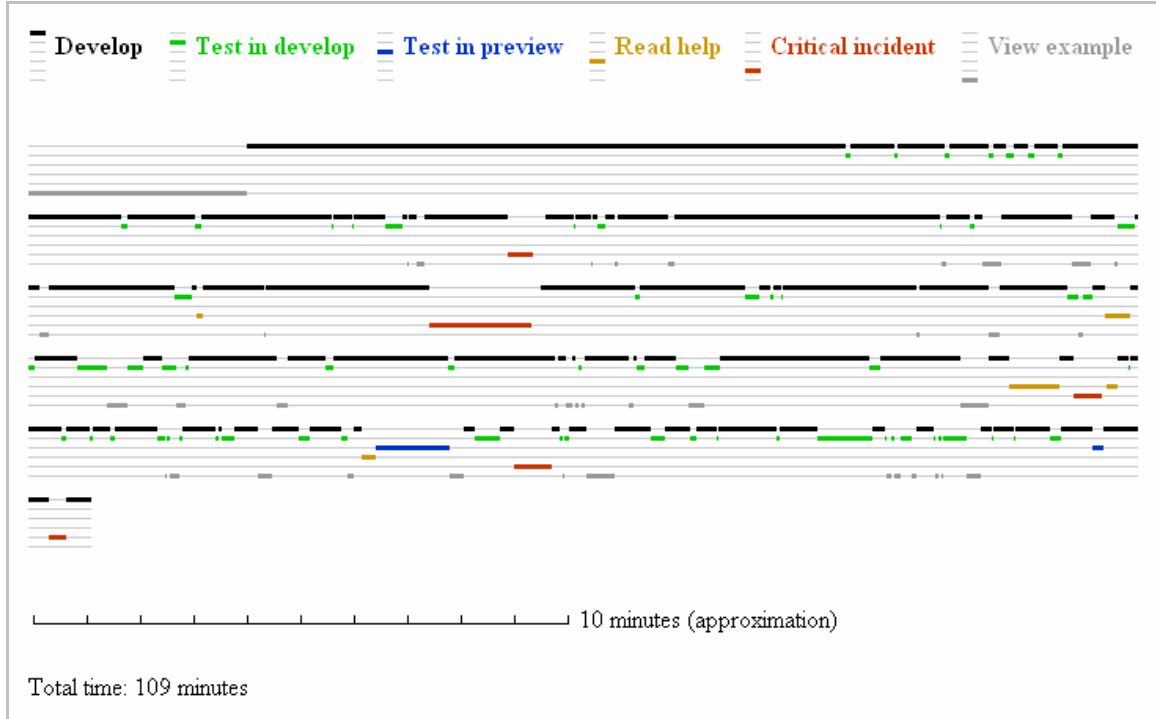


Figure 26: Visualized timeline of participant 6’s behavior as derived from the activity log

Although the design-at-runtime concept was just briefly introduced in the introduction tutorial video, all participants (but P1) used it without encountering major problems. When P4 asked: “*I can test from right here, right?*” and we (intentionally) did not answer, he used it correctly.

All six participants strongly agreed with the statement “*It was convenient to be able to both edit and test components within the same screen (i.e. the Develop view). (1=strongly disagree, 5=strongly agree)*”. When asked “*Was there anything you did NOT like about switching between editing and testing? Any ideas how it could be improved?*” in the post-study questionnaire only participant 5 raised a concern (“*Sometimes I'd click on the edit button when I meant to test. Maybe the edit buttons could be a bit further away from the integrated functionality?*”). The other four comments were all very encouraging in nature (P2 did not reply which may be interpreted as a positive answer):

- P1: “It seemed pretty seamless to me.”
- P3: “No.”
- P4: “I liked it very much.”
- P6: “Actually, that was the most useful aspect.”

Three of the six participants even referred to the design-at-runtime feature when asked about their three favorite aspects of Click (“*What are the top 3 aspects of the Click tool you like?*”).

In conclusion, the design-at-runtime concept seems to be very successful and well liked by end-user developers. EUDWeb tools may use this concept to facilitate a faster develop-test cycle.

6.2.9 Critical Incidents and General Observations

While the study participants replicated the functionality of the example ride board application using Click, we noted the problems that arose, particular focusing on *critical incidents* – problems that the participants experienced as severe impediments to their work. At the onset of the study participants were instructed to work autonomously and only ask for help if they felt “completely lost”. These are the episodes that we logged as *critical incidents*. Due to time and other practical constraints of the study, we could not verify that all of the problems logged as critical incidents would be insurmountable in real-life situations. It appears likely that many of the problems may be overcome if the participant had spent additional time exploring options or referring to help information. Nonetheless, at a minimum each critical incident stands for a usability problem that needs to be addressed by redesign or improvements to the help materials. The following list contains all the critical incidents that occurred during the study; the markers in parenthesis indicate which participant encountered the particular problem. Next to each critical incident I offer ideas of how the issue could be addressed.

The participant...

- a) does not notice the component property “show on every page” while trying to replicate identical navigation links on every page (P1, P4, P6)
 - *potential fix*: Concept of header and footer or layout templates may be more natural (however “show on every page” was well understood when discovered)
- b) creates components that belong to two different screens on one page; initially does not know that multiple pages should be created (P3)
 - *potential fix*: Help, tutorial (participant appears to imagine *one* page with changing contents, rather than multiple pages)
- c) does not know how to link to a page that does not yet exist (P3)
 - *potential fix*: Offer “add new page” option within the link dialog

- d) connects the “Offer ride” link to the wrong page (i.e. *Search*); later notices the faulty behavior but does not find the source of the problem (P3)
 - *potential fix*: Tool tip or other mechanism could indicate link target page
- e) enters a search keyword; then adds the “clear input fields” action to the search button but is confused that the field does not immediately clear; participant does not notice she needs to click the button to initiate the action (P2)
 - *potential fix*: When “clear fields” action is selected, run it immediately
- f) does not know how to make the search work (does not use the search wizard, or not notice/understand the “filter” properties of the table component) (P1, P4)
 - *potential fix*: Offer a “Search” action for buttons (many participants looked for such an option); Make wizards more prominent
- g) attempts to implement the table on the *Home* page using static and dynamic text components (P6)
 - *potential fix*: Help, tutorial; Tool tip or help icon next to each component in the library that describe and exemplify the purpose
- h) does not know how to implement the *Details* page (P3)
 - *potential fix*: Help, tutorial; Make wizards more prominent
- i) creates *Details* page with dynamic text component but does not know how to link the rows of the table to the *Details* page (P6)
 - *potential fix*: Help, tutorial; Show example of connection between dynamic table and text in property dialog of dynamic text (the relationship between dynamic table and dynamic text was unclear for this participant)
- j) uses text field place holders instead of database field place-holders (in the dynamic text component) for implementing the *Details* page; is confused about the difference (P1,P3)
 - *potential fix*: Redesign dynamic text dialog to better distinguish the two
- k) uses the Overview/details wizard to create a dynamic text component that only shows the “destination”; then calls the same wizard again to create the “departuredate”; this produces not only a second dynamic text component but also a second table components and results in confusion (P4)

- *potential fix*: Clarify the role of wizards as *creators* of sets of components
- l) confused that the dynamic text component does not automatically update after the participant creates more database fields (P4)
 - *potential fix*: Help, tutorial
- m) clicks on a link in the To-do list to create a new dynamic text component but Click displays low-level, unintelligible error message because the participant had previously deleted the target page (P4)
 - *potential fix*: Offer user-friendly error message
- n) does not understand that the table is still showing “edit” & “delete” links although nobody is logged in; participant had previously entered some data records while not being logged in (P6)
 - *potential fix*: Help, tutorial; Offer additional option that asks the developer whether or not entries submitted by anonymous persons should be editable
- o) does not know that custom database fields need to be created (despite a notice in the To-do list) before wizards can be used; does not understand the error message in the wizard (P3)
 - *potential fix*: Help, tutorial; Improve error message in wizard
- p) creates “departure date” database field as type “number” instead of type “text”; consequently all entered dates are truncated but participant does not notice the source of the problem (P2, P3)
 - *potential fix*: Offer pre-defined “date” data type; Display pop-up warning message to developer if entered test data does not fit the type
- q) does not know how to proceed after creating the database fields “destination” and “departuredat” in the *Database* view (P4)
 - *potential fix*: Help, tutorial
- r) uses database entry form wizard to create the “Offer ride” screen but at that time only the fields “destination” and “departuredat” exist in the database; does not know that more database fields should be created (P3)
 - *potential fix*: Help, tutorial; critical incident may be due to limited knowledge of requirements which may not happen in real life

- s) uses database entry form wizard to create the “Offer ride” screen (at that time only the fields “destination” and “departure date” exist in the DB); creates more database fields (name, email, comments); is confused that there are no corresponding text fields for the recently created database fields (P3)
 - *potential fix*: Help, tutorial
- t) attempts to create login form manually using input text field components (P3)
 - *potential fix*: Help, tutorial
- u) creates a login box component on the *Offer ride* page (confused about how to implement the login-restriction for the *Offer ride* page (P4)
 - *potential fix*: Help, tutorial; Prominent clue for private *and* public pages
- v) does not find the link to publish the application (P1, P4, P6)
 - *potential fix*: Offer publish option (along with other application-level functions such as “rename” and “edit developers”) in *Develop* view

Many of the critical incidents we observed can be addressed through minor changes to Click’s user interface, in particular, improved help, tutorial, and error messages, and the use of tool tips. We have observed that the *exact choice of wording* for error messages and To-do list items strongly influences the developer’s success. Often, potentially helpful messages were ultimately disregarded because the developer did not understand them in the brief amount of attention they were given. None of the critical incidents stands out as a frequent problem, with the exception of (a) and (v) which are usability hurdles that can be overcome by minor redesign. No other type of critical incident occurred for more than two of the six participants.

However, still a few of the observed critical incidents revealed considerable mismatches between the participants’ expectations (their “natural” mental models) and Click’s features. In particular, the implementation of the search function, and also the exact role of place-holders in the dynamic text component seem unintuitive. Most participants expected “search” to be an action tied to a button – a concept that EUDWeb tools like Click should adopt. An alternative to Click’s place-holder concept is less

apparent. Possibly, a redesign of the dynamic text component, which clarifies the roles of input text place-holders and database field place-holders, may address the problem.

Finally, many of the critical incidents may be circumvented if developers would more often take advantage of Click's wizards. However, we frequently observed that participants would manually implement a particular function instead of relying on the more efficient and less error-prone wizards. In one case, a participant (P6) remarked that he had initially mistaken the role of wizards as pure help tools rather than devices that automatically create a related set of components.

Apart from the critical incidents that triggered participants to "surrender" and ask for help, we observed a number of small usability problems. For example, many participants:

- attempted to reach a context-menu by right-clicking on components (a concept not (yet) available in Click),
- worked hard to discover how to "delete" components,
- were unaware of the necessity for highlighting text before creating a link,
- tried to deselect components by clicking on the background (Click highlights the currently active component),
- struggled to discover how to increase the size of the "comments" text box,
- used upper-case initials and spaces when naming components and database fields (however, the resulting error messages helped them quickly to recover),
- tried to edit the data within a record by clicking directly on an empty cell in the Database view (perhaps a concept known from spreadsheet applications).

Last but not least, the study exposed a small number of bugs (occasionally broken preview function in the property dialog of the dynamic table component) and missing functionality such as undo/redo features, support for multiple-component selection and alignment tools.

We frequently observed that when participants encountered difficulties they tried to work on the problem for a little while and then abandoned it to make progress on an unrelated feature, returning to the problem at a later point in time.

On a more general note, we found that all six participants thought they were finished before they actually were; nobody replicated the functionality without being pointed at missing features. However, I believe that this issue reflects the somewhat unrealistic aspect of my approach of *replicating* an already existing application; therefore, this observation may be of little consequence for real-life development.

6.3 Summary and Conclusions

Table 28 summarizes the findings from the formative studies and summative evaluation of Click. All statements in the table relate to end-user developers.

Table 28: Results of Click’s formative studies and summative evaluation

Conclusions for Future End-User Web Development Tools
<ul style="list-style-type: none"> • Separating layout and behavior definition into two tools is problematic; an EUDWeb tool should integrate those aspects (6.1.1) • Monolithic tool with lack of code editing feature is likely to be too inflexible for EUDWeb (6.1.2) • Concept of <i>basic</i> button action rules is an intuitive concept (6.2.5.1) • Current implementation of action rules are <i>not</i> a good match for advanced conditions (branching) (6.1.2; Table 18 on page 145) • As an example of a high-level component, the dynamic table is a good abstraction for listing data, providing edit and delete functionality and linking individual records to a details page (6.2.5.2) • Exposing input validation as properties of associated input fields matches expectations (6.2.5.3) • “Persistence-by-default” matches natural mental model <i>but</i> exceptions exists (6.2.5.4) • Templating/place-holder feature of the dynamic text component is problematic (6.2.5.5) • High integration between layout, behavior, database, testing supports seamless workflow (6.2.6.1) • To-do list feature with its current (limited) level of sophistication provides no clear advantage; major problems are clarity of messages, and context-relevancy (6.2.6.2) • Sitemap provides no clear benefit; may be due to limited complexity of the study problem (6.2.6.3) • Overall approach of specifying restrictions as page properties is easily understood; actual implementation is problematic (6.2.7.1) • Concept “record owned by user” is understood; the inner workings likely not (6.2.7.2) • Design-at-Runtime concept works well and is rated highly (6.2.8)

Overall, the summative evaluation has shown that the Click prototype tool comes close to meeting my vision for an end-user web application development tool; it does enable nonprogrammers to create basic database-driven web applications in a short amount of time.

Although it is difficult – in the absence of extensive experiments – to attribute the participants’ success to particular features, I mainly credit the developers’ success to these aspects of Click design:

- *high-level components and concepts,*
- *level of tool integration* (layout, behavior, database, and hosting), and the
- *proximity of Click’s concepts to end users’ “natural” mental models.*

In combination, these three design features/approaches address the main entry barrier to web application development – the problem of complexity. Furthermore, the high-level components and automatic code generation features (potentially) solve security and cross-platform compatibility problems. Click is an example for an EUDWeb tool that hides most security-related and cross-platform-related problems by integrating routines and filters for input validation and by only producing cross-platform compatible code (note that Click does not yet fully implement these features).

Although I have not investigated new approaches to handling the debugging problem (3.1.2.4), it is likely that the users of a high-level EUDWeb tool similar to Click will encounter fewer *low-level* errors (until they begin to take advantage of the extensibility layer that allows custom low-level code).

The following and final Chapter 7 summarizes my research findings, highlights the contributions, and outlines possible directions for future research in the area of end-user web application development.



Chapter 7

Conclusions and Future Work

7.1 Summary of Findings

This research has explored many different aspects of how nonprogrammers can be empowered to develop basic web applications. However, I began the work reported here with three general research questions (see 1.4) which I now return to in summarizing my overall research findings.

7.1.1 What are the main entry barriers to EUDWeb?

From an end-user development perspective, the arena of web programming seems exceptionally challenging because even experienced programmers encounter many barriers in their daily work. Section 3.4 summarizes these problems. Perhaps, the main technical obstacles are cross-platform compatibility issues, the need for integration of numerous diverse technologies, ensuring security, and the process of debugging.

Overall, from an end user's point of view, *web application development is simply too complex*, involving too many concepts, technologies, and relationships. Current tools that are targeted at end-user developers *lack a holistic approach* towards supporting developers from start (requirements phase) to finish (publishing and maintenance).

Finally, my studies of end users' natural mental models of web programming concepts (see Chapter 4) have shown that there is a considerable mismatch between novices' expectations and the state-of-the-art in web development. This mismatch creates further entry barriers to EUDWeb.

7.1.2 How do novice developers naturally think about web programming concepts?

When I set out to explore how nonprogrammers “naturally” think about the inner workings of web applications (without having prior exposure to the underlying technology), I was hopeful to uncover a number of mental models which could be used to implement tools that feel more intuitive by reducing Norman’s (1986) “gulf of evaluation” and “gulf of execution”. The reality, however, is simpler. Most of the participants of my two “mental models” studies (see Chapter 4) did not have any pre-conceived notions and often even struggled to develop deeper mental representations of the functionality they were asked to analyze. Although, this may seem like a “non-result” it underlines the level of support and guidance end-user developers require.

The study participants’ descriptions were high-level only, both, when given the freedom to choose the level of abstraction (MMODELS-1, see 4.1) and when asked to be as detailed as possible (MMODELS-2, see 4.2). Participants showed a good knowledge of the terminology of the web development arena, although they frequently used words like database or field in a nonspecific or imprecise way. They generally used a mix of constraints (e.g., “this field is required”) and rules (e.g., “If the password is incorrect, that field is cleared”) to describe certain functionality, without paying attention to order or flow of control. They expect functions such as search and overview/detail relationships to be available as basic components. Only few participants showed any interest or awareness of implementation details for basic services such as session management, database connection, input validation, or security checking. However, their descriptions indicate certain expectations such that data and state persist until they are explicitly changed which were often at odds with the technical implementation (e.g., HTTP’s stateless nature). Section 4.3 summarizes the expectations end-user developers are likely to have regarding concepts frequently found in web applications.

In order to facilitate EUDWeb, the natural mental models of end users should be taken into account and tools and concepts be reshaped in order to make a better fit for users’ expectations and skills.

7.1.3 What are viable approaches for making web application development more accessible for nonprogrammers?

Above all, end-user developers need tools that abstract the complexity of the web application technologies. It is unrealistic at best, to turn nonprogrammers quickly into skilled web developers. Furthermore, because of the special requirements for availability, compatibility, accessibility and security, it seems unwise to advocate the use of professionals' tools for end-user developers. The analysis of common problems in web application development (see 3.4) has shown that, what end-user developers need more than anything, are *integrated tools that hide complexity and guide the developer from start to finish*. Database schema design, layout design, graphics design, business logic programming, cross-browser compatibility and accessibility testing, and even the process of publishing and production hosting, should no longer be regarded separate phases that are supported by different tools but rather be integrated into one tool.

This tool should offer its functions at a *level of abstraction and mode of operation that nonprogrammers expect* (see 7.1.2), for example by providing a set of pre-defined high-level components or by offering built-in session management with persistence-by-default. Furthermore, the developers' tendencies to opportunistic behavior should be embraced by avoiding "premature commitment" (Green 1989) and allowing changes to the design at any point in time without penalty.

Finally, it is important to consider that as end-user developers learn, their needs and objectives are likely to grow along with their knowledge. A tool that only offered high-level functionality would quickly become obsolete and merely postpone the point when nonprogrammers encounter classical programming concepts. I support the idea of tools that expose a "*gentle slope of complexity*" (MacLean, Carter et al. 1990) by providing *multiple levels of programming support*, reaching from functions that allow novices the customization of template applications, over configurable high-level components, up to providing access to a full featured low-level programming language (see 5.4.25).

We have developed Click as a proof-of-concept prototype that addresses most of these requirements, even though not yet to the full extent (e.g., graphics design is not yet an integral part, and the “gentle slope” concept not fully implemented). Section 5.6 summarizes Click’s contributions.

7.2 Summary of Research Contributions

I have described the initial phases of a user-centered approach towards understanding and supporting end-user development of web applications. From investigating end users’ needs I have found that basic data collection, storage and retrieval applications such as surveys, registration systems, service forms, or database-driven websites are an important target for end-user development. While focusing on this particular domain and on casual (nonprogrammer) web developers as my target audience, I have made the following contributions to the fields of end-user development and web engineering:

- An analysis of end-users’ needs and opportunities for EUDWeb (3.1.1.1),
- A summary of barriers to and recommendations for EUDWeb (3.4),
- An analysis of experiences and behavior of semi-professional web developers (3),
- An analysis of the mental models and strategies of end-user developers (4),
- The “design-at-runtime” concept in the realm of web-based applications (5.1),
- An analysis of the elements of typical web applications (3.3),
- The design rationale, prototype, and evaluation of a EUDWeb tool (5.3-6.3),
- A conceptual framework and partial prototype implementation for exposing a “gentle slope of complexity” for EUDWeb tools (5.4.25).

7.3 Future Directions

Much work needs to be done to refine the analysis of what “feels most natural” to end-user developers (which is likely to change with time) and to identify barriers to EUDWeb. For example, a novice-friendly yet powerful concept is needed to help end-user developers define complex conditions and actions – as discussed in Section 6.2.5.1, a promising alternative to the currently used set of independent rules are multi-way if-statements in the format `if-elseif-elseif...else`.

Apart from this evolutionary work, I feel that three research areas in particular deserve close attention – one being collaborative aspects of EUDWeb, another the problem of sustainability and evolution, and the third the application of artificial intelligence concepts.

The first topic recognizes that end-user development rarely takes place in a vacuum; that there are typically a number of people involved, ranging from the future users to professional IT staff who serve the roles of consultants, partners or programmers of particularly challenging components of the application in development. Nardi (1993) has recognized the need for investigating and considering collaborative aspects of end-user programming long ago and this argumentation certainly also applies to the domain of web application development.

The second topic is at the heart of end-user development itself. The central problem of empowering end users is finding a good balance of ease-of-use and flexibility. Naturally, less flexible tools, not only limit the power but also the number of problems a developer may encounter, and are thereby easier to use than more powerful tools. However, as users of these tools learn over time, not only their knowledge but also their aspirations evolve. MacLean et al. (1990) and Repenning and Ioannidou (1997) have argued for supporting a “gentle slope of complexity” – tools that “grow” with the skills and needs of their users. I have shown how this problem may be addressed for EUDWeb by providing different layers of programming support (see 5.4.25). However, much future work is required to fully implement, evaluate, and evolve these ideas.

Third, technology has opened new horizons for approaches that could revolutionize application development. I believe that many new opportunities (and challenges) lie within the field of artificial intelligence. If computers can begin to “understand” what developers *mean* instead of just blindly responding to what they are explicitly told, application development may become easier. The path to true natural language programming – as exemplified by the “magical machine” metaphor used in my first study (see 4.1.1) – seems long. Also, the general superiority of this approach is still far from being proven. However, even basic mixed-initiative development environments may have a profound impact on usability and power. For a start, such systems could automatically recognize under-specification and query the developer for more details when needed. Click’s To-do list feature (see 5.4.5, 6.2.6.2) shows a *very early* attempt of applying this idea, although it still exhibits most of the shortcomings of computer-initiative systems and few of the benefits. The future may bring systems that hold true conversations with the developer in a similar way to how developers hold conversations with their clients today.

Finally, basic database-driven web applications, which were the focus of this research, are only a part of the “big picture” of web sites and web-based applications. The work reported here needs to be integrated with other aspects of web development and web publishing such as the problem of web content management (the distributed authoring of a web site), web portals, or e-Commerce web sites. A complementary approach to component-based web development tools such as Click are tailorable web applications. As observed during the initial requirements analysis (3.1.1.1), a large fraction of webmasters’ needs could be satisfied through customization of generic web applications (e.g., calendar, resource reservation, message board, content management, e-commerce). Tailorable systems should be built on top of a flexible framework that allows advanced custom changes similar to the application templates offered by Click (5.4.2).

Much work needs to be done before we can claim that end-user web application development is a reality. The research I have presented here is one early step into the promising future of end-user web application development and I hope that other work will follow.

References

- Adelson, B. and E. Soloway (1985). "The role of domain experience in software design." IEEE Transactions on Software Engineering SE-11: 233-242.
- Adobe (2003). GoLive. <http://www.adobe.com/products/golive/>
- Ambler, A. and J. Leopold (1998). Public Programming in a Web World. Proceedings of IEEE Symposium on Visual Languages, Nova Scotia, Canada: 100-107. Sep. 1-9, 1998
- Apache Software Foundation (2005). Velocity Template Engine - an Apache Jakarta Project. <http://jakarta.apache.org/velocity/>
- Apple (1987). HyperCard User's Guide.
- Balthaser:Fx (2005). <http://www.balthaser.com/>
- Bare Bones Software (2003). BBEdit. <http://www.barebones.com/products/bbedit/>
- Berners-Lee, T. (1996). WWW: past, present, and future. IEEE Computer. 29: 69-77.
- Bhardwaj, Y. (2005). Reverse Engineering End-User Developed Web Applications into a Model-based Framework, Virginia Tech.
- Blackwell, A. F. (1996). Metacognitive theories of visual programming: What do we think we are doing? Proceedings of IEEE Visual Languages: 240-246.
- Brabrand, C., A. Moeller and M. I. Schwartzbach (2002). "The <bigwig> project." ACM Transactions on Internet Technology 2(2): 79-114.
- Brooks, R. (1983). "Towards a theory of the comprehension of computer programs." International Journal of Man-Machine Studies 18: 543-554.
- Brown, P. S. and J. D. Gould (1987). "An experimental study of people creating spreadsheets." ACM Transactions on Office Information Systems 5(3): 258-272.
- Burnett, M., J. Atwood, R. Djang, H. Gottfried, J. Reichwein and S. Yang (2001). "Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm." Journal of Functional Programming 11(2): 155-206.

- Burnett, M., S. K. Chekka and R. Pandey (2001). FAR: An End-User Language to Support Cottage E-Services. Proceedings of HCC - 2001 IEEE Symposia on Human-Centric Computing Languages and Environments; Symposium on End-User Programming, Stresa, Italy: 195-202.
- Burnett, M., B. Ren, A. Ko, C. Cook and G. Rothermel (2001). Visually testing recursive programs in spreadsheet languages. Proceedings of IEEE Symposium on Human-Centric Computing: HCC 2001, New York, IEEE: 288-295.
- Carroll, J. M. (1990). The Nurnberg Funnel: Minimalist Instruction for Computer Skill. Cambridge, MA, MIT Press.
- Carroll, J. M. (2000). Making use: Scenario-based design of human-computer interactions. Cambridge, MA, MIT Press.
- Carroll, J. M., J. C. Thomas and A. Malhotra (1979). "Clinical-experimental analysis of design problem solving." Design Studies 1: 84-92.
- Ceri, S., P. Fraternali and A. Bongio (2000). "Web Modeling Language (WebML): a modeling language for designing Web sites." Computer Networks 33(1-6): 137-157.
- Codejay (2003). Codejay. <http://www.codejay.com/>
- curl (2005). Curl. <http://www.curl.com/>
- Deshpande, Y. and S. Hansen (2001). "Web Engineering: Creating a Discipline among Disciplines." IEEE MultiMedia 8(2): 82-87.
- Dijkstra, E. (1968). "GOTO considered harmful." Communications of the ACM 11(3): 147-148.
- Eclipse (2005). Eclipse. <http://www.eclipse.org>
- Eisenberg, M. (1995). "Programmable applications: Interpreter meets interface." SIGCHI Bulletin 27(2): 69-93.
- FileMaker (2005). FileMaker Pro. <http://www.filemaker.com/>
- Fischer, G. and A. C. Lemke (1988). "Construction kits and design environments: Steps toward human problem-domain communication." Human-Computer Interaction: 179-222.
- Fraternali, P. (1999). "Tools and Approaches for Developing Data-Intensive Web Applications: A Survey." ACM Computing Surveys 31(3): 227-263.

Fraternali, P. and P. Paolini (2000). Model-driven development of Web applications: the AutoWeb system. Proceedings of ACM Transactions on Information Systems (TOIS): 323-382. October 2000

Gaedke, M., D. Schempf and H.-W. Gellersen (2000). WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. Proceedings of 2000 ACM Symposium on Applied Computing (SAC 2000), Villa Olmo, Como, Italy: 748-755. March 19-21, 2000

Garrett, J. J. (2005). Ajax: A New Approach to Web Applications.
<http://www.adaptivepath.com/publications/essays/archives/000385.php>

Gilmore, D. J., K. Pheasey, J. Underwood and G. Underwood (1995). Learning graphical programming: An evaluation of KidSim. Proceedings of Interact'95, Amsterdam, Netherlands. September 1995

Ginige, A. and S. Murugesan (2001). "Web Engineering: An Introduction." IEEE MultiMedia January-March: 14-18.

Green, T. R. G. (1989). Cognitive dimensions of notations. People and Computers IV. Cambridge, Cambridge University Press.

Green, T. R. G. (1990). Programming languages as information structures. In Psychology of Programming. London, Academic Press: 117-138.

Green, T. R. G. and M. Petre (1996). "Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework." Journal of Visual Languages and Computing 7: 131-174.

Guindon, R. (1990). "Designing the design process: Exploiting opportunistic thoughts." Human-Computer Interaction 5: 305-344.

Harrison, W. (2004). The Dangers of End-User Programming. IEEE Software. 21: 5-7.

Helman, T. and K. Fertalj (2003). A Critique of Web Application Generators. Proceedings of Information Technology Interfaces (ITI), June 16-19, 2003, Cavtat, Croatia.

Homestead (2005). <http://www.homestead.com/>

Hostetter, M., D. Kranz, C. Seed, C. Terman and S. Ward (1997). "Curl: A Gentle Slope Language for the Web." WWW Journal II(2).

IBM (2005a). IBM Rational Web Developer for WebSphere Software.
<http://www.ibm.com/software/awdtools/developer/web/>

IBM (2005b). Integrated Development Environment for Laszlo.

<http://www.alphaworks.ibm.com/tech/ide4laszlo>

Instantis (2003). SiteWand. <http://www.instantis.com/>

Kernighan, B. W. and D. M. Ritchie (1978). The C Programming Language, Prentice Hall, Inc.

Lang, M. and B. Fitzgerald (2005). "Hypermedia Systems Development Practices: A Survey." IEEE Software 22(2): 68-75.

Laszlo Systems Inc. (2005). OpenLaszlo. <http://www.openlaszlo.org/>

Lerdorf, R., A. Gutmans and Z. Suraski (1995). History of PHP and related projects.

<http://www.php.net/manual/en/history.php>

Lewis, C. (1982). Using the "thinking-aloud" method in cognitive interface design. Technical Report RC9265. Watson Research Center. Yorktown Heights, NY

Lieberman, H. (2001). Your wish is my command: Programming by example. San Francisco, CA, USA, Morgan Kaufmann.

Loureiro, N. (2002). Programming PHP with Security in Mind. LINUX Journal.

<http://www.linuxjournal.com/article/6061>

MacLean, A., K. Carter, L. Lövstrand and T. Moran (1990). User-Tailorable Systems: Pressing Issues with Buttons. Proceedings of ACM CHI 1990: 175-182.

Macromedia (2002a). ColdFusion. <http://www.macromedia.com/software/coldfusion/>

Macromedia (2002b). Flash MX. <http://www.macromedia.com/software/flash/>

Macromedia (2002c). Flash Player Version Penetration.

http://www.macromedia.com/software/player_census/flashplayer/version_penetration.html

Macromedia (2003). Homesite. <http://www.macromedia.com/software/homesite/>

Macromedia (2005a). Contribute. <http://www.macromedia.com/software/contribute/>

Macromedia (2005b). Dreamweaver.

<http://www.macromedia.com/software/dreamweaver/>

Macromedia (2005c). Flex. <http://www.macromedia.com/software/flex/>

- McDaniel, R. (2001). Demonstrating the Hidden Features that Make an Application Work. Your Wish is My Command: Programming By Example. H. Lieberman. San Francisco, CA, USA, Morgan Kaufmann: 163-174.
- Mecca, G., P. Merialdo, P. Atzeni and V. Crescenzi (1999). The (Short) Araneus Guide to Web-Site Development. Proceedings of Second International Workshop on the Web and Databases (WebDB'99). May 1999
- Merialdo, P., P. Atzeni, M. Magnante, G. Mecca and M. Pecorone (2000). HOMER: a Model-Based CASE tool for Data-Intensive Web Sites. Proceedings of Exhibition Section of SIGMOD'2000, May 2000.
- Microsoft (2002). ASP.NET. <http://www.asp.net/>
- Microsoft (2003a). Visual Studio. <http://msdn.microsoft.com/vstudio/>
- Microsoft (2003b). Web Matrix. <http://www.asp.net/webmatrix/>
- Microsoft (2004). Inside XAML.
<http://www.ondotnet.com/pub/a/dotnet/2004/01/19/longhorn.html>
- Microsoft (2005a). FrontPage. <http://www.microsoft.com/frontpage/>
- Microsoft (2005b). Visual Web Developer. <http://lab.msdn.microsoft.com/express/vwd/>
- Miller, L. A. (1974). "Programming by non-programmers." International Journal of Man-Machine Studies 6(2): 237-260.
- Miller, L. A. (1981). "Natural language programming: Styles, strategies, and contrasts." IBM Systems Journal 20(2): 184-215.
- Mozilla (2005). XML User Interface Language (XUL).
<http://www.mozilla.org/projects/xul/>
- Myers, B. A. and R. McDaniel (2001). Demonstrational Interfaces: Sometimes You Need a Little Intelligence, Sometimes You Need a Lot. Your Wish is My Command: Programming By Example. H. Lieberman. San Francisco, CA, USA, Morgan Kaufmann: 45-58.
- MySQL (2005). MySQL. <http://www.mysql.com>
- Nardi, B. A. (1993). A Small Matter of Programming: Perspectives on End User Computing. Cambridge, MA, MIT Press.

- Nardi, B. A. and J. R. Miller (1991). "Twinkling lights and nested loops: distributed problem solving and spreadsheet development." International Journal of Man-Machine Studies 34: 161-184.
- Newman, M., Landay, J. (2000). Sitemaps, storyboards, and specifications: a sketch of Web site design practice. Proceedings of Conference on Designing interactive systems: processes, practices, methods, and techniques, New York City, New York, United States: 263-274. August 17-19, 2000
- Newman, M., J. Lin, J. Hong and J. Landay (2003). "DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice." Human-Computer Interaction 18: 259-324.
- Norman, D. A. (1986). Cognitive engineering. User Centered System Design. D. A. Norman and S. D. Draper. Hillsdale, NJ, Lawrence Erlbaum Associates: 31-61.
- Norman, D. A. (1988). The Design of Everyday Things, Currency Doubleday.
- Pane, J. F. (2002). A Programming System for Children that is Designed for Usability. Ph.D. Thesis. Computer Science Department. Pittsburgh, PA, Carnegie Mellon University.
- Pane, J. F. and B. A. Myers (2000). Tabular and textual methods for selecting objects from a group. Proceedings of VL 2000: IEEE International Symposium on Visual Languages, Seattle, WA, IEEE Computer Society: 157-164.
- Pane, J. F., B. A. Myers and L. B. Miller (2001). "Using HCI Techniques to Design a More Usable Programming System." submitted for publication.
- Pane, J. F., C. A. Ratanamahatana and B. A. Myers (2001). "Studying the language and structure in non-programmers' solutions to programming problems." International Journal of Human-Computer Studies 54: 237-264.
- Papert, S. (1980). Mindstorms. New York, Basic Books.
- Papert, S. (1993). The Children's Machine. New York, Basic Books.
- Pennington, N. and B. Grabowski (1990). The tasks of programming. In Psychology of Programming. T. R. G. G. J.-M. Hoc, R. Samurçay, and D. J. Gilmore. London, Academic Press: 45-62.
- Pitkow, J. and C. Kehoe (1998). GVU. 10th WWW User Survey. http://www.cc.gatech.edu/gvu/user_surveys/survey-1998-10/

-
- Reenskaug, T. (1979). Models - Views - Controllers. Technical Report. scanned copy at: <http://heim.ifi.uio.no/~trygver/mvc/index.html>. Xerox PARC.
- Repenning, A. (1994). "Designing domain-oriented visual end user programming environments." *Journal of Interactive Learning Environments* 4: 45-74.
- Repenning, A. (1995). Bending the Rules: Steps Toward Semantically Enriched Graphical Rewrite Rules. *Proceedings of Visual Languages*, Darmstadt, Germany: 226-233.
- Repenning, A. and A. Ioannidou (1997). Behavior Processors: Layers between End-Users and Java Virtual Machine. *Proceedings of IEEE VL 1997*, Capri, Italy. Sep. 23-26
- Robertson, T. J., S. Prabhakararao, M. Burnett, C. Cook, J. R. Ruthruff, L. Beckwith and A. Phalgune (2004). Impact of Interruption Style on End-User Debugging. *Proceedings of CHI 2004*, Vienna, Austria. April 2004
- Rode, J. (2002). Survey.vt.edu. <http://survey.vt.edu/> and <http://vtsurvey.sourceforge.net>
- Rode, J. (2002b). Survey: "Interactive Websites", Results. <http://filebox.vt.edu/users/jrode/publish/2002-05-survey/results/>
- Rode, J. (2005). Click - Tutorial Video. <http://phpclick.sourceforge.net/demo.php>
- Rode, J. and Y. Bhardwaj (2004a). Results from formative evaluations of Click prototypes. <http://purl.vt.edu/people/jrode/publish/2004-05-click-evaluation1/results.html> and <http://purl.vt.edu/people/jrode/publish/2004-11-click-evaluation3/results.html>
- Rode, J. and Y. Bhardwaj (2004b). Results from post-study questionnaire in formative evaluation #1 of Click prototype #1. <http://purl.vt.edu/people/jrode/publish/2004-05-click-evaluation1/questionnaire-results.html>
- Rode, J. and Y. Bhardwaj (2004c). Results from post-study questionnaire in formative evaluation #3 of Click prototype #2. <http://purl.vt.edu/people/jrode/publish/2004-11-click-evaluation3/questionnaire-results.html>
- Rode, J., Y. Bhardwaj, M. B. Rosson, M. Pérez-Quñones and J. Howarth (2005). Click. <http://phpclick.sourceforge.net>
- Rode, J. and J. Howarth (2004). Results from post-study questionnaire in formative evaluation #2 of Click prototype #2. <http://purl.vt.edu/people/jrode/publish/publish/2004-07-click-evaluation2/questionnaire-results.html>
-

Rode, J., J. Howarth, M. A. Pérez-Quiñones and M. B. Rosson (2004). An End-User Development Perspective on State-of-the-Art Web Development Tools. Technical Report #TR-05-03. Virginia Tech Computer Science.

Rode, J. and M. B. Rosson (2003). Programming at Runtime: Requirements & Paradigms for Nonprogrammer Web Application Development. Proceedings of IEEE Symposia on Human Centric Computing Languages and Environments, Auckland, New Zealand: 23-30. Oct. 28-31

Rode, J. and M. B. Rosson (2004). End-users' Mental Models of Concepts Critical to Web Application Development. Proceedings of IEEE Symposia on Human Centric Computing Languages and Environments, Rome, Italy. Oct. 26-29

Rosson, M. B. (1996). The human factor in software development. In Handbook of Computer Science and Engineering. Boca Raton, FL, CRC Press: 1596-1617.

Rosson, M. B., J. Ballin and H. Nash (2004). Everyday programming: Challenges and opportunities for informal web development. Proceedings of IEEE VL/HCC 2004, Rome, Italy: 123-130.

Rosson, M. B. and J. M. Carroll (1993). Active programming strategies for reuse. Proceedings of ECOOP'93: Object-Oriented Programming, 7th European Conference, Kaiserslautern, Germany, 26-30 July, Springer-Verlag: 4-20.

Rosson, M. B. and J. M. Carroll (1996). "The reuse of uses in Smalltalk programming." ACM Transactions on Computer-Human Interaction 3(3): 219-253.

Rosson, M. B., J. Ballin, J. Rode, B. Toward (2005). Designing for the Web revisited: A Survey of Casual and Experienced Web Developers. Proceedings of International Conference on Web Engineering, Sydney, Australia. July 27-29

Schwabe, D., G. Rossi and S. D. J. Barbosa (1996). Systematic Hypermedia Application Design with OOHDM. Proceedings of ACM Hypertext '96, Washington DC, USA: 116-128.

Scriven, M. (1967). The methodology of evaluation. Perspectives of curriculum evaluation. R. G. R. Tyler, & M. Scriven, Rand McNally: 39-83.

Shimomura, T. (2004). "Visual design and programming for Web applications." Journal of Visual Languages and Computing 16: 213-230.

Shneiderman, B. (1980). Software Psychology: Human Factors in Computer and Information Systems. Reading, MA, Addison-Wesley Publishers.

Shneiderman, B. (1983). Direct Manipulation: A Step Beyond Programming Languages. IEEE Computer. 16: 57-69.

Smarty (2005). Smarty Template Engine. <http://smarty.php.net/>

Sun Microsystems (2002a). <http://java.sun.com/products/jsp/>

Sun Microsystems (2005). JavaServer Faces. <http://java.sun.com/j2ee/javaserverfaces/>

Tanimoto, S. (1990). "VIVA: A Visual Language for Image Processing." Journal of Visual Languages and Computing 2(2): 127-139.

Taylor, M. J., J. McWilliam, H. Forsyth and S. Wade (2002). "Methodologies and website development: a survey of practice." Information and Software Technology 44: 381-391.

TechSmith (2005). Morae: A Complete Usability Testing Solution for Web Sites and Software. <http://www.techsmith.com/products/morae/>

Trellix (2005). Web Express. <http://www.trellix.com/products/trellixwebexpress.asp>

Turau, V. (2002). A framework for automatic generation of web-based data entry applications based on XML. Proceedings of 17th Symposium on Applied Computing, Madrid, Spain, ACM: 1121-1126.

UML (2005). Unified Modeling Language. <http://en.wikipedia.org/wiki/Uml>

Vora, P. R. (1998). "Designing for the Web: A Survey." ACM interactions May/June: 13-30.

Vroman Systems Inc. (2005). FormSite. <http://www.formsite.com/>

Wall, L. (1987). Practical Extraction and Report Language.

Watchfire (2005). Bobby/WebXACT. <http://webxact.watchfire.com/>

WebModels (2005). WebRatio. <http://www.webratio.com>

Whitley, K. N. and A. F. Blackwell (1997). Visual programming: The outlook from academia and industry. ESP-7. Workshop on Empirical Studies of Programmers: 180-208.

Wolber, D., Y. Su and Y. T. Chiang (2002). Designing Dynamic Web Pages and Persistence in the WYSIWYG Interface. Proceedings of IUI 2002, San Francisco, CA, USA. Jan 13-16

World Wide Web Consortium (1998). Cascading Style Sheets 2.
<http://www.w3.org/Style/CSS/>

Xamlon (2005). Xamlon. <http://www.xamlon.com/>

Xue, Q. (2005). PRADO: Component-based and event-driven Web programming framework for PHP 5. <http://www.xisc.com/>

YAHOO! (2005). YAHOO! Store. <http://store.yahoo.com/>

YesSoftware (2003). CodeCharge Studio.
http://www.yessoftware.com/products/product.php?product_id=1

Zdun, U. (2002). Dynamically Generating Web Application Fragments from Page Templates. Proceedings of 17th Symposium on Applied Computing, Madrid, Spain, ACM: 1113-1120.

Zorn, W. (2004). JavaScript: DHTML API, Drag & Drop for Images and Layers.
http://www.walterzorn.com/dragdrop/dragdrop_e.htm

ZyWeb (2005). <http://www.zyweb.com/>

Appendix A Survey of Virginia Tech Webmasters

A.1 IRB Approval

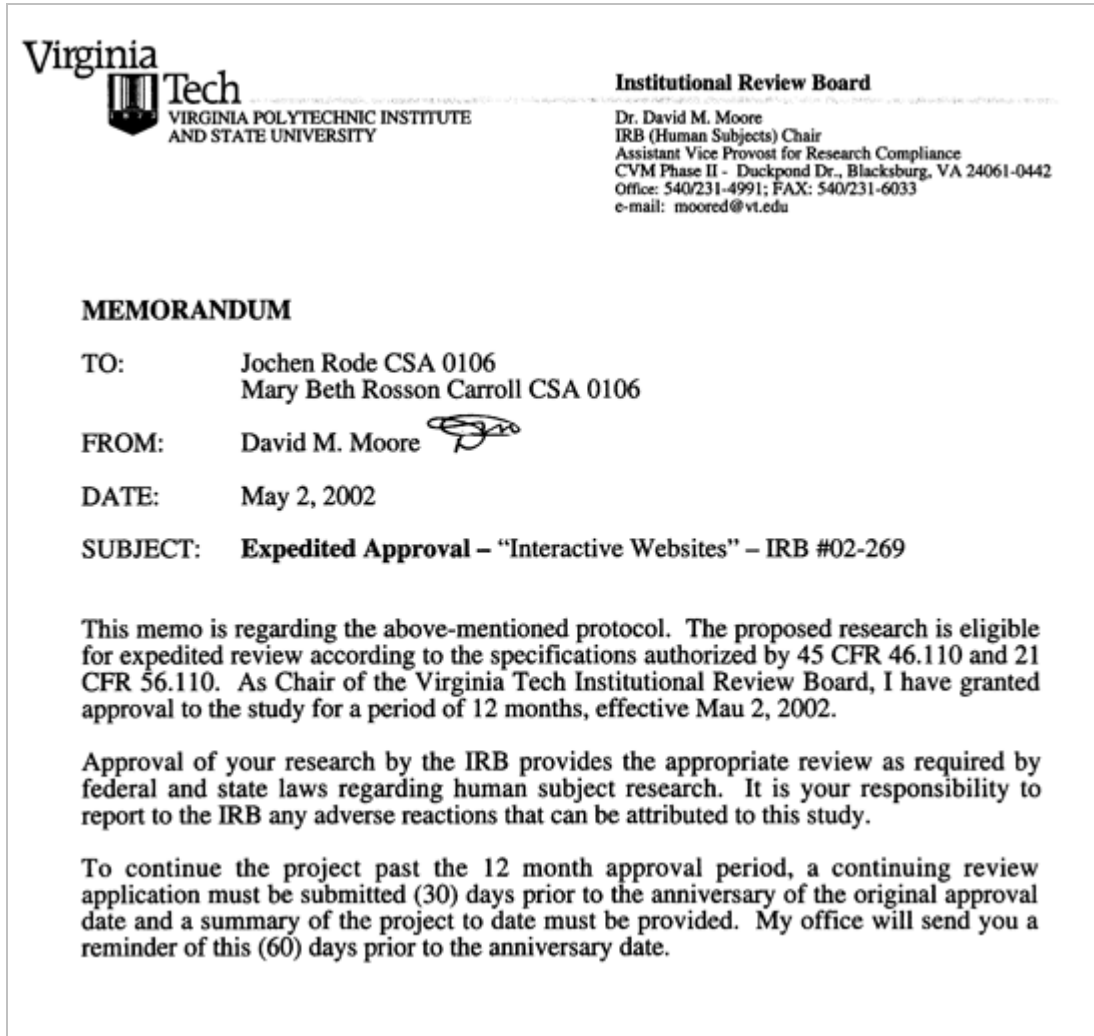


Figure 27: IRB approval for survey of VT webmasters

A.2 Survey Questionnaire and Summary Results

Table 29: Questionnaire and summary results from survey of Virginia Tech webmasters

The survey has **67** entries.

Interactive Websites

Interactive Website: **A website that offers advanced features** beyond static text and images (online auction, group calendar, discussion forum etc.). This would include websites that allow users to interact by inputting data, customizing the look & feel etc.

The goals of our research are to determine what types of interactive websites users want and how we can facilitate and simplify the development of these sites. **The data we collect from this survey will only be published in anonymous form.**

Thanks for your participation!



The Use of Interactive Websites

Which interactive websites do you use regularly (related *and unrelated* to Virginia Tech)?
(examples: eBay, MyYahoo, a room reservation software in our department)
66 responses



Where do you see opportunities for interactive websites in your environment (related *and unrelated* to Virginia Tech)?
For example, think about what is currently done on paper but may be done more efficiently or conveniently via the web.
(examples: A website for ... may help our bowling club to...; A website for ... would help the people in our department to...)
62 responses

From the opportunities that you listed above, consider the one in which you are most interested. What are the top 3 *interactive* features that you would like to see implemented in that website?
(examples: my online nutritional database should allow users to input what they ate today, allow them to search for a particular item etc.)
56 responses

Did you ever develop a computer program (*whether or not for the web*)?

Yes	43 (64%)	
No	23 (34%)	
no answer	1 (1%)	

Did you ever develop an interactive website?

Yes	29 (43%)	
No	38 (57%)	
no answer	0 (0%)	

Only respond to this question if you answered with "No" in the previous one:

Why haven't you developed an interactive website yet?

(Choose the option that best describes the reason.)

I don't have a need for an interactive website	8 (12%)	■
I am not interested in developing a website myself	5 (7%)	■
I would want to, but I expect it will be too difficult	10 (15%)	■
I would want to, but don't have the time	10 (15%)	■
I started building one, but dropped the project because it was too difficult	0 (0%)	
Other:	7 (10%)	■
<i>no answer</i>	27 (40%)	■

If you have not yet developed an interactive website please [click here to skip the following section.](#)

The Development of Interactive Websites

What interactive websites (aka web applications) did you develop or help to develop in the past?

If you want please also note their URL.

28 responses

From the interactive websites that you developed consider one that was particularly challenging. What were the top 3 most challenging issues you encountered while developing this website?

29 responses

What pending ideas, requests or plans do you have for the development of interactive websites?

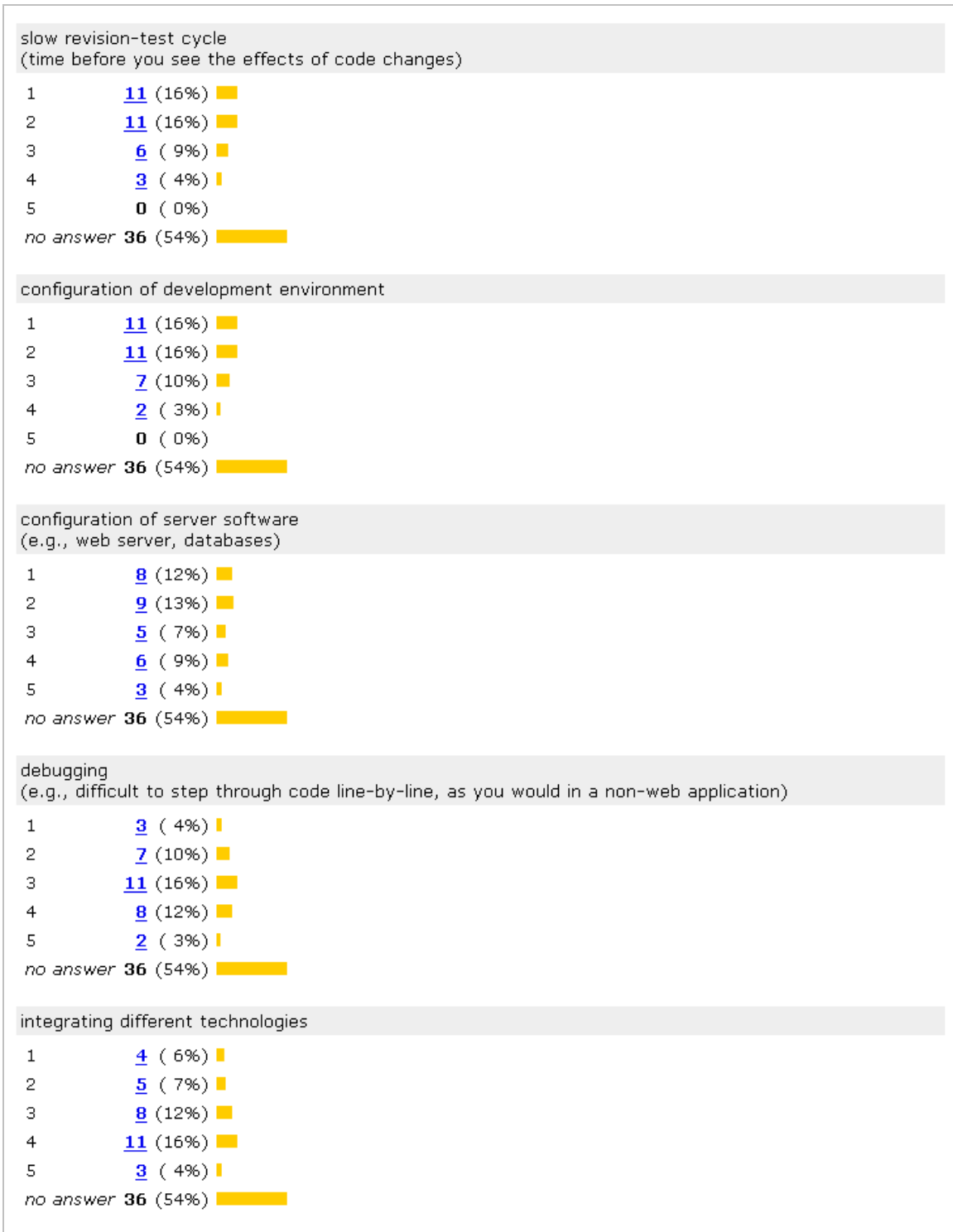
Please also include ideas that seem infeasible today because of constraints in time, budget, technology etc.

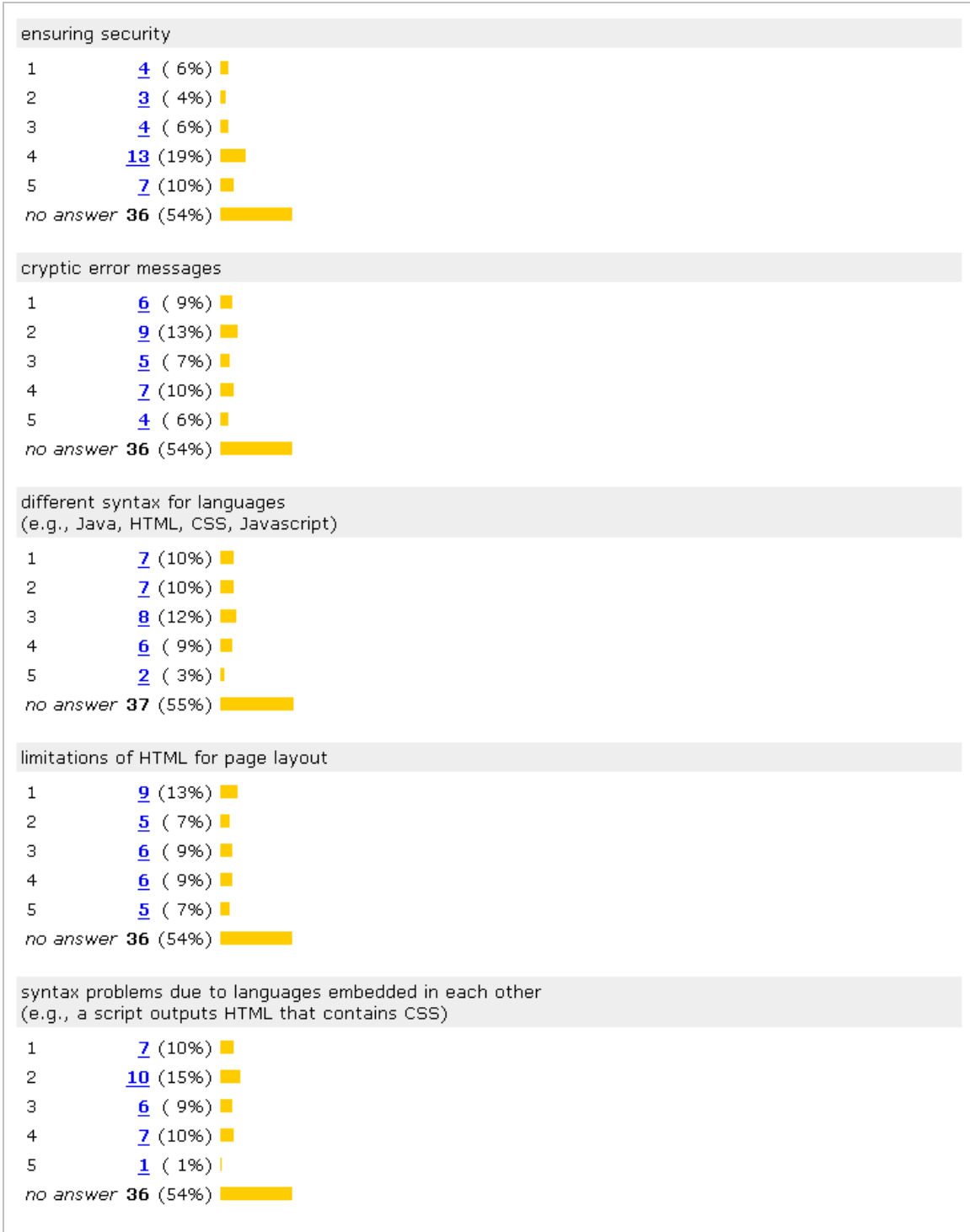
29 responses

Below you find a list of web development concerns.

Please rate each concern according to how problematic it is when building an interactive website.

Concern	1=not a problem at all, 5=severe problem
web-browser compatibility issues	
1	1 (1%)
2	7 (10%) ■
3	9 (13%) ■
4	8 (12%) ■
5	6 (9%) ■
<i>no answer</i>	36 (54%) ■





Other problems that you typically encounter during web development:

16 responses

Which web development tool(s) do you use on a regular basis?

Adobe GoLive!	<u>2</u> (3%)
Borland JBuilder	0 (0%)
IBM WebSphere Studio	0 (0%)
Macromedia Dreamweaver	<u>11</u> (16%) ■
Macromedia Flash	<u>5</u> (7%) ■
Macromedia Ultradev	<u>1</u> (1%)
Microsoft Frontpage	<u>12</u> (18%) ■
Microsoft Visual Interdev	<u>1</u> (1%)
Microsoft Visual Studio .NET	<u>1</u> (1%)
NetObjects Fusion	0 (0%)
Others:	<u>13</u> (19%) ■

Describe your "dream" web application development tool?

How would it facilitate development?

Consider this question a "wish list"!

(e.g. It would have a wizard to...; a feature that...)

24 responses

How long have you been developing interactive websites (aka web applications)?






























less than 1 year	<u>6</u> (9%) ■
about 1 year	<u>3</u> (4%)
about 2 years	<u>6</u> (9%) ■
about 3 years	<u>5</u> (7%) ■
about 4 years	<u>4</u> (6%) ■
about 5 years	<u>1</u> (1%)
more than 5 years	<u>6</u> (9%) ■
<i>no answer</i>	36 (54%) ■

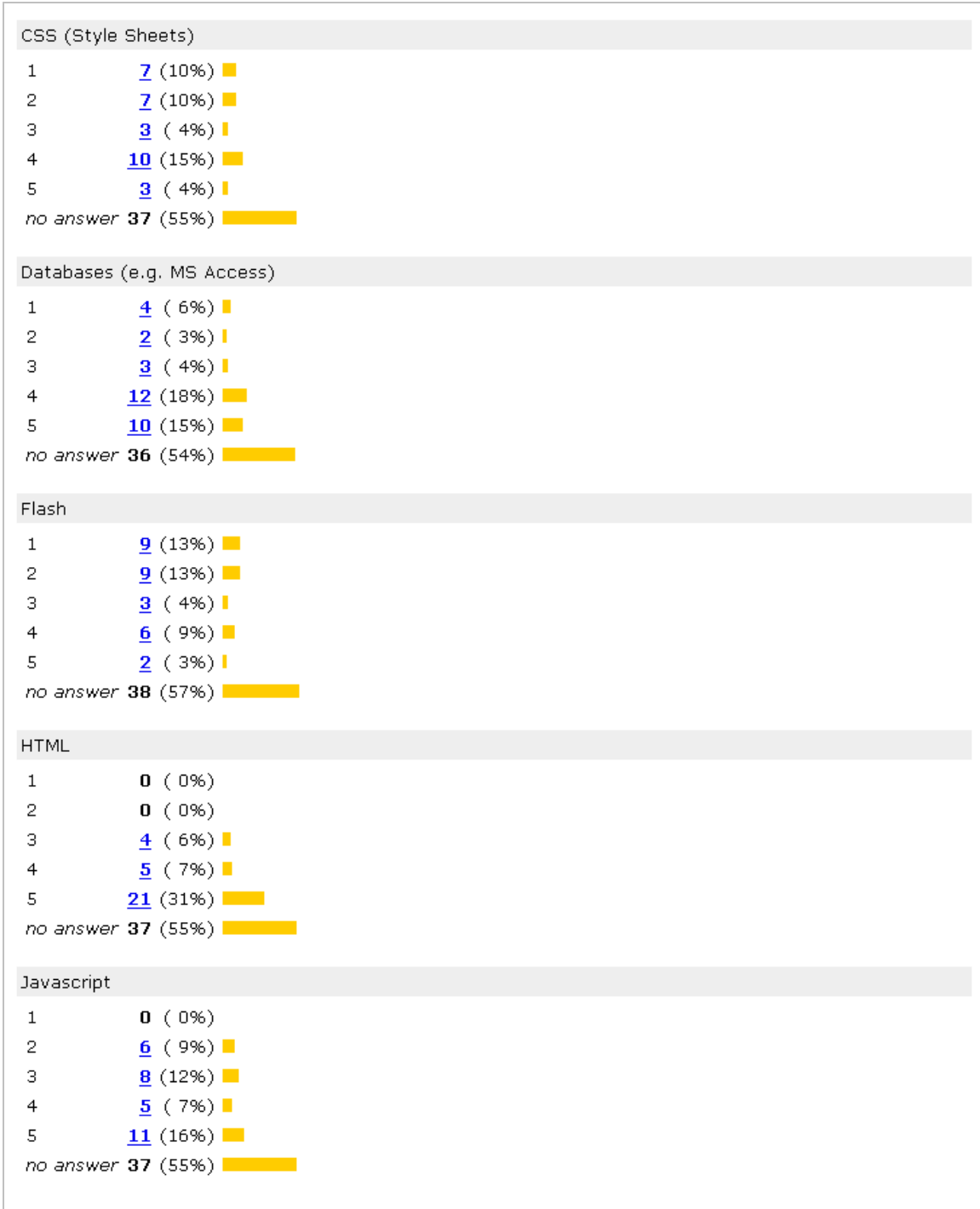
How do you rate your knowledge in web application development?

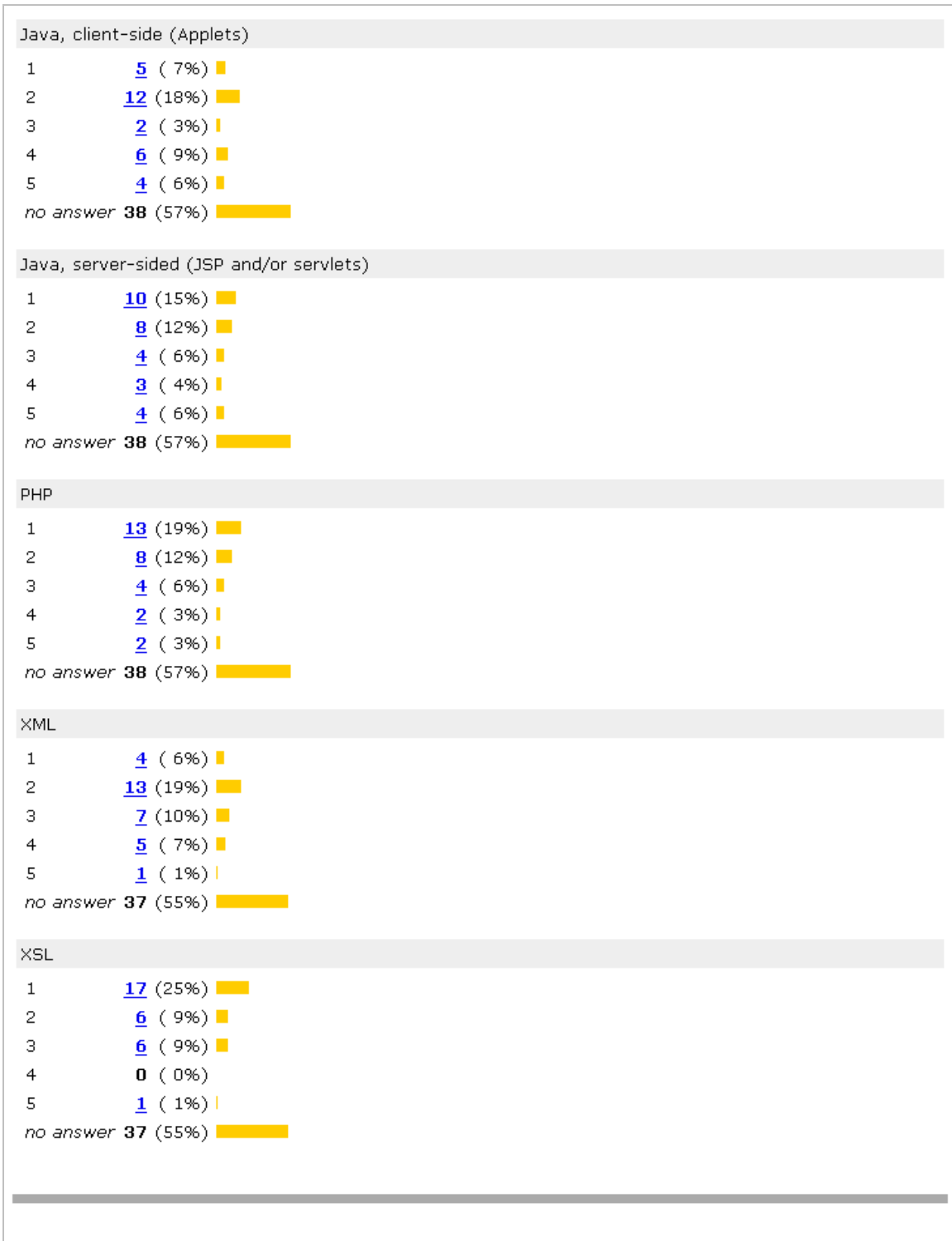
(1=no knowledge, 5=expert knowledge)

1	0 (0%)
2	<u>8</u> (12%) ■
3	<u>12</u> (18%) ■
4	<u>8</u> (12%) ■
5	<u>3</u> (4%)
<i>no answer</i>	36 (54%) ■

How familiar are you with the following technologies?

Technology	1=not familiar, 5=very familiar	
ASP/C#		
1	18 (27%)	
2	8 (12%)	
3	1 (1%)	
4	0 (0%)	
5	2 (3%)	
no answer	38 (57%)	
ASP/VB		
1	9 (13%)	
2	10 (15%)	
3	2 (3%)	
4	3 (4%)	
5	6 (9%)	
no answer	37 (55%)	
CGI/C		
1	9 (13%)	
2	6 (9%)	
3	8 (12%)	
4	5 (7%)	
5	1 (1%)	
no answer	38 (57%)	
CGI/Perl		
1	5 (7%)	
2	7 (10%)	
3	4 (6%)	
4	6 (9%)	
5	7 (10%)	
no answer	38 (57%)	
ColdFusion		
1	19 (28%)	
2	6 (9%)	
3	2 (3%)	
4	2 (3%)	
5	1 (1%)	
no answer	37 (55%)	











Personal Information

(The survey results will be published in **anonymous** form.)

Please choose the group(s) that best describe you.

VT Student (undergraduate)	19 (28%)	
VT Student (graduate)	16 (24%)	
VT Faculty	13 (19%)	
VT Staff	16 (24%)	
VT Alumni	8 (12%)	
Other:	2 (3%)	

Which club(s) or organization(s) are you a member of?

If any, which role(s) do you assume?

(examples: member of..., president of VT mikado club, treasurer of Kappa Omega Phi, webmaster of...)

43 responses




Do you maintain any websites (*whether or not interactive*)?

If so, what kinds of websites?












(examples: my personal homepage, a website for our research group, club website, class website)

59 responses

Your gender:

female	26 (39%)	
male	37 (55%)	
I prefer not to answer this question	4 (6%)	
<i>no answer</i>	0 (0%)	

Your age group:


under 21	5 (7%)	
21-25	26 (39%)	
26-30	9 (13%)	
31-35	4 (6%)	
36-40	5 (7%)	
41-45	2 (3%)	
46-50	6 (9%)	
51-55	3 (4%)	
56-60	2 (3%)	
over 60	1 (1%)	
I prefer not to answer this question	4 (6%)	
<i>no answer</i>	0 (0%)	

Your name: **63 responses**

Your e-mail (required for participation in the raffle): **63 responses**

Your major/department (no abbreviations please): **62 responses**

If you are interested in the results of this survey please check the box below.

E-mail me a copy of the survey results once they are published. [42](#) (63%) 

Appendix B Interviews of Semi-Professional Developers

B.1 IRB Approval

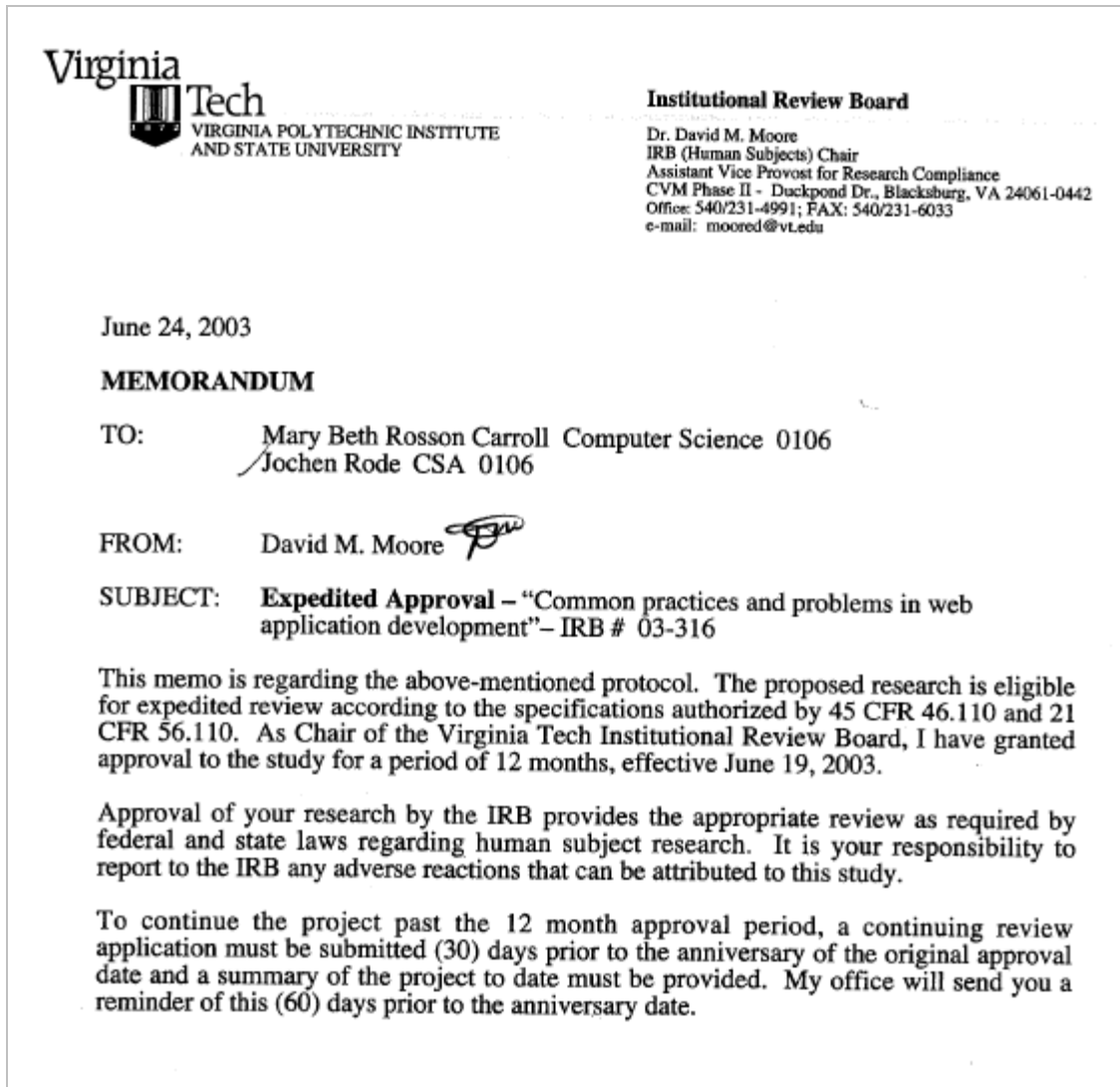


Figure 28: IRB approval for interview study

B.2 Pre-Interview Questionnaire

Table 30: Pre-Interview questionnaire of semi-professional web developers

Web Application Development

The purpose of this study is to gauge current trends in web application development. We define "web applications" as dynamic websites that provide interactive features to its users. This includes online databases that manage for example personnel information, and online calendars, forums, shopping sites, etc.

We are interested in **YOUR PERSONAL experience** in web application development. When responding to the following questions please think back to experiences you personally had while developing web applications.

1. General issues

1.1. What web applications did you develop in the past or do you currently develop?

Description:

URL:

Description:

URL:

Description:

URL:

Description:

URL:

Description:

URL:

1.2. What were the most challenging issues you encountered while developing the above mentioned applications?

1.4. Below you find a list of web development concerns. Please rate each concern according to how problematic it is when building web applications.

1.4.1. Needs analysis (1=not a problem at all, 7=severe problem):
 don't know 1 2 3 4 5 6 7

1.4.2. Availability & configuration of server software (databases, web servers, software libraries) (1=not a problem at all, 7=severe problem):
 don't know 1 2 3 4 5 6 7

1.4.3. Configuration of development environment (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.4. Designing & implementing the user interface layout (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.5. Designing graphics & icons (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.6. Integrating different technologies (e.g., PHP, ColdFusion, ASP, HTML, CSS, JavaScript) (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.7. Database design and connectivity (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.8. Authentication and authorization (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.9. Standard compliance, browser compatibility (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.10. Limitations of HTML for page layout (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.11. Ensuring security (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.12. Ensuring usability (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.13. Debugging (e.g., difficult to step through code line-by-line, as you would in a non-web application) (1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.14. Other:

(1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.15. Other:

(1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

1.4.16. Other:

(1=not a problem at all, 7=severe problem):

don't know 1 2 3 4 5 6 7

2. Development Practices

2.1. How do you typically approach a new web development project?

Do you have some kind of informal or formal process you go through? If so, please describe it.

2.2. Do you do any prototyping before you start the implementation? If so, how?

2.3. How do you debug web applications?

Please evaluate the following statements.

2.4. I search the web for snippets of code that I copy, paste & edit. (1=never, 7=very frequently):

1 2 3 4 5 6 7

2.5. I consult and scavenge code I have previously written myself. (1=never, 7=very frequently):

1 2 3 4 5 6 7

2.6. When developing web applications I typically collaborate with colleagues/friends. (1=never, 7=very frequently):

1 2 3 4 5 6 7

2.6.1. If you do collaborate, of what nature is the collaboration typically (e.g. validation of your ideas, brainstorming, share work, etc.)?

2.7. Do you use a HTML code validator to verify the standard-compliance of your code? (1=never, 7=always):

- 1 2 3 4 5 6 7

2.8. Do you check for cross-browser compatibility? (1=never, 7=always):

- 1 2 3 4 5 6 7

2.8.1. If so, how?

2.9. Do you check for usability? (1=never, 7=always):

- 1 2 3 4 5 6 7

2.9.1. If so, how?

2.10. Do you check for accessibility (for users with disabilities)? (1=never, 7=always):

- 1 2 3 4 5 6 7

2.10.1. If so, how?

2.11. Do you check for scalability & performance issues? (1=never, 7=always):

- 1 2 3 4 5 6 7

2.11.1. If so, how?

2.12. When learning about a new web technology I prefer learning from examples over learning from more general and verbose descriptions. (1=I strongly disagree, 7=I strongly agree):

- 1 2 3 4 5 6 7

2.13. What do you enjoy about web application development?

2.14. In a typical web development project I spend my time in the following way:

% Planning & conceptual design

% Designing the user interface

% Implementing the program logic (including debugging)

% Testing (functionality, security, compatibility, scalability, usability etc.)

% Learning new technologies

% Other:

% Other:

% Other:

100 %

2.15. How do you stay up-to-date regarding new web technologies?

2.15.1. Social network (colleagues, friends) (1=never, 7=very frequently):

1 2 3 4 5 6 7

2.15.2. Browsing books in a bookstore (1=never, 7=very frequently):

1 2 3 4 5 6 7

2.15.3. Regular visits to websites (1=never, 7=very frequently):

1 2 3 4 5 6 7

2.15.3.1. If so, which websites?

2.15.4. Mailing lists (1=never, 7=very frequently):

1 2 3 4 5 6 7

2.15.4.1. If so, which mailing lists?

2.15.5. Magazines (1=never, 7=very frequently):

1 2 3 4 5 6 7

2.15.5.1. If so, which magazines?

2.15.6. Other:

(1=never, 7=very frequently):

1 2 3 4 5 6 7

2.15.7. Other:

(1=never, 7=very frequently):

1 2 3 4 5 6 7

2.15.8. Other:

(1=never, 7=very frequently):

1 2 3 4 5 6 7

3. Tools

3.1. Please list the web development tool(s) you use on a regular basis (e.g. Dreamweaver, Visual Studio .NET). Then evaluate these tools and note what you do and don't like about them.

3.1.1. Tool:

3.1.1.1. Your level of experience (1=novice, 7=expert):

1 2 3 4 5 6 7

3.1.1.2. Ease of learning (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.1.3. Ease of use (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.1.4. Functionality (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.1.5. Overall Satisfaction (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.1.6. What you like:

3.1.1.7. What you don't like:

3.1.2. Tool:

3.1.2.1. Your level of experience (1=novice, 7=expert):

1 2 3 4 5 6 7

3.1.2.2. Ease of learning (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.2.3. Ease of use (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.2.4. Functionality (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.2.5. Overall Satisfaction (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.2.6. What you like:

3.1.2.7. What you do not like:

3.1.3. Tool:

3.1.3.1. Your level of experience (1=novice, 7=expert):

1 2 3 4 5 6 7

3.1.3.2. Ease of learning (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.3.3. Ease of use (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.3.4. Functionality (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.3.5. Overall Satisfaction (1=low, 7=high):

don't know 1 2 3 4 5 6 7

3.1.3.6. What you like:

3.1.3.7. What you don't like:

3.2. In your opinion, what could be done to simplify web application development?

**3.3. Describe your "dream" web application development tool.
How would it facilitate development? Consider this question a "wish list"!**

**3.4. Image a tool that enables people *who do not have programming experience* to design *simple* web applications (like basic online databases as opposed to just static websites).
How would the tool look like? What features would it need to provide?**

4. Demographics

4.1. Your name:

4.2. Your E-mail:

4.3. Your gender:

- female male I prefer not to answer this question

4.4. Your age group:

- under 21
 21-25
 26-30
 31-35
 36-40
 41-45
 46-50
 51-55
 56-60
 over 60
 I prefer not to answer this question

4.5. How do you rate your overall knowledge in web application development? (1=no knowledge, 7=expert knowledge):

- 1 2 3 4 5 6 7

4.6. How do you rate your knowledge of HTML? (1=no knowledge, 7=expert knowledge):

- 1 2 3 4 5 6 7

4.7. How do you rate your knowledge of Cascading Style Sheets? (1=no knowledge, 7=expert knowledge):

- 1 2 3 4 5 6 7

4.8. How do you rate your knowledge of client-side scripting (JavaScript, etc.)? (1=no knowledge, 7=expert knowledge):

- 1 2 3 4 5 6 7

4.9. How do you rate your knowledge of server-side scripting (PHP or ASP or JSP or ColdFusion, etc.)? (1=no knowledge, 7=expert knowledge):

- 1 2 3 4 5 6 7

4.10. How long have you been developing web applications?

- less than 6 months
 6-11 months
 1 year
 2 years
 3 years
 4 years
 5 years
 more than 5 years

4.11. Your professional background: Please give a short description of your major area(s) of formal education and/or job experience (whether or nor related to web development).

4.12. Please choose all the groups that apply to you:

- Student (graduate)
- Faculty
- Staff
- Alumni
- Professional web developer

Other:

4.13. What are your principal reasons for designing web applications (check all that apply):

- Principal job responsibility (i.e., Webmaster-type responsibilities)
- One of several tasks required by job
- Hobby/personal interest
- To help friends/family/non-profit organizations


Other:

4.14. Do you have any comments?

Appendix C Comprehensive Survey of Web Developers

C.1 IRB Approval

PENNSTATE

 Vice President for Research
Office for Research Protections

The Pennsylvania State University
212 Kern Graduate Building
University Park, PA 16802-3301

(814) 865-1775
Fax: (814) 863-8699
www.research.psu.edu/orp

Date: August 11, 2003

From: Jodi Mathieu, IRB Administrator

To: Mary Beth Rosson

Subject: Results of Review of Proposal - Expedited (IRB #16151)

Approval Expiration Date: August 10, 2004

"Web Development by Non-Programmers"

The Social Science Committee of the Institutional Review Board has reviewed and approved your proposal for use of human participants in your research. **This approval has been granted for a one-year period.**

COMMENT: Enclosed is the dated, IRB-approved informed consent to be used when recruiting participants for this research.

Approval for use of human participants in this research is given for a period covering one year from today. **If your study extends beyond this approval period, you must contact this office to request an annual review of this research.**

Subjects must receive a copy of any informed consent documentation that was submitted to the Office for Research Protections for review.

By accepting this decision you agree to notify the Office for Research Protections of (1) any additions or procedural changes that modify the participants' risks in any way and (2) any unanticipated subject events that are encountered during the conduct of this research. Prior approval must be obtained for any planned changes to the approved protocol. Unanticipated participant events must be reported in a timely fashion.

On behalf of the committee and the University, I thank you for your efforts to conduct your research in compliance with the federal regulations that have been established for the protection of human participants.











JLM/mbc
Enclosure

cc: Jochen Rode ✓
Department Head, Information Sciences and Technology
Associate Dean, Information Sciences and Technology

Figure 29: IRB approval for comprehensive survey of web developers

C.2 Questionnaire and Summary of Results

Table 31: Comprehensive survey of web developers: Questionnaire and summary of results









Results Summary (n=334)			
1. Have you ever developed a website yourself from scratch?			
		Response Percent	Response Total
Yes		95.4%	314
No		4.6%	15
Total Respondents			329
(skipped this question)			5
2. Have you ever maintained a website that someone else built or taken over the development of a website that someone else started?			
		Response Percent	Response Total
Yes		78.8%	260
No		21.2%	70
Total Respondents			330
(skipped this question)			4
3. What is the most common reason for you to develop or maintain a website?			
		Response Percent	Response Total
School/education		5.1%	17
Work		54.7%	181
Special interest or hobby		16.6%	55
Civic, volunteer, or community work		12.4%	41
Family and friends		4.8%	16
Other (please specify)		6.3%	21
Total Respondents			331
(skipped this question)			3

4. The following question asks about 10 web features or functions you may have used in your web development activity. Please select the choice which best describes whether you have incorporated the feature or function in your web work.

	never attempted to create	tried but did not finish	completed successfully	Response Total
Online surveys and forms	28% (90)	9% (29)	63% (206)	325
Online database (e.g., employee database)	38% (124)	9% (30)	53% (171)	325
Online resource scheduling (e.g., meeting rooms)	70% (224)	5% (16)	25% (80)	320
Event calendar	42% (135)	6% (20)	52% (169)	324
eCommerce transaction(s)	55% (176)	8% (25)	37% (118)	319
Content management system (collaborative website maintenance /publishing)	48% (154)	10% (31)	42% (135)	320
Discussion board	53% (168)	8% (27)	39% (123)	318
Real-time (synchronous) chat	78% (249)	8% (24)	14% (45)	318
Access restrictions/login	33% (107)	8% (26)	59% (191)	324
Member registration	40% (128)	10% (32)	50% (162)	322
Total Respondents				329
(skipped this question)				5

5. The following question asks you to judge the value of these same 10 features in your web development projects, regardless of whether you have worked with them yet or not. Please answer using a scale from 1 (one) to 5 (five), where 1 means not valuable at all for your current or future web projects, and 5 means extremely valuable for your current or future projects.

	Not Valuable 1	2	3	4	Extremely valuable 5	Response Average
Online surveys and forms	10% (31)	6% (20)	18% (56)	25% (79)	42% (132)	3.82
Online database (e.g., employee database)	12% (39)	6% (18)	10% (33)	15% (49)	56% (179)	3.98
Online resource scheduling (e.g., meeting rooms)	25% (80)	20% (63)	21% (67)	15% (48)	18% (57)	2.81
Event calendar	12% (37)	9% (30)	18% (56)	28% (89)	33% (105)	3.62
eCommerce transaction(s)	20% (62)	9% (30)	14% (43)	15% (48)	42% (133)	3.51
Content management system (collaborative website maintenance /publishing)	12% (38)	8% (24)	18% (57)	24% (76)	38% (119)	3.68
Discussion board	12% (38)	17% (54)	28% (88)	22% (69)	21% (65)	3.22
Real-time (synchronous) chat	33% (104)	27% (85)	21% (67)	12% (38)	7% (21)	2.32
Access restrictions/login	9% (30)	8% (24)	7% (21)	18% (57)	58% (184)	4.08
Member registration	12% (39)	8% (24)	9% (29)	21% (65)	50% (159)	3.89
Total Respondents						320
(skipped this question)						14

6. What is the primary development tool you use for working on your site(s)?			
		Response Percent	Response Total
Macromedia Dreamweaver		42.1%	130
Microsoft Frontpage		12.6%	39
Adobe GoLive!		2.3%	7
NetObjects Fusion		1%	3
Text Editor (e.g., notepad, textpad, TextEdit, Emacs, vi)		9.7%	30
Netscape Composer		2.9%	9
Namo WebEditor		0.6%	2
HTML editor (Bbedit, Homesite)		12.3%	38
Site/Page builder tools (e.g., Lycos Tripod, Lycos Angelfire, Yahoo! Geocities, Homestead)		1.6%	5
Other (please specify)		14.9%	46
Total Respondents			309
(skipped this question)			25

7. What is the main reason you use your primary web development tool?			
		Response Percent	Response Total
It was already installed on my computer		1.9%	6
It is what my employer/colleagues/organization uses		11.4%	35
The cost, it was the most cost-effective option		7.8%	24
A friend or coworker recommended it to me		3.9%	12
It has the functionality that meets my needs		52.9%	163
It is very easy to use		12.7%	39
Other (please specify)		9.4%	29
Total Respondents			308
(skipped this question)			26

8. What are the three things you like MOST about your primary web development tool?			
		Response Percent	Response Total
1)		100%	286
2)		95.1%	272
3)		86%	246
Total Respondents			286
(skipped this question)			48

9. What are the three things you like LEAST about your primary web development tool?			
		Response Percent	Response Total
1)		100%	259
2)		74.5%	193
3)		55.2%	143
Total Respondents			259
(skipped this question)			75

10. How much experience do you have with the following tools or resources in your web work? Please answer using a scale from 1 (one) to 5 (five) where 1 means no experience and 5 means a great deal of experience.						
	no experience 1	2	3	4	great deal of experience 5	Response Average
Database packages such as Access, Oracle, Filemaker, Foxpro	22% (68)	11% (34)	16% (49)	19% (59)	31% (94)	3.25
Multimedia tools such as Macromedia Flash or Director	32% (96)	26% (79)	20% (61)	11% (32)	12% (35)	2.44
Streaming media tools/features such as video or audio	40% (121)	24% (72)	22% (66)	8% (24)	7% (21)	2.18
Client side scripting languages such as Javascript or Jscript	23% (71)	17% (51)	20% (60)	21% (65)	19% (57)	2.95
Server side scripting/programming languages (e.g., PHP, ASP, Coldfusion)	33% (99)	11% (32)	11% (32)	15% (46)	31% (95)	3.02
Total Respondents						304
(skipped this question)						30

11. If you were building a web site that required you to learn something new, how likely would you be to consult the following resources for assistance? Please rate the resources on a scale from 1 (one) to 5 (five) where 1 means not likely to consult, and 5 means very likely to consult.

	not likely to consult 1	2	3	4	very likely to consult 5	Response Total
An interactive software wizard that takes you step-by-step through the new task	21% (54)	14% (35)	24% (63)	22% (57)	19% (50)	259
Examples of similar sites from which you can get ideas and copy code	4% (11)	10% (25)	16% (42)	25% (64)	45% (117)	259
A class or seminar on how to design and build your site	23% (59)	28% (71)	22% (57)	16% (41)	12% (30)	258
A software agent that makes suggestions on an as-needed basis	41% (107)	23% (59)	14% (35)	13% (34)	9% (23)	258
FAQs, books, or tutorials	2% (6)	2% (6)	5% (12)	22% (56)	69% (177)	257
A friend or coworker who knows how to do it	8% (21)	10% (26)	16% (41)	30% (78)	36% (93)	259
Tech support/phone hotline	46% (119)	27% (69)	15% (40)	7% (17)	5% (14)	259
Total Respondents						261
(skipped this question)						73


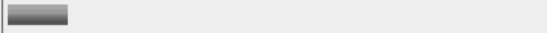

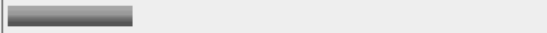

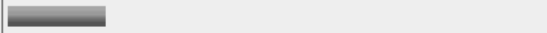
<p>12. This question asks about some specific types of website testing. Please answer using a scale from 1 (one) to 5 (five) where 1 means never and 5 means always. When working on websites, how often do you test to make sure ...</p>						
	Never 1	2	3	4	Always 5	Response Average
It is easy for users to do what they want to do on the site and to find what they might be looking for (usability)	2% (6)	2% (6)	13% (35)	26% (73)	57% (156)	4.33
Users who might have disabilities will be able to use your site (ADA compliance, section 508, Equal Access, etc.)	26% (71)	21% (59)	21% (59)	16% (43)	16% (44)	2.75
It will work across different operating systems (Windows, Mac, Linux, etc.) and different web browsers such as Internet Explorer, Netscape Navigator, Safari, etc. (cross-platform and browser compatibility)	8% (22)	10% (27)	18% (49)	29% (79)	36% (99)	3.75
Total Respondents						276
(skipped this question)						58
<p>13. Please briefly describe <i>when and how</i> you test the websites you build or maintain?</p>						
Total Respondents						246
(skipped this question)						88

14. How often do you experience problems with the following kinds of issues that sometimes arise in web development work? Please use a scale from 1 (one) to 5 (five) where 1 means hardly ever, and 5 means quite often.						
	hardly ever a problem 1	2	3	4	quite often a problem 5	Response Average
difficulty in adapting someone else's code for your work	18% (49)	28% (76)	31% (84)	17% (45)	6% (16)	2.64
getting content in a timely manner from others you depend on	16% (43)	14% (38)	20% (54)	24% (64)	27% (73)	3.32
links to pages or services that break or change	22% (59)	34% (92)	23% (63)	14% (39)	8% (21)	2.53
getting interactive forms to work	31% (82)	27% (72)	22% (58)	15% (39)	6% (16)	2.38
pages that don't work on all browsers	19% (53)	32% (86)	27% (74)	14% (39)	7% (20)	2.58
reformatting from text editors or word-processed documents	27% (74)	23% (61)	23% (62)	15% (41)	12% (33)	2.62
something that works on your machine when you are building, but not after it is uploaded (links, images)	33% (92)	31% (85)	23% (64)	9% (25)	4% (10)	2.19
making your pages look exactly how you want them to look	12% (34)	30% (84)	30% (82)	18% (50)	9% (26)	2.82
fighting the constraints of WYSIWYG editors (Dreamweaver, Frontpage)	36% (98)	20% (53)	20% (54)	13% (34)	11% (31)	2.43
setting up and configuring your web development tool(s)	40% (107)	32% (86)	19% (51)	6% (15)	3% (9)	2.00
integrating different technologies such as HTML, style sheets, JavaScript, or Server-side scripting	29% (79)	29% (78)	22% (61)	13% (35)	7% (20)	2.41
Total Respondents						276
(skipped this question)						58

15. In your web development work, how often are the following statements true? Please answer using a scale from 1 (one) to 5 (five) where 1 means hardly ever, and 5 means quite often.						
	hardly ever 1	2	3	4	quite often 5	Response Average
I work with colleagues to create or maintain sites	32% (86)	18% (49)	18% (50)	17% (46)	15% (41)	2.66
I am dependent on colleagues to provide content for the site	26% (71)	21% (58)	18% (49)	16% (44)	19% (52)	2.81
I have colleagues who regularly test or critique my work	16% (44)	21% (58)	25% (68)	21% (57)	17% (45)	3.00
I have 1 or more people I regularly ask for help working on my sites	39% (107)	22% (59)	20% (54)	11% (31)	7% (20)	2.25
I work with a design expert or architect for initial design of the site	63% (170)	17% (45)	11% (31)	4% (10)	5% (14)	1.71
I search the web for snippets of code that I can directly copy, paste and edit	17% (46)	19% (52)	28% (76)	19% (51)	18% (48)	3.01
I consult and reuse/copy code I have previously written myself	12% (33)	5% (13)	11% (31)	25% (67)	47% (129)	3.90
I search for example websites on which to model my website designs	15% (41)	23% (62)	26% (70)	19% (51)	18% (49)	3.02
Total Respondents						273
(skipped this question)						61

16. How much do you agree or disagree with the following statements about your web development work? Please rate each statement on a scale from 1 (one) to 5 (five) where 1 means strongly disagree and 5 means strongly agree.						
	strongly disagree 1	2	3	4	strongly agree 5	Response Average
I spend a lot of time making sure my site's layout, formatting, content, and interactive elements are just right before I "go live"	1% (3)	4% (11)	17% (47)	32% (87)	46% (126)	4.18
When taking on a new web project, I immediately start constructing pages	29% (79)	28% (77)	20% (55)	15% (41)	7% (20)	2.43
When working on a web site, I have a systematic process I follow	2% (5)	18% (50)	28% (77)	26% (70)	26% (71)	3.56
As I work on a web project, I think about how I might come back later to change or expand it	1% (4)	5% (13)	16% (45)	31% (86)	46% (126)	4.16
After my websites "go live", I check back frequently to make sure that everything works like it should (links, images, forms, etc.)	0% (1)	7% (20)	20% (56)	24% (67)	47% (130)	4.11
As I work on a web project, I consider the needs of people who may revise it in the future	10% (26)	13% (35)	25% (69)	26% (72)	26% (71)	3.47
Total Respondents						273
(skipped this question)						61

17. Please tell us about how often you use the following kinds of applications:								
	Never	About once per year	Several times per year	About once per month	Several times per month	1-2 times per week	Daily	Response Average
Multimedia Software (e.g., Director, Flash)	34% (92)	13% (35)	16% (42)	10% (28)	9% (24)	7% (20)	10% (27)	3.09
Spreadsheet (e.g., Excel)	5% (13)	4% (12)	14% (37)	11% (29)	22% (59)	21% (56)	23% (63)	4.97
Graphics (e.g., Photoshop, Paint Shop Pro)	2% (6)	2% (5)	5% (13)	7% (20)	19% (52)	25% (66)	40% (107)	5.72
Publishing (e.g., Pagemaker)	44% (117)	11% (29)	16% (43)	9% (25)	10% (27)	6% (16)	4% (10)	2.64
Database (e.g., Access, Oracle)	15% (41)	4% (12)	11% (30)	6% (16)	11% (30)	15% (41)	37% (99)	4.86
Web Editors (e.g., Dreamweaver, Frontpage)	13% (36)	4% (11)	7% (20)	4% (12)	10% (28)	14% (37)	46% (124)	5.21
Version Management Tools (e.g., SourceSafe, CVS)	63% (168)	6% (16)	6% (16)	2% (6)	5% (13)	3% (9)	14% (38)	2.47
Content Management Systems (e.g., RedDot, Zope, Contribute)	65% (174)	6% (17)	6% (16)	5% (14)	6% (15)	4% (12)	7% (20)	2.24
Total Respondents								269
(skipped this question)								65

18. For approximately how many years have you used computers?			
		Response Percent	Response Total
1-5 years		1.1%	3
5-10 years		11.8%	32
10-15 years		22.1%	60
15-20 years		23.6%	64
20-25 years		23.2%	63
25 years or more		18.1%	49
Total Respondents			271
(skipped this question)			63

19. On a scale from 1 (one) to 5 (five), where one is beginner and five is advanced, how would you characterize yourself as a computer user?			
		Response Percent	Response Total
beginner	1	0.4%	1
	2	1.1%	3
	3	9.3%	25
	4	31.7%	85
advanced	5	57.5%	154
Total Respondents			268
(skipped this question)			66

20. Have you had formal (classes, etc.) or professional training in programming?			
		Response Percent	Response Total
Yes		51.3%	138
No		48.7%	131
Total Respondents			269
(skipped this question)			65

21. Do you consider yourself a programmer, whether in web or other types of programming?			
		Response Percent	Response Total
Yes		51.3%	138
No		48.7%	131
Total Respondents			269
(skipped this question)			65

22. Approximately how many hours per week do you spend on web development or maintenance?			
		Response Percent	Response Total
less than 1 hour		11.2%	30
1-5 hours		18.7%	50
6-10		14.2%	38
11-15		4.9%	13
16-20		11.2%	30
21-30		10.1%	27
31-40		12.4%	33
41 + hours		17.2%	46
Total Respondents			267
(skipped this question)			67

23. Approximately how many years have you been doing web development or maintenance?			
		Response Percent	Response Total
less than 1 year		3.7%	10
1-3 years		16.7%	45
4-5 years		34.9%	94
6-10 years		40.5%	109
11 or more years		4.1%	11
Total Respondents			269
(skipped this question)			65

24. Approximately how many pages does the largest web site you've developed or maintained have?			
		Response Percent	Response Total
1-5 pages		1.9%	5
6-10		7.8%	21
11-15		7.5%	20
16-20		5.2%	14
21-30		7.8%	21
31-50		10.4%	28
51-100		13.1%	35
101-200		11.2%	30
201-500		12.3%	33
501-1000		7.1%	19
1001 or more		15.7%	42
Total Respondents			268
(skipped this question)			66

25. Please tell us where you learned about web development. Select the option(s) below that apply to you (you may check more than one option).			
		Response Percent	Response Total
self taught		95.2%	257
formal classes		32.2%	87
online tutorials or tutorials bundled with web development software		61.1%	165
coworkers or students in my program		33.7%	91
Other (please specify)		22.6%	61
Total Respondents			270
(skipped this question)			64

26. When you think back about your web development history, what memory stands out most in your mind?	
Total Respondents	213
(skipped this question)	121

27. Please rate yourself in the following 9 statements on a scale from 1 (one) to 5 (five) where 1 means rarely and 5 means often.

	rarely 1	2	3	4	often 5	Response Average
Even when I know I have satisfied the requirements for a task or project, I continue to add new features or enhance it in other ways	3% (7)	10% (27)	16% (43)	33% (88)	38% (102)	3.94
When I work through a process, I am confident that I have done a good job when I reach the end	1% (2)	1% (4)	15% (41)	41% (109)	42% (111)	4.21
When an object (e.g., furniture, bicycle) is broken, I work skillfully with my hands to fix or repair it	6% (17)	13% (36)	20% (54)	27% (73)	33% (87)	3.66
When I'm engaged and making progress on a task, I stop to pursue questions just out of curiosity	2% (6)	9% (25)	22% (60)	36% (95)	30% (81)	3.82
When there is a flaw in my work, whether serious or minor, I analyze and resolve it	0% (0)	1% (3)	6% (17)	35% (92)	58% (154)	4.49
I review work I have done to double check for errors	0% (1)	3% (7)	7% (19)	38% (102)	52% (138)	4.38
To explain an idea to those who don't understand it, I design graphs, charts, and other visual displays	9% (25)	14% (37)	21% (56)	26% (70)	29% (77)	3.52
With possessions (e.g., cars, bicycles, computers) and projects (e.g. household, work, personal), I am conscientious about maintenance tasks over time	1% (4)	8% (21)	31% (83)	35% (94)	24% (65)	3.73
When in front of a group, I tell jokes, stories, and make analogies to demonstrate a point	8% (20)	10% (27)	14% (36)	33% (87)	36% (96)	3.80
Total Respondents						267
(skipped this question)						67

28. Which of the following categories best describes your occupational status? Are you ...			
		Response Percent	Response Total
a student		3.4%	9
currently working in a civilian job		44.4%	118
currently working in a university job		12.8%	34
currently working in a military or government job		6%	16
retired		13.5%	36
unemployed/not currently working		2.6%	7
Other (please specify)		17.3%	46
Total Respondents			266
(skipped this question)			68

29. What is your occupation (if currently working)?	
Total Respondents	222
(skipped this question)	112

30. What is your gender?			
		Response Percent	Response Total
Female		27.1%	72
Male		72.9%	194
Total Respondents			266
(skipped this question)			68



31. What is your age group?			
		Response Percent	Response Total
under 21 years of age		0.8%	2
21-25		6.8%	18
26-30		14.8%	39
31-35		13.6%	36
36-40		10.2%	27
41-45		8.3%	22
46-50		9.1%	24
51-55		11%	29
56-59		6.4%	17
60 years of age or older		18.9%	50
Total Respondents			264
(skipped this question)			70

32. What is your race or ethnicity?			
		Response Percent	Response Total
white (non-Hispanic) or Caucasian		84.4%	222
Hispanic Origin		4.2%	11
Black/African American		1.5%	4
Asian		4.2%	11
Native Hawaiian or Pacific Islander		0.8%	2
American Indian or Alaskan Native		0%	0
Other (please specify)		4.9%	13
Total Respondents			263
(skipped this question)			71

33. What is the highest level of school/education you have completed?			
		Response Percent	Response Total
did not complete high school/secondary school		1.1%	3
high school diploma/GED/completion of secondary school		2.6%	7
some college (university) or technical school		20.3%	54
associate's degree		9%	24
4-year college or university degree		30.5%	81
some graduate work		9.8%	26
master's degree		20.7%	55
doctorate		3.8%	10
Other (please specify)		2.3%	6
Total Respondents			266
(skipped this question)			68

34. What is your country of residence?	
Total Respondents	264
(skipped this question)	70

35. How did you find out about this survey (e.g., through what organization)?	
Total Respondents	258
(skipped this question)	76

36. Would you like to participate in the drawing for one of 10 prizes of \$50 (U.S. dollars)?			
		Response Percent	Response Total
Yes		81.5%	211
No (if no, please skip to question 39)		18.5%	48
Total Respondents			259
(skipped this question)			75

37. What is your full name (this is required if you wish to participate in the prize drawing)? [Your name and email address will not be seen by anyone outside of the research team and will only be used to contact you if you win a prize.]	
Total Respondents	217
(skipped this question)	117

38. What is your email address? (Please verify that you have entered the correct email address. We cannot contact you if you win a prize and we do not have a working email address for you)	
Total Respondents	218
(skipped this question)	116

39. Is there anything else you would like to tell us about web development or this survey?	
Total Respondents	117
(skipped this question)	217

Appendix D Mental Models Study 1

D.1 IRB Approval

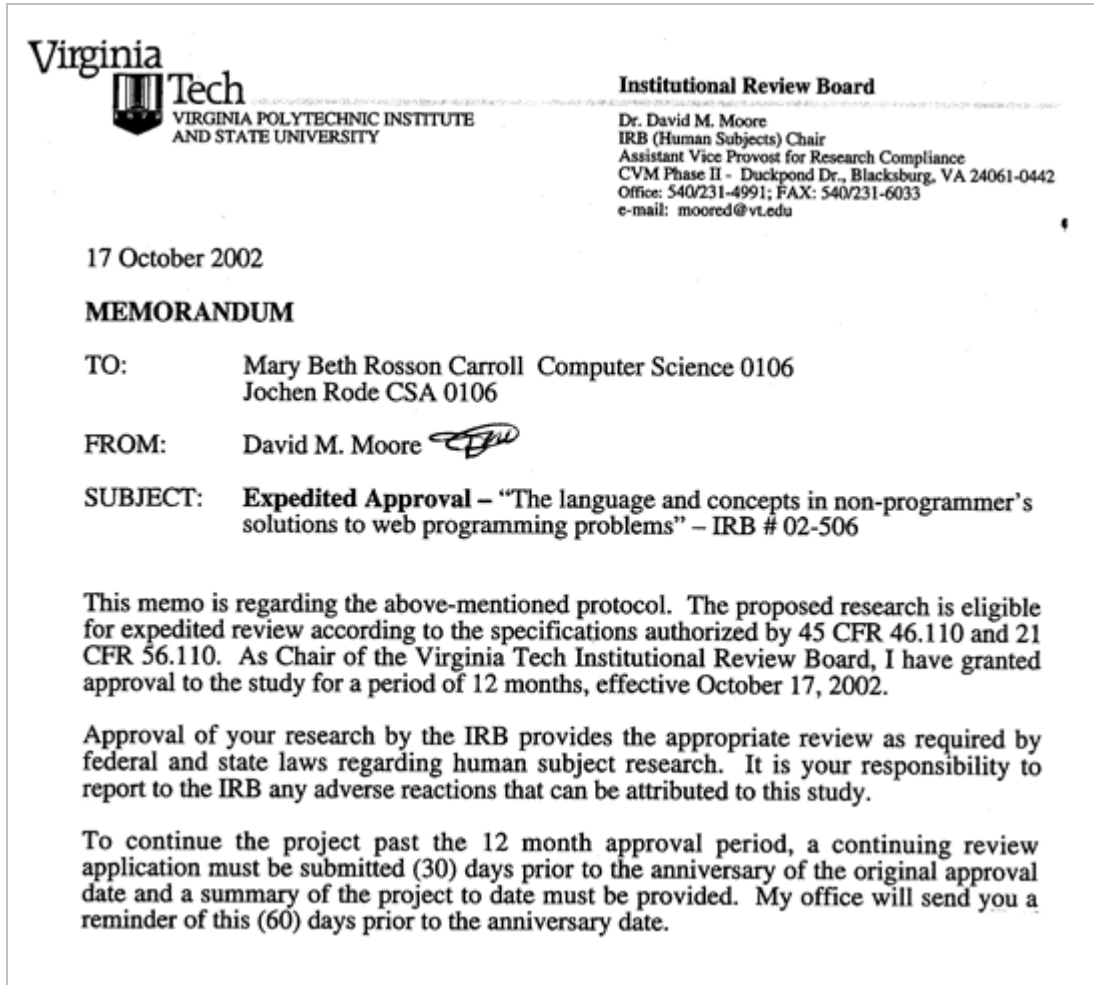


Figure 30: IRB approval for MMODELS-1

D.2 Participants' Instructions

Table 32: Participants' instructions for mental models study 1

The language and concepts in non-programmer's solutions to web programming problems

Purpose of the study

We intend to design a system which allows webmasters without prior programming experience to build interactive websites (aka web applications). With this study we want to discover what a “natural approach” to web programming may look like. The system we envision should use the language and concepts that webmasters are familiar with and avoid unnecessary hurdles.

Please answer the following questions with words and concepts that are most intuitive to you and require the least possible amount of “mental effort”. Remember to reply according to what feels most intuitive to *you*, not what you expect to be most intuitive to others.

The study consists of two parts.

Part I: Identify and name elements

We have provided you with a series of screen-shots of a simple web application for tracking membership information. Divide each of the given screen-shots into a number of parts or elements. Circle the elements with a pencil and give them a *general name*. Figure 1 illustrates the task using an example outside of the domain of web-design.

You can circle and name single elements or groups of elements. You can re-use the names you have given to other elements. Do it according to what feels most intuitive to you. The names you choose will help us to find intuitive labels and descriptions for our programming system.

Part II: Describe the website's behavior

Take some time to play with the membership tracking application on the computer until you have the feeling that you understand what it does and how it behaves.

Now imagine the following scenario: You have a magical machine that can read and understand your handwriting. If you give this machine a *full description* of how a web application should work, it will create it for you. The machine has the level of intelligence of a “naïve alien.” Use the screen-shots and paper & pencil to explain the application's behavior to the magical machine. Don't assume that the magical machine can infer much. Try to describe the details of the application's behavior. If you need to refer to any elements in the screen-shots just mark them with an asterisk, or number etc. for convenient reference. *While the magical machine can read and understand screen-shots it cannot infer which elements of a screen-shot are static and which ones change according to the user's action. This should be part of your description.*

The following tasks cover a subset of the functionality of the web application. Try to describe the behavior of the application *in detail* for each one of the tasks. On your paper please refer to the task number.

1. Login with the user-ID “marym” and the (wrong) password “sesamo”, then with the (correct) password “mapa”. Describe how the web application behaves.
2. Play with “<<previous page” and “next page >>” (on the bottom). Describe what the elements on the screen show, in particular the ones that change dependent on the user and the user’s actions.
3. Note the availability of “add new member”, “edit” and “delete”. Now logout and login with the user-ID “johnd” and the password “papajohn”. Note that the three functions are no longer available. Describe how the web application provides different functionality to different user-ID’s (note that the users “chrism” and “susi” get the same functionality as “marym”)
4. Add a new member (just make up some data). Assume you do *not* have an e-mail address. Continue with “OK”. Now enter an e-mail address. Continue with “OK”. Describe how the web application behaves.
5. Sort the members by first name or last name. Describe how the web application behaves.
6. Search for a member with the last name “Miller”. Describe how the web application behaves.
7. Delete a member (you pick). Describe how the web application behaves.

D.3 Screen Labeling Example

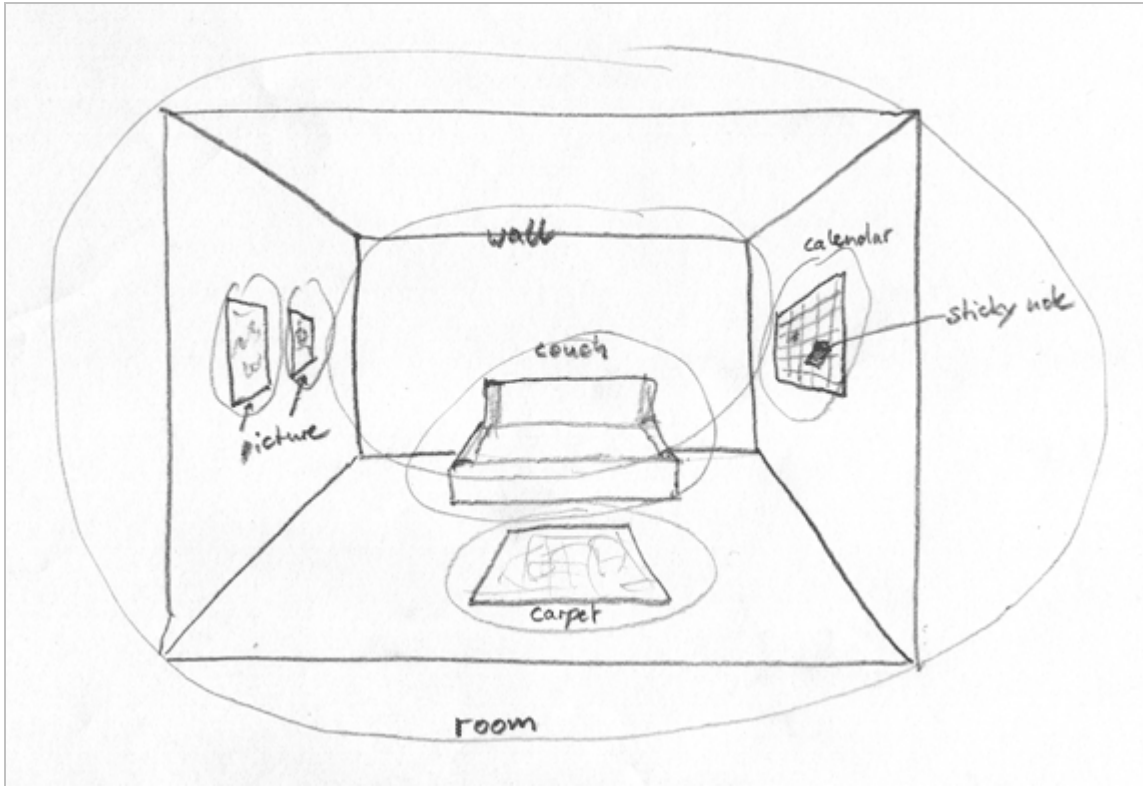


Figure 31: Screen labeling example provided to study participants

D.4 Screenshots of Example Application

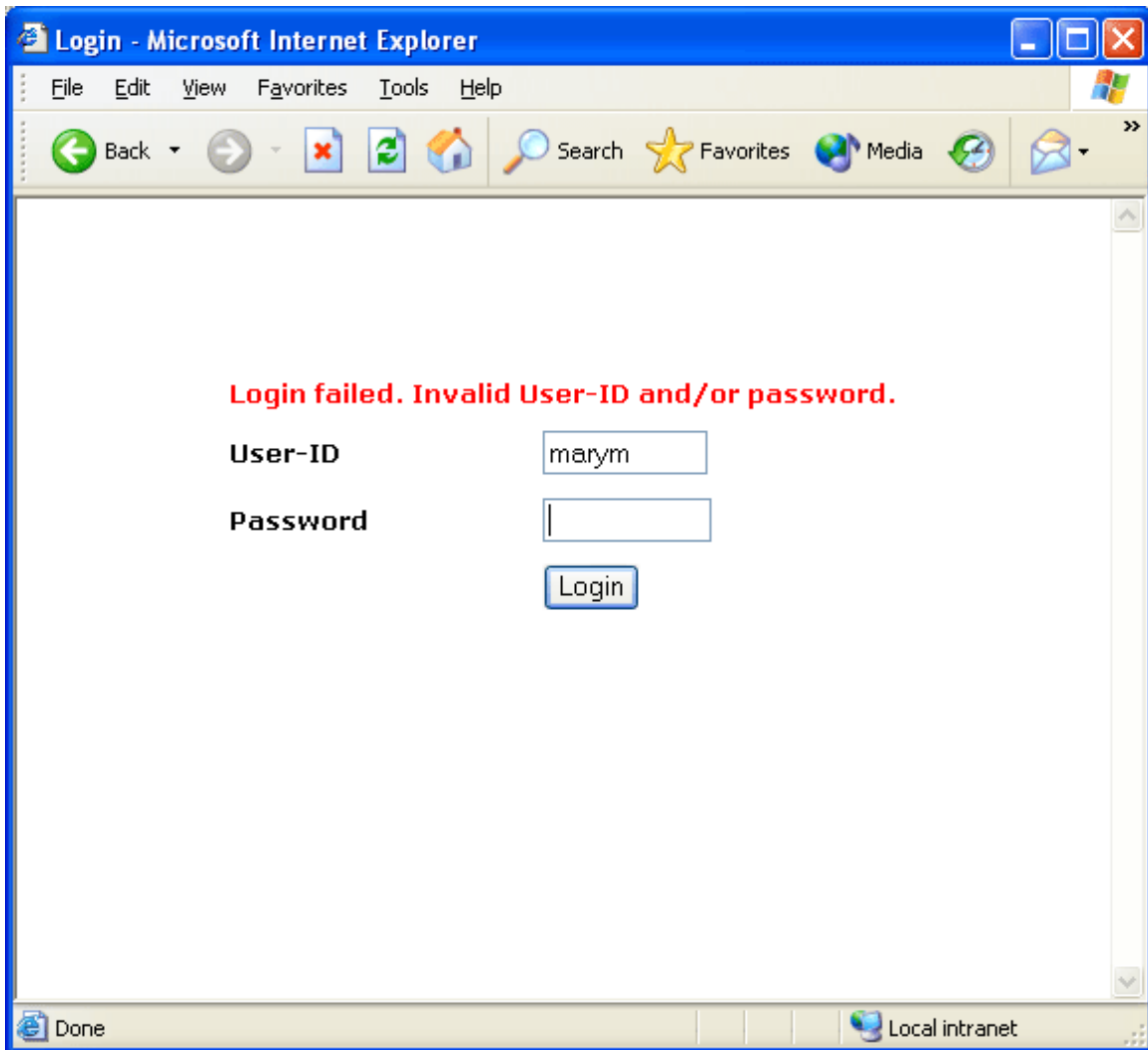


Figure 32: Screenshot of example application from MMODELS-1: Login

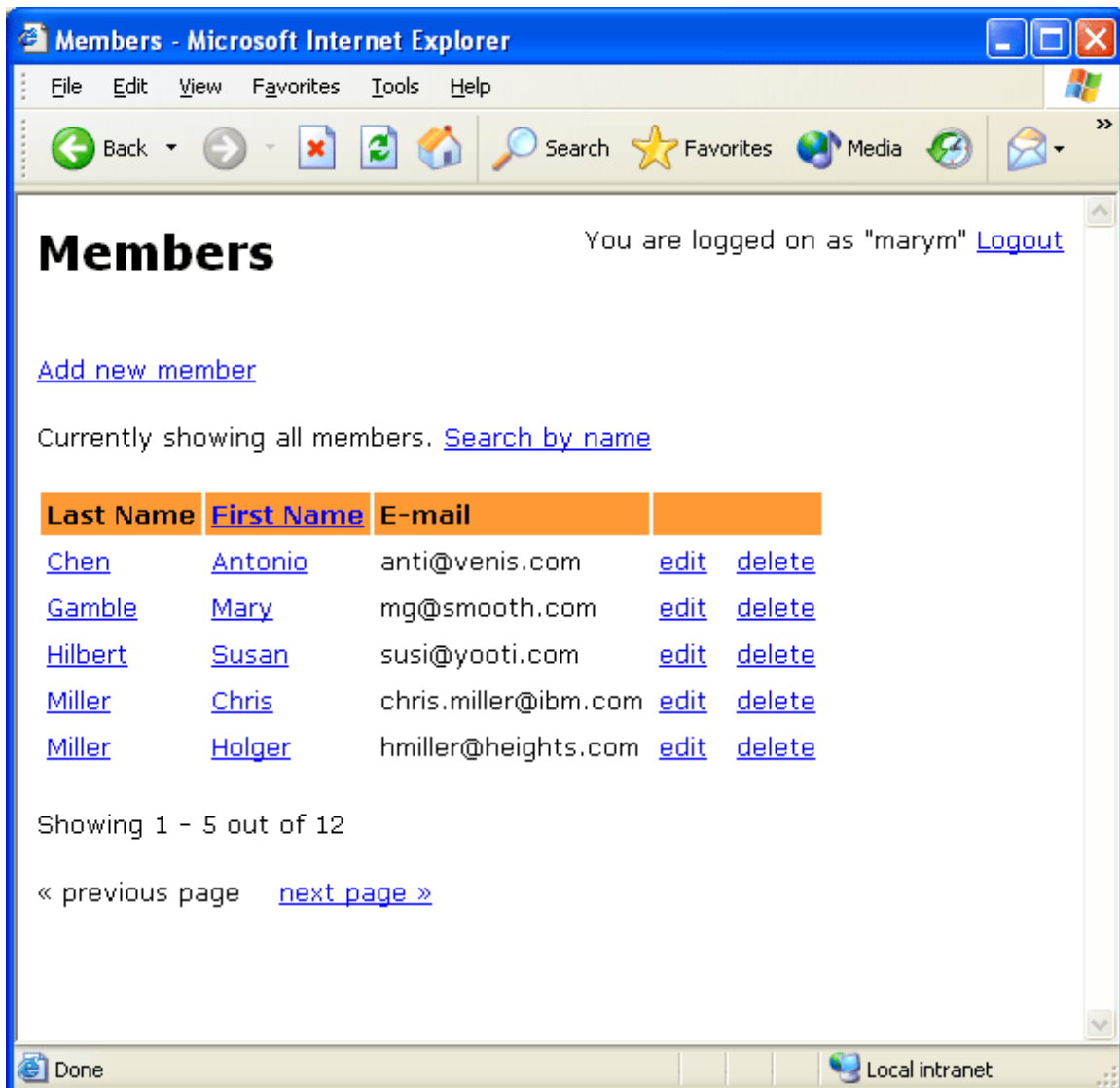


Figure 33: Screenshot of example application from MMODELS-1: View all members

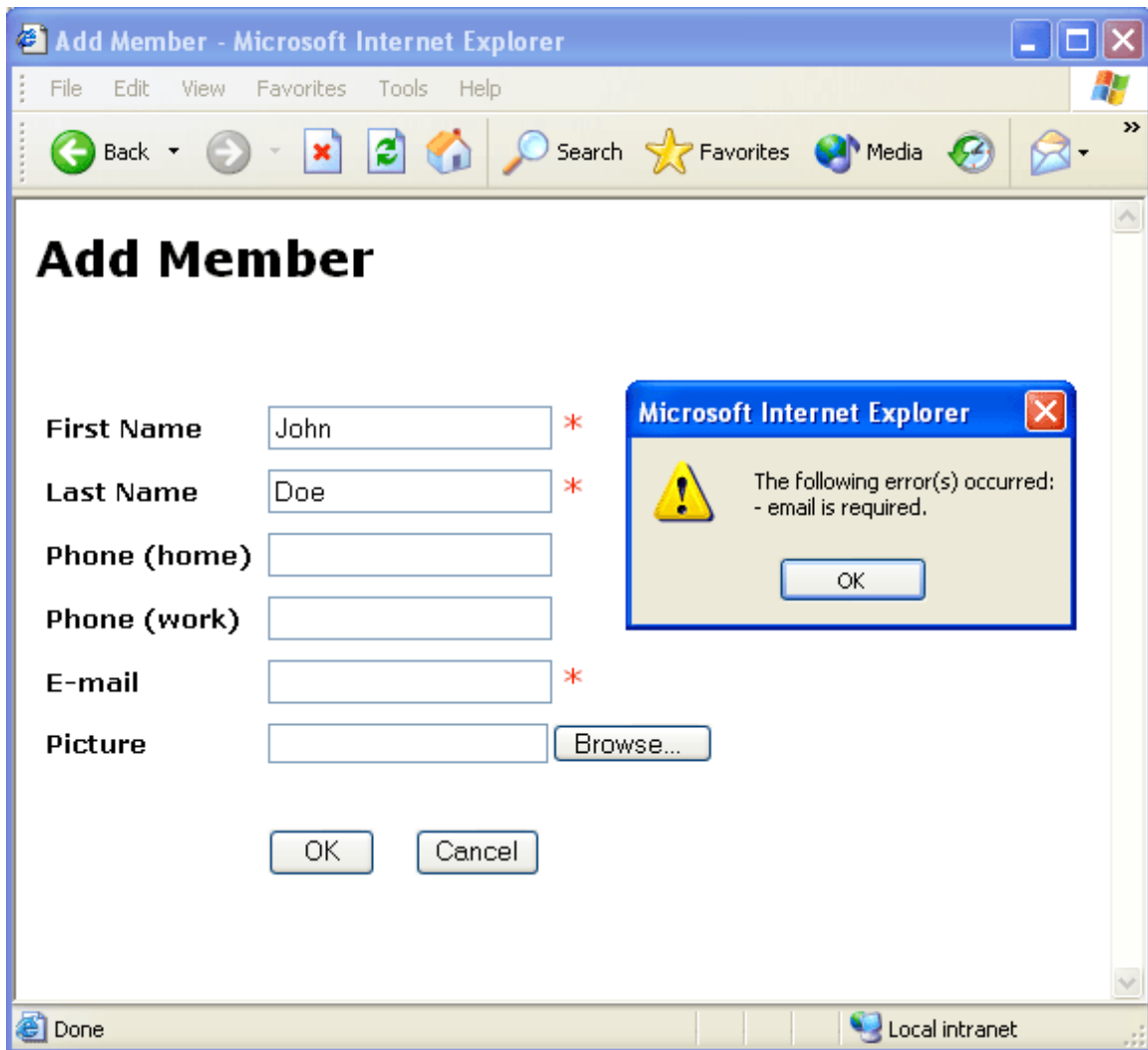


Figure 34: Screenshot of example application from MMODELS-1: Add member

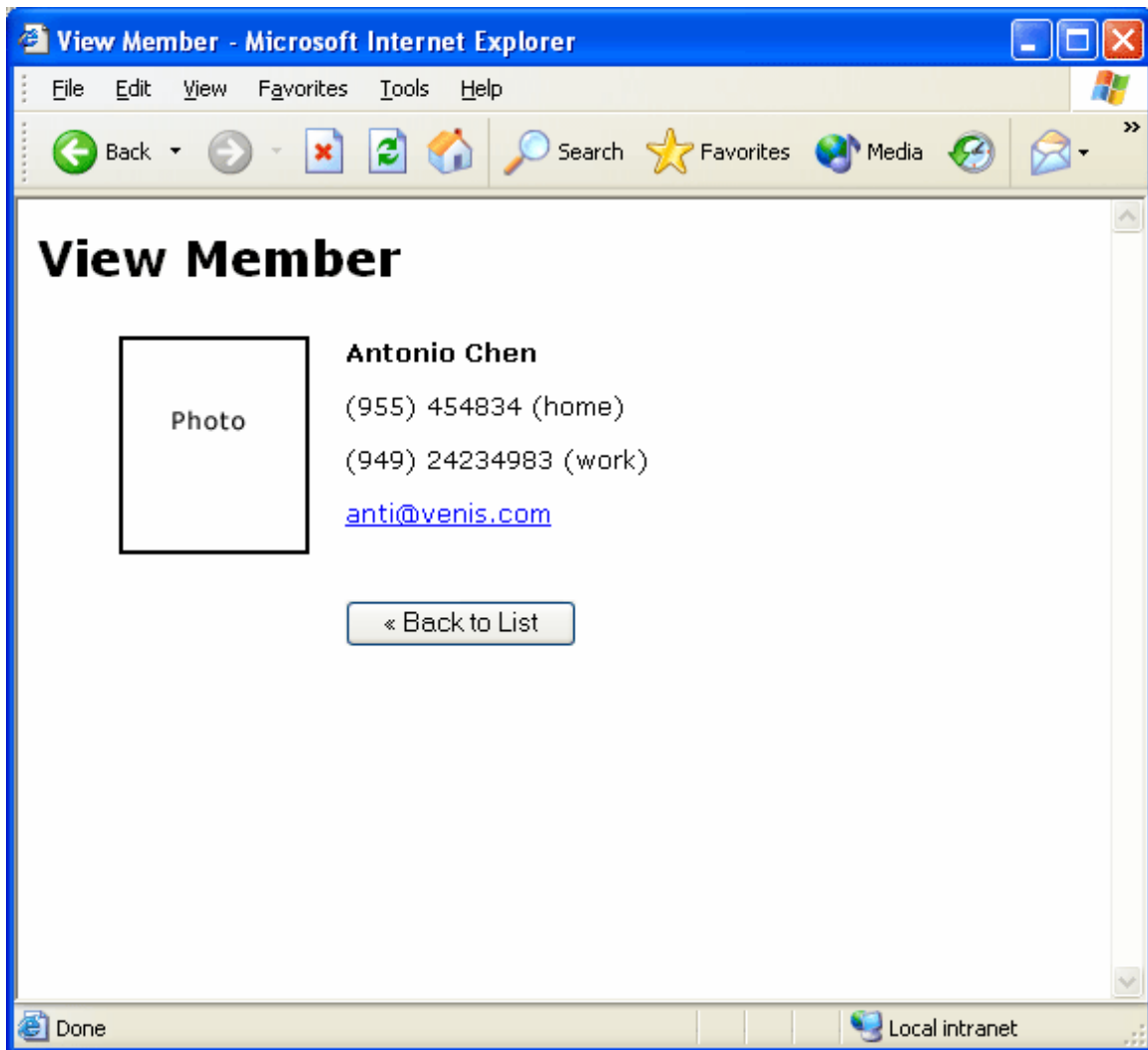


Figure 35: Screenshot of example application from MMODELS-1: View member

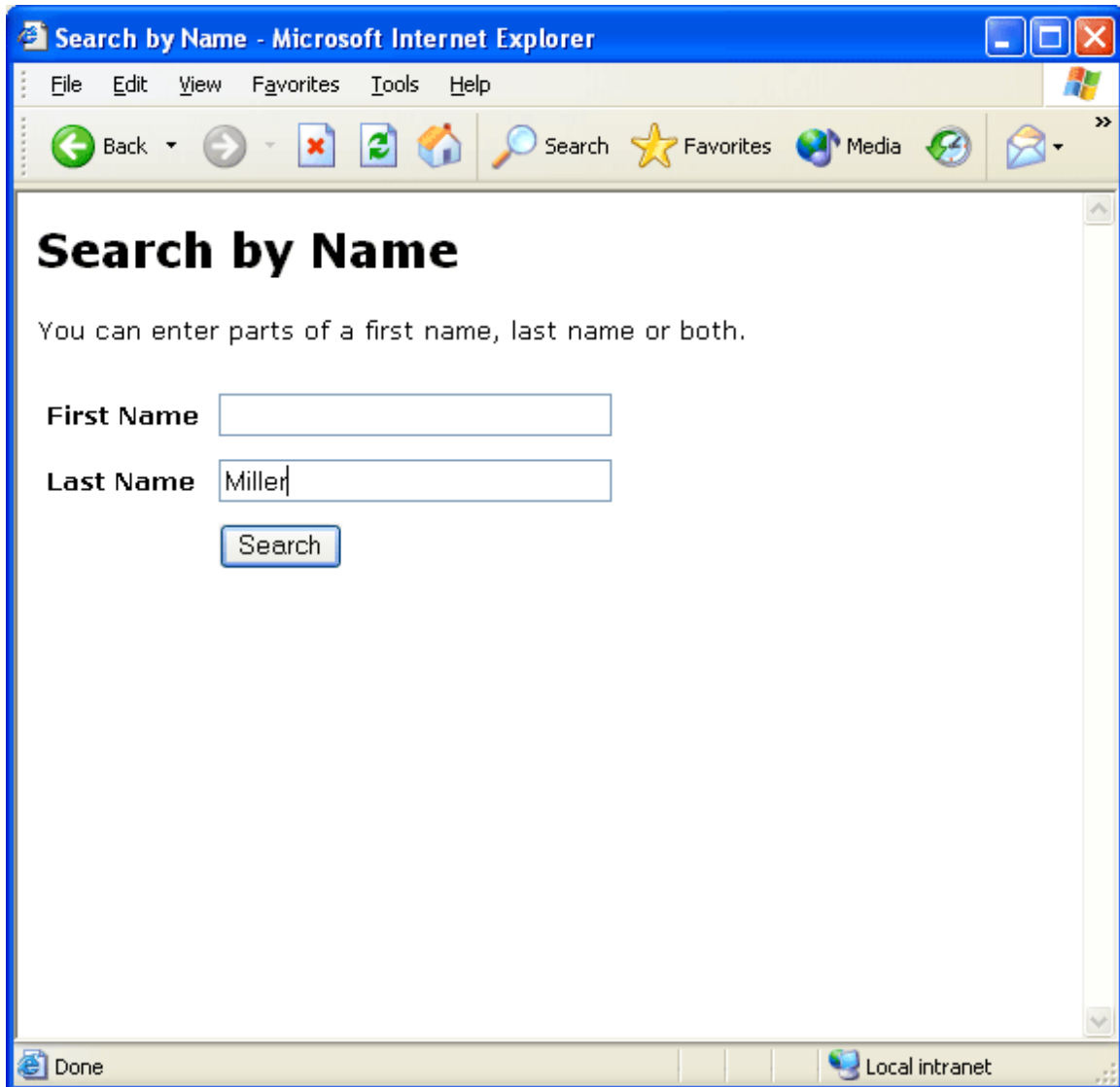


Figure 36: Screenshot of example application from MMODELS-1: Search

Appendix E Mental Models Study 2

E.1 IRB Approval

 <p>Virginia Tech VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY</p>	<p>Institutional Review Board Dr. David M. Moore IRB (Human Subjects) Chair Assistant Vice Provost for Research Compliance CVM Phase II - Duckpond Dr., Blacksburg, VA 24061-0442 Office: 540/231-4991; FAX: 540/231-6033 e-mail: moored@vt.edu</p>
<p>June 24, 2003</p>	
<p>MEMORANDUM</p>	
<p>TO:</p>	<p>Mary Beth Rosson Carroll Computer Science 0106 Jochen Rode CSA 0106</p>
<p>FROM:</p>	<p>David M. Moore </p>
<p>SUBJECT:</p>	<p>Expedited Approval – “Nonprogrammer Web Application Development”– IRB # 03-315</p>
<p>This memo is regarding the above-mentioned protocol. The proposed research is eligible for expedited review according to the specifications authorized by 45 CFR 46.110 and 21 CFR 56.110. As Chair of the Virginia Tech Institutional Review Board, I have granted approval to the study for a period of 12 months, effective June 19, 2003.</p>	
<p>Approval of your research by the IRB provides the appropriate review as required by federal and state laws regarding human subject research. It is your responsibility to report to the IRB any adverse reactions that can be attributed to this study.</p>	
<p>To continue the project past the 12 month approval period, a continuing review application must be submitted (30) days prior to the anniversary of the original approval date and a summary of the project to date must be provided. My office will send you a reminder of this (60) days prior to the anniversary date.</p>	

Figure 37: IRB approval for mental models study 2

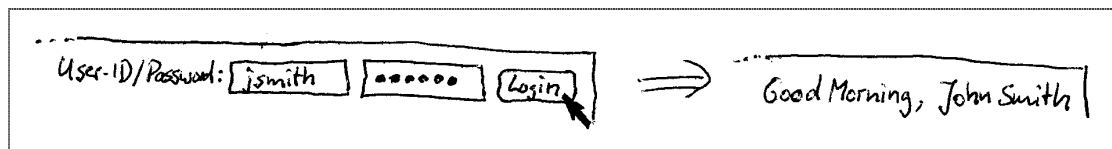
E.2 Scenarios

Table 33: Nine scenarios from MMODELS-2

Imagine the following scenario...

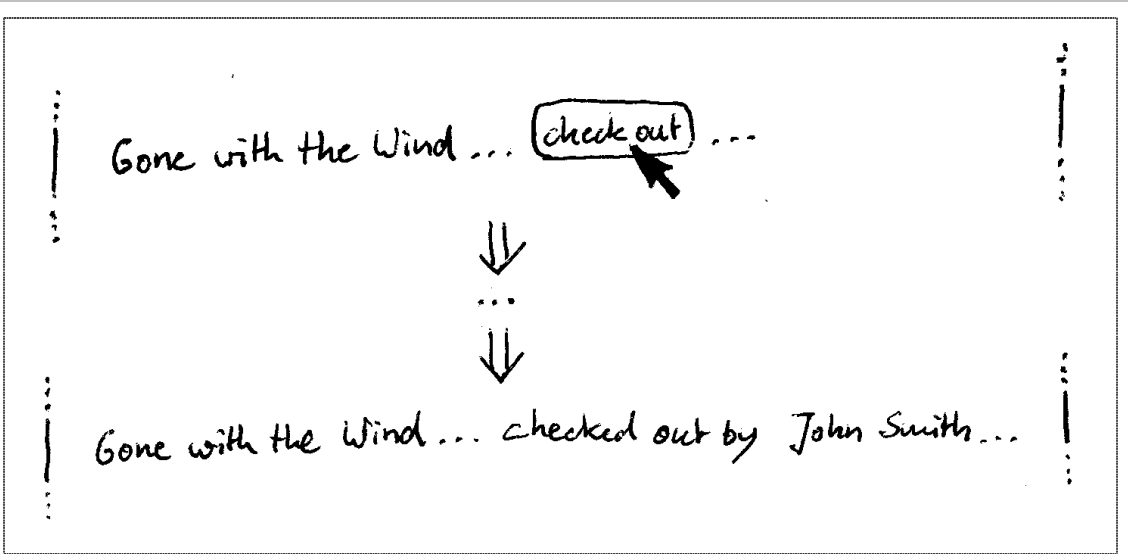
A university has a video library that lends movies to its students. The library has a web-based information system that offers basic functions like catalog browsing, searching, listing of videos checked-out by a patron, etc.

In the following you will be asked a series of questions regarding how *you think* certain features of the web site work *behind the scenes*. Please do not be concerned that you have no training as a web programmer. This is not a test. Just tell us what you think. There are no right or wrong answers.



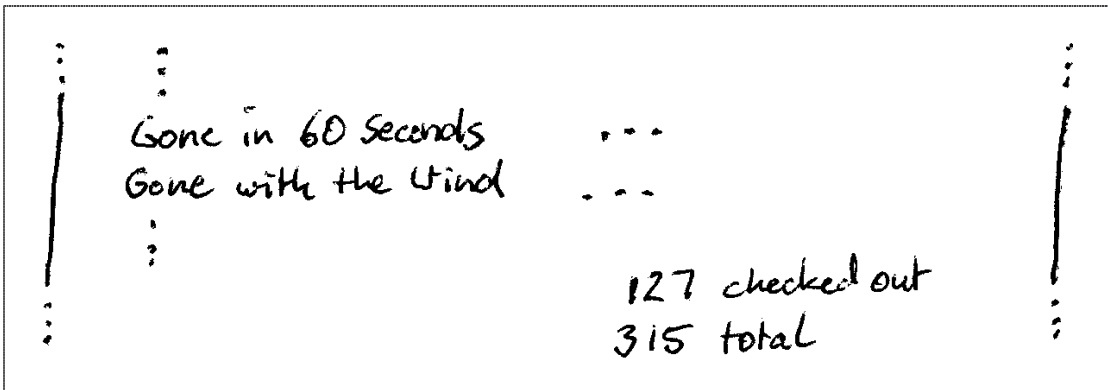
1) After logging in with your user-ID the web site always shows your full name and a logout button in the upper right corner.

- a) What do you think the web site must do to keep track of the fact that you are logged in even though you go from page to page?
- b) What do you think the web site must do to show your full name, although you only entered a short user-ID? Take the user-ID “jsmith” as an example and show step-by-step how the web site determines the name “John Smith”.
- c) Note that the library home page only displays your name when you are logged in. If you are not logged in, it shows a login box instead. How do you think this feature works behind the scene?



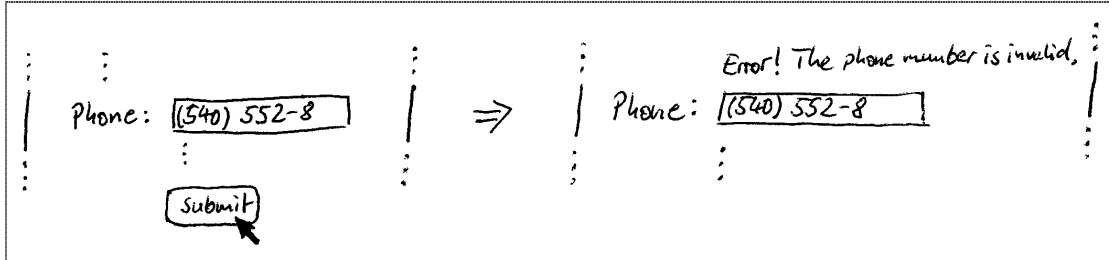
2) When you check out a movie, the librarian enters this fact into the web site and the web site “remembers” that you did.

- a) What do you think the web site must do to keep track of the fact that you checked-out a particular video?
- b) What specific pieces of information does the website store about each video in the library?
- c) In what form and format do you think the web site keeps record of the checked-out videos?
- d) Consider the case where there are two copies of a one movie. What do you think the web site must do to keep track of the fact that one of the movies is checked out by you but the other one is still available?



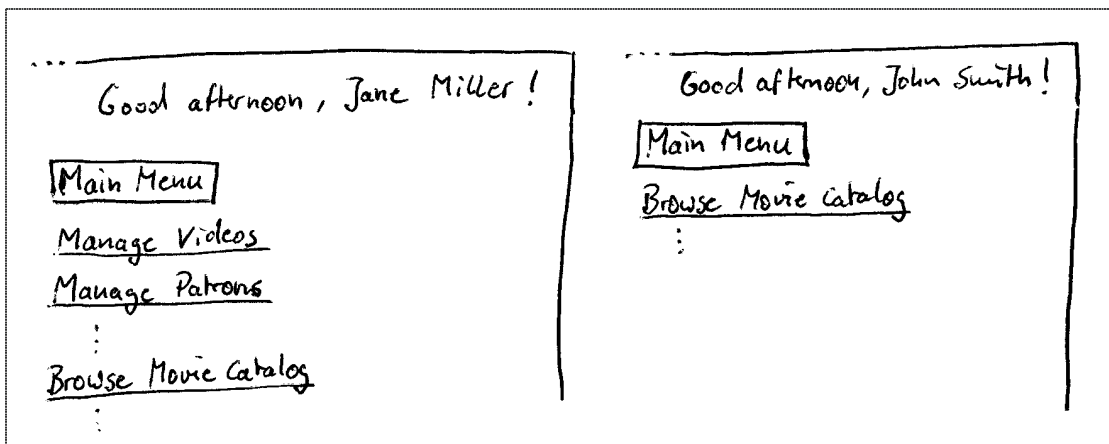
3) The web site displays the total number of checked-out videos.

- a) What do you think the web site must do to determine this number?



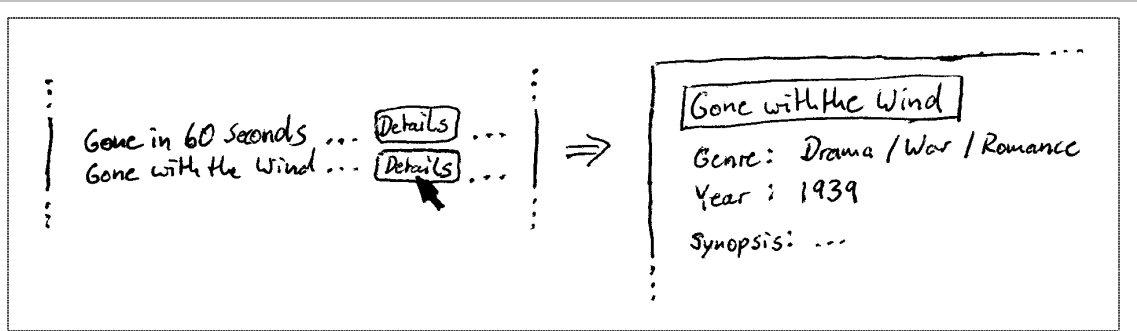
4) If the librarian accidentally enters an invalid telephone number (e.g. too short) while registering a new patron an error-message is displayed.

- What do you think the web site must do to determine whether or not the telephone number is invalid?
- At what point in time do you think the web site checks whether the telephone-number is valid? Why at this time?
- How does the web site know where to display the error message?



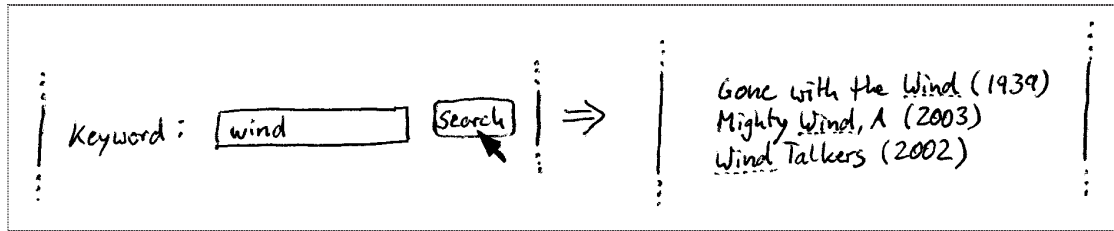
5) Access to particular sections of the web site is restricted to particular users. For example, everyone can login to see which videos they have checked out. But only librarians can login and get a report of *all* checked-out videos.

- How do you think the web site keeps track of the users who are allowed to login?
- How do you think the web site keeps track on which user is allowed to see which part of the web site?
- How do you think the web site checks whether or not your user-ID and password are correct?
- What specific pieces of information does the website store about each user in the library?



6) In the tabular list of available videos there is a little button titled “Details” right besides each individual video. If clicked, a new screen comes up showing such details as actors, genre, synopsis et cetera.

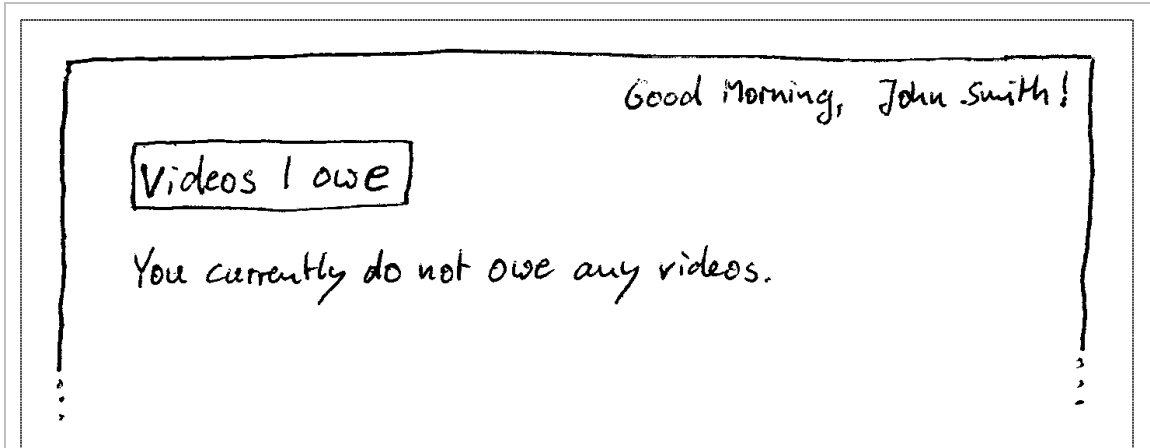
a) How do you think the web site knows which movie’s details it should display on the new screen? How does it associate a particular “Details” button with a particular movie?



7) The user can enter one or more keywords and the web site will show all videos that match.

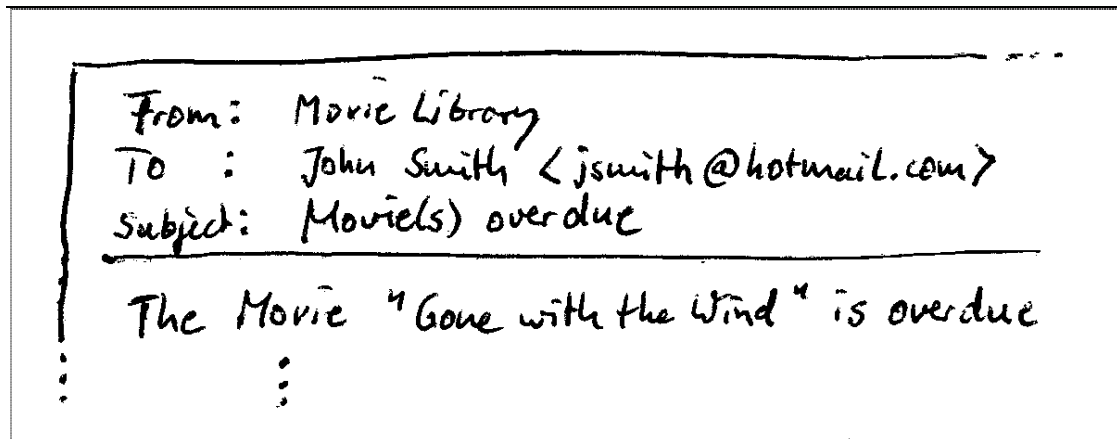
a) How does the web site determine which movies to list? How does it determine whether a movie matches the entered keywords?

b) Imagine there is another search field which allows you to also restrict the age of the movie (e.g. all movies younger than year 1998). How would the web site determine which movies to list if it had to check for keyword and age?



8) Once a person returns a checked-out movie, the movie no longer shows on the "Videos I owe" screen.

- How does the web site keep track of the fact that the movie has been returned to the library?
- How does the web site determine which movies are owed by the currently logged-on user?



9) The web site sends a reminder e-mail if the video is over-due.

- How do you think the web site knows when a video is over-due?
- How does the web site determine the e-mail address of the person who checked out the movie?

Appendix F Click

F.1 Screenshots

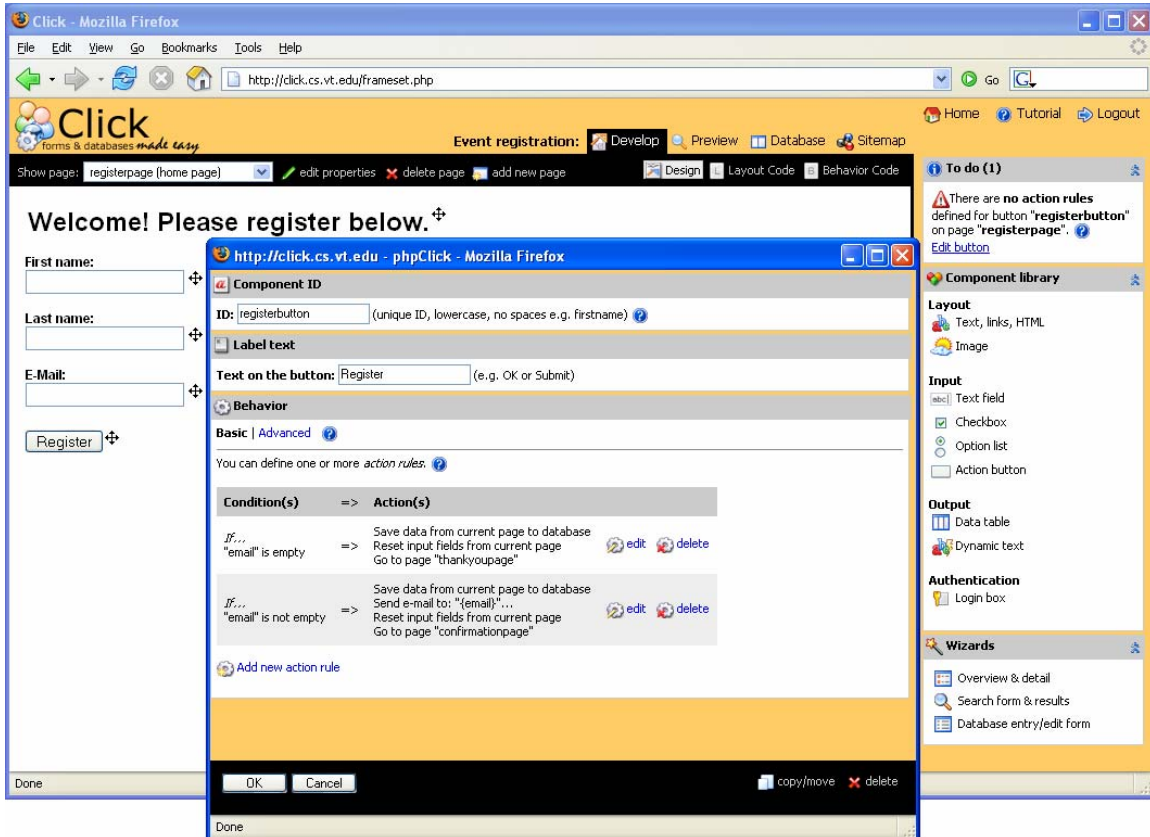


Figure 38: Defining a “Register” button and associated action using the form-based UI of Click

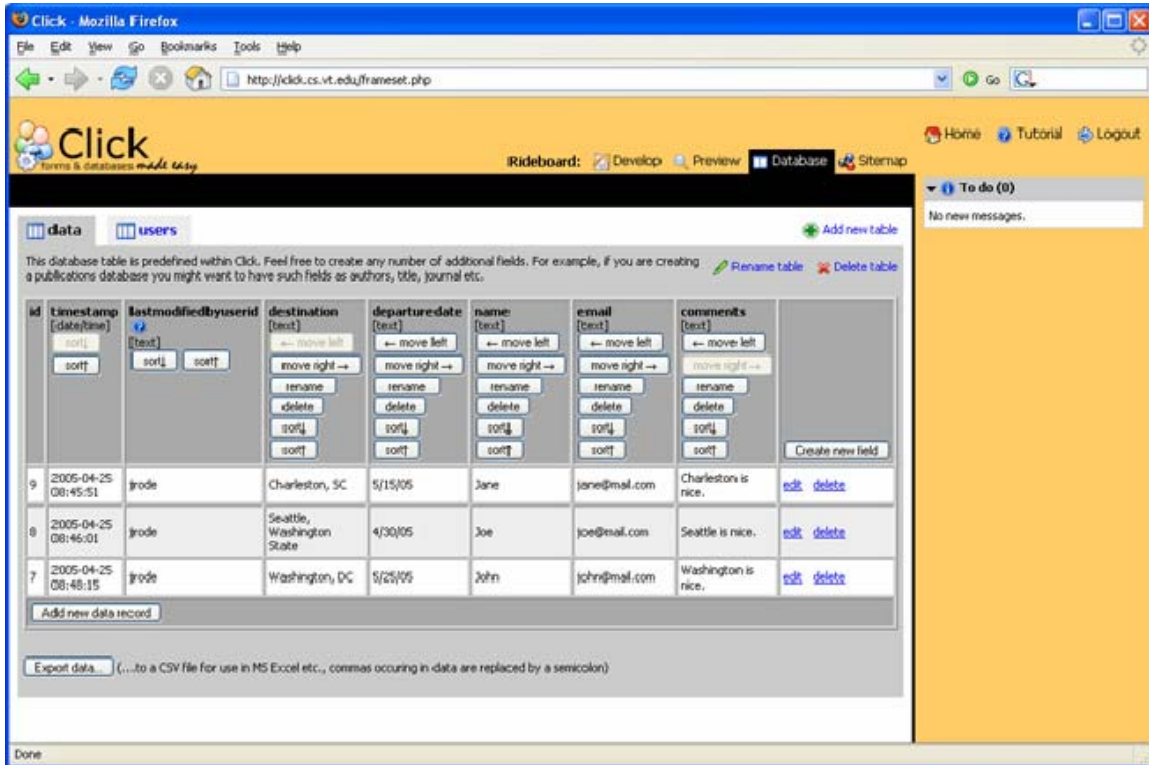


Figure 39: The "Database" view of Click allows the modification of database schema and data

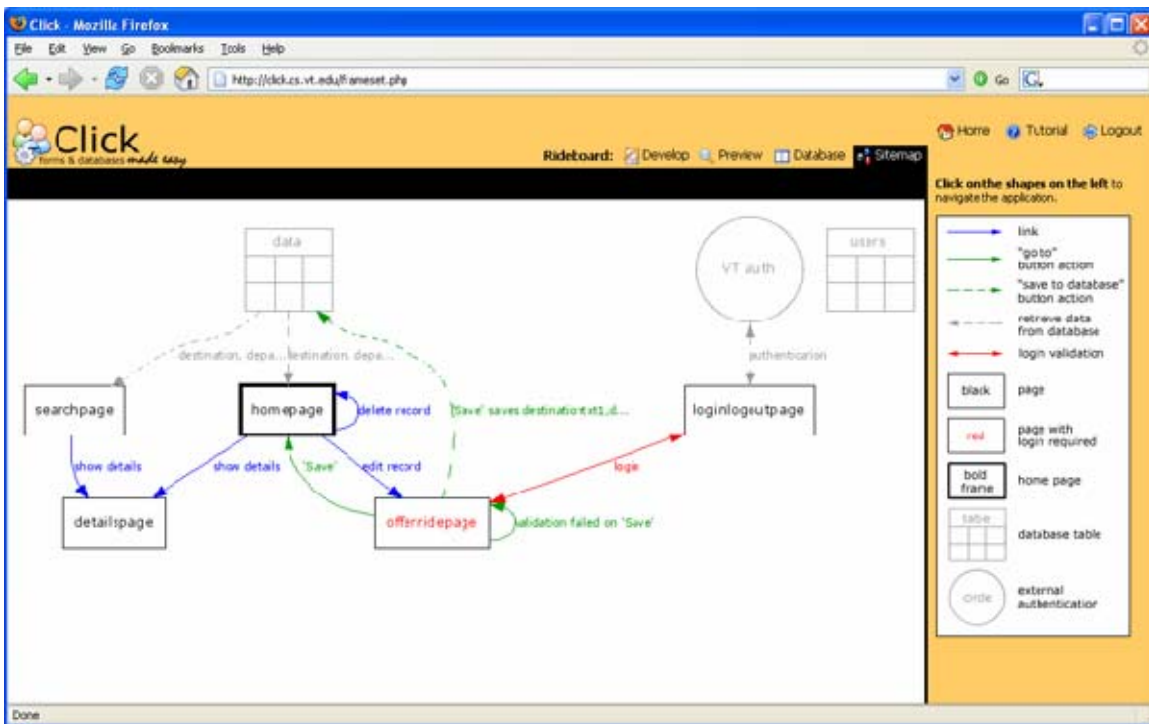


Figure 40: The "Sitemap" view of Click showing the example "Ride board" application

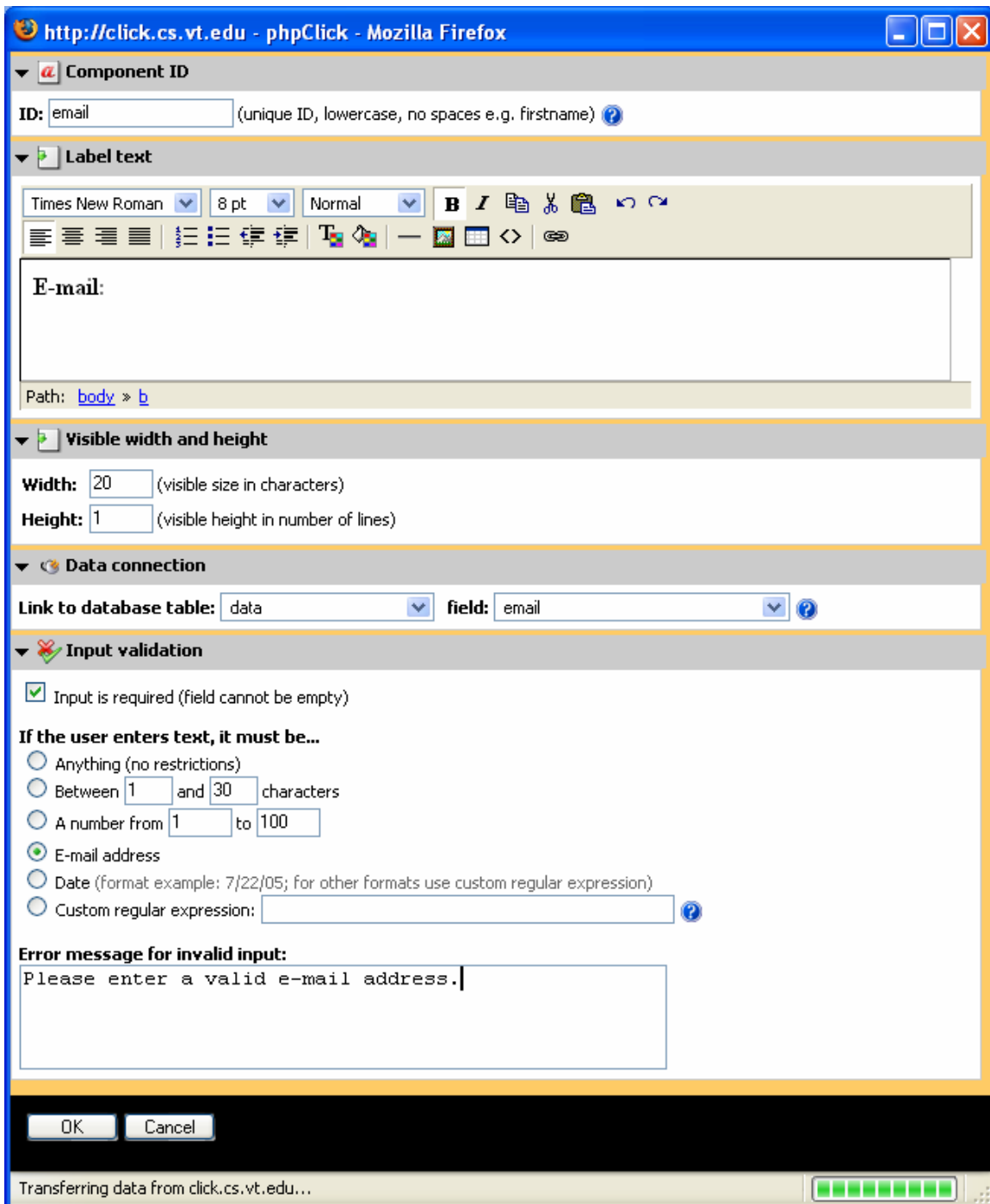


Figure 41: The property dialog of the "Text field" component

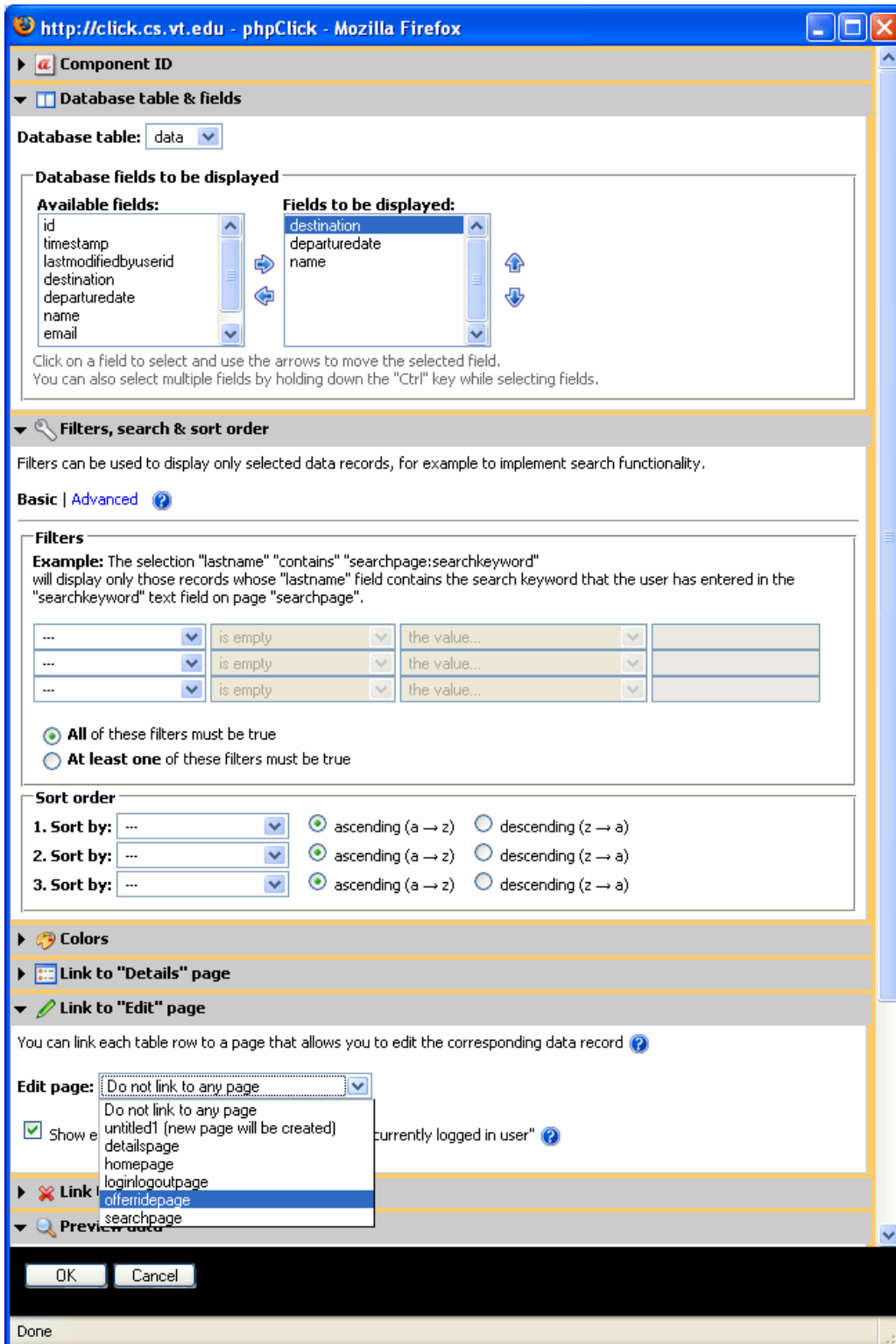


Figure 42: Parts of the properties dialog of the "Dynamic table" component

http://demo.hosting.vt.edu - phpClick - Mozilla Firefox

Define conditions under which the actions will be performed

Always

Only if:

Input Field

homepage.checkbox1 is checked

...

...

All of these conditions must be true

At least one of these conditions must be true

Define actions to be performed when the conditions are met

Save data to the database...

from the input fields on the current page

from all input fields that are part of the application

Send email...

You can use these place-holders (including the braces or brackets) to represent the current value of...

input fields (in curly braces):

{firstname} {lastname} {email} {checkbox1}

From: webmaster@example.com

To: {email}

Subject: Conference registration

Message: Dear {firstname},
Thank you for your registration!
...|

Reload current page (May be used to refresh the search results in a table on the current page)

Clear input fields...

on the current page

on every page that is part of the application

Go to page...

untitled1 (new page will be created)

Done

Done

Figure 43: Dialog to specify action rules

Appendix G Summative Evaluation of Click

G.1 IRB Approval



 <p>Virginia tech VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY</p>	<p>Institutional Review Board</p> <p>Dr. David M. Moore IRB (Human Subjects) Chair Assistant Vice President for Research Compliance CVM Phase II, Duckpond Dr., Blacksburg, VA 24061-0442 Office: 540/231-4991; FAX: 540/231-6033 email: moored@vt.edu</p>
<p>DATE: April 8, 2004</p>	
<p>MEMORANDUM</p>	
<p>TO: Manuel A. Perez-Quinones Computer Science 0106 Iochen Rode CSA 0106 Yogita Bhardwaj</p>	
<p>FROM: David Moore </p>	
<p>SUBJECT: IRB Expedited Continuation: "Nonprogrammer Web Application Development" IRB # 05-275 ref 04-577</p>	
<p>This memo is regarding the above referenced protocol which was previously granted expedited approval by the IRB on June 19, 2004. The proposed research is eligible for expedited review according to the specifications authorized by 45 CFR 46.110 and 21 CFR 36.110. Pursuant to your request of last week, as Chair of the Virginia Tech Institutional Review Board, I have granted approval for extension of the study for a period of 12 months, effective as of June 19, 2004.</p>	
<p>Approval of your research by the IRB provides the appropriate review as required by federal and state laws regarding human subject research. It is your responsibility to report to the IRB any adverse reactions that can be attributed to this study.</p>	
<p>To continue the project past the 12-month approval period, a continuing review application must be submitted (30) days prior to the anniversary of the original approval date and a summary of the project to date must be provided. Our office will send you a reminder of this (60) days prior to the anniversary date.</p>	
<p>Virginia Tech has an approved Federal Wide Assurance (FWA00000572, exp. 7/20/07) on file with OHRP, and its IRB Registration Number is IRB00000667.</p>	

Figure 44: IRB approval for summative evaluation of Click

G.2 IRB Informed Consent

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
Informed Consent for Participants in Research Projects Involving Human Subjects

Title of Project: Nonprogrammer Web Application Development

Investigators: Dr. Manuel Pérez Quiñones, Jochen Rode, Yogita Bhardwaj

I. Purpose of this Research

We explore ways to empower end-users without any or much prior programming experience in building interactive, database-driven websites. The purpose of this study is to evaluate a prototype for an end-user web development tool.

II. Procedures

Participants will be asked to evaluate a prototype of a web development tool and reply to an evaluation questionnaire. We will voice and screen actions for later analysis.

III. Risks

There is no more than minimal risk to you if you choose to participate in this study.

IV. Benefits

There are no direct benefits to the participants beyond the compensation of \$30.

V. Extent of Anonymity and Confidentiality

All data that we will collect will be kept confidential and only be used by the investigators. The written records, voice and screen recordings will be stored in a secured location. Before publishing study results we will remove any information that would identify a person.

VI. Compensation

Each participant will receive a compensation of \$30 for participating in the study.

VII. Freedom to Withdraw

Participants are free to withdraw from the study at any time.

VIII. Approval of Research

This research project has been approved, as required, by the Institutional Review Board for Research Involving Human Subjects and by the Department of Computer Science at Virginia Polytechnic Institute and State University

IX. Subject's Responsibilities
I voluntarily agree to participate in this study.

X. Subject's Permission
I have read and understand the Informed Consent and conditions of this project. I have had all my questions answered. I hereby acknowledge the above and give my voluntary consent:

_____ Date _____

Participant's signature _____

Should I have any questions about this research or its conduct, I may contact:

Jochen Rode Investigator	540-231-6140 / jrode@vt.edu
Yogita Bhardwaj Investigator	540-231-7542 / yogitab@vt.edu
Dr. Manuel Pérez Quiñones Investigator, Faculty Advisor	540-231-2646 / perez@cs.vt.edu
David M. Moore Chair, IRB Office of Research Compliance Research & Graduate Studies	540-231-4991 / moored@vt.edu

Figure 45: IRB informed consent form for summative evaluation of Click

G.3 Online Screening Questionnaire

Table 34: Online screening questionnaire for formative and summative studies of Click



CLICK - A tool for creating surveys, online forms and databases

Do you like survey.vt.edu? If so, read on! If not, read on too as things may get better! :-)

We are in the process of developing a new web-based tool that will help users at Virginia Tech to create not only basic surveys (as can be done with survey.vt.edu) but also more complex multi-page online forms (e.g. a tutorial registration system or service request forms) and databases (e.g online staff directory or plant pathology database).

This tool is primarily targeted at people who don't consider themselves programmers but still have the need for certain interactive and advanced features on their websites. The first version of the tool is scheduled to be released for the upcoming spring semester.

If you are interested in such a tool, please help us make it fit your needs by evaluating and criticizing the current prototype.

Your input can make the difference!

If you are willing to spend between 45-60 min to test and criticize the current prototype please fill in the form below. The evaluation sessions will take place in Torgersen Hall (the building connected to the library via the bridge) or McBryde Hall (across the street from Torgersen Hall). Filling in the form is not a commitment for participation. However, if you do participate you may be compensated with chocolate. :-)

Please tell us what you might use such a tool for, what you expect in terms of tool functionality and ease-of-use.

Please choose the group(s) that best describes you.

VT Staff

VT Student

VT Faculty

VT Alumni

Other:

How do you rate your knowledge of your primary visual web development tool (Frontpage, or Dreamweaver, GoLive etc.)?

(1=no knowledge, 5=expert knowledge)

1 2 3 4 5 I don't use a visual tool

How do you rate your knowledge of HTML?

(1=no knowledge, 5=expert knowledge):

1 2 3 4 5

How do you rate your knowledge in web programming (use of Javascript, PHP, ASP, or Java etc.)?

(1=no knowledge, 5=expert knowledge)

1 2 3 4 5

How do you rate your knowledge in databases (MS Access, or Oracle, MySQL etc.)?

(1=no knowledge, 5=expert knowledge)

1 2 3 4 5

Please give us your name, department and e-mail address.

Your Name:

Your Department:

Your E-mail:

G.4 Study Procedure Instructions

Table 35: Participant's instructions for summative study of Click

Instructions

The goal of our research is to explore ways to empower end users for building interactive, database-driven websites. The purpose of this study is to evaluate Click, a prototype for an end-user web development tool.

During the next 2 hours you will use Click to develop a basic web application, trying to replicate an online ride board which will be provided as an example.

First, you will watch a 14-minute video that introduces you to Click's main concepts and features.

Next, you can explore the example ride board application in order to experience the functionality that you will later replicate using Click. You will get a small instruction sheet to help you discover and understand the functionality.

Then, you will begin developing your own ride board application in Click while we will observe and take note of the problems and bugs you discover in the software.

Finally, once you have finished creating the ride board application, we will ask you to fill in a questionnaire to get your opinion on some of the features you have experienced.

Please remember that *we don't test you* but Click, the prototype tool. We are trying to find out how it can be improved in terms of functionality and ease-of-use.

Before we get started, do you have any questions?

G.5 Ride board Example Application Screenshots



Figure 46: Screenshot of ride board example application: Home

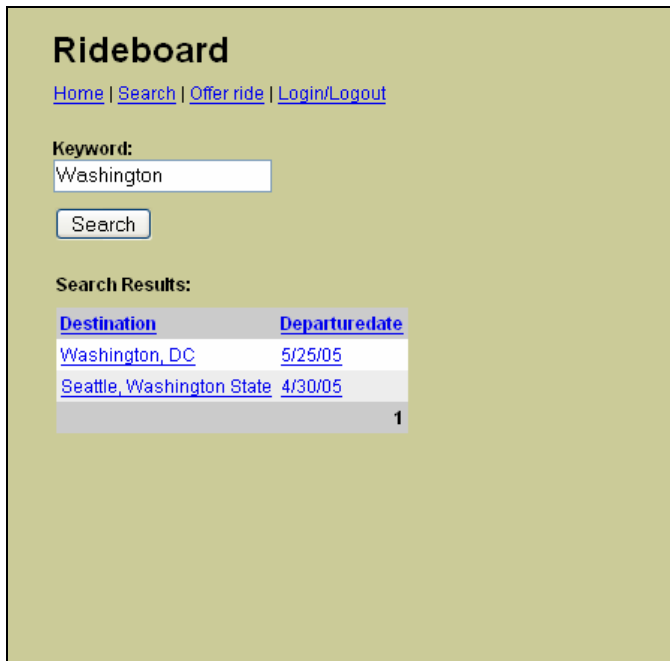


Figure 47: Screenshot of ride board example application: Search

The screenshot shows a web form titled "Rideboard" with a light green background. At the top, there are navigation links: [Home](#), [Search](#), [Offer ride](#), and [Login/Logout](#). The form contains several input fields with associated labels and error messages:

- Destination:** A text input field with the value "Please enter between 1 and 100 characters." displayed in red text to its right.
- Departure date:** A text input field with the value "today" and a red error message: "Please enter a date in the format: 7/22/05".
- Name:** An empty text input field.
- Email:** A text input field with the value "john at mail.com" and a red error message: "Please enter a valid e-mail address."
- Comments:** A large, empty text area.

At the bottom of the form is a "Save" button.

Figure 48: Screenshot of ride board example application: Offer ride

The screenshot shows a web form titled "Rideboard" with a light green background. At the top, there are navigation links: [Home](#), [Search](#), [Offer ride](#), and [Login/Logout](#). The form contains a login section:

- The text "Nobody is logged in." is centered above the input fields.
- User-ID:** A text input field containing the value "jsmith".
- Password:** A text input field with masked characters (asterisks).
- A "Login" button is positioned below the password field.

Figure 49: Screenshot of ride board example application: Login/Logout

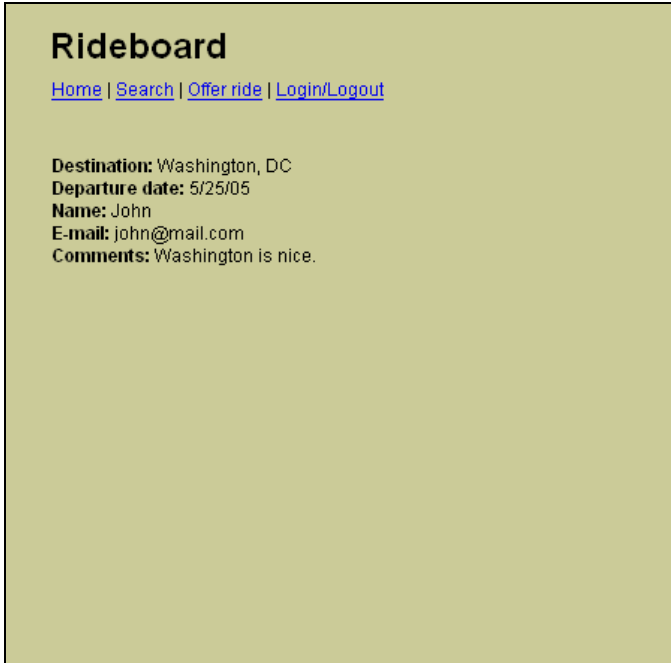


Figure 50: Screenshot of ride board example application: Details

G.6 Ride board Example Exploration Instructions

Table 36: Click - summative study: Ride board example exploration instructions

Example ride board application: Guidance

Imagine the ride board application to be used by all people at Virginia Tech to advertise/offer car rides to other people or search for available rides.

Take some time to try out and explore the following functionality:

- 1) Look at the *Home* page which lists the destination and date of all posted rides in tabular format
- 2) Go to the *Search* page.
Try to search for “Washington”.
Note that all records are listed that have the keyword appear somewhere in the destination text.
- 3) Go to the *Offer ride* screen. Use your Virginia Tech PID and password to log in.
Note that as soon as you have logged in you can enter a new ride.
Just press “Save” without entering any data.
Note the three error messages.
Now enter some test data and click “Save” again.
- 4) Note that you have returned to the *Home* page which now shows your new ride.
Also, note that there is an “edit” and “delete” link next to the ride you have just entered.
- 5) Click on the “edit” link and change the destination text.
- 6) Go to the *Login/Logout* page.
Click on “Logout”
Return to the *Home* page.
Note how the “edit” and “delete” link have disappeared.
- 7) Click on the ride you have recently entered.
Note how it shows all the details of the ride on another page.
Return to the Home page.
Click on some other ride. Note how it shows all the details of the other ride.

Take some more time to explore the application until you feel that you really understand what it does. Don't worry, you can refer back to the example application at any point in time; so don't make an effort to memorize anything.

G.7 Help Instructions

Table 37: Click - summative study: Help instructions

Help instructions

Now you are ready to try to replicate the ride board application using Click. Try to get as close to the example as possible.

Also, try to do so without asking us for help. Just pretend you are alone.

Please only ask for help when you feel “completely lost” and we will assist you.

If you get stuck, first try to find a way out by yourself, referring to the online help icons, Click’s To-do list or the Sitemap. If, after a minute or so, you still don’t know how to proceed, please ask.

We will record all screen actions and our discussion for later analysis.

Do you have any more questions before we get started?

G.8 Facilitators’ Functionality Checklist

Table 38: Click - summative study: Facilitators' functionality checklist

Ride board: Functionality Check List

1. Headline and navigation links show up on EVERY page
2. *Home*: Table shows Destination & Departuredate
3. *Home*: Table shows “delete” link ONLY for currently logged in user
4. *Home*: Table shows “edit” link ONLY for currently logged in user
5. *Home*: “edit” link goes to *Offer ride* page
6. *Home*: Each table row is linked to *Details* page
7. *Search*: Table shows Destination & Departuredate
8. *Search*: Searches Destination field
9. *Search*: Each table row is linked to *Details* page
10. *Offer rides*: Login protected (all users at Virginia Tech)
11. *Offer rides*: Text fields for destination, departuredate, name, email, comments
12. *Offer rides*: Input validation for destination, departuredate, and email
13. *Offer rides*: Clicking “Save” saves to database
14. *Offer rides*: Clicking “Save” sends user to page *Home*
15. *Login/Logout*: Contains Login box
16. *Login/Logout*: is defined as the application’s “login” page
17. *Details*: Shows details of destination, departuredate, name, email, comments

G.9 Post-study Survey Questionnaire

Table 39: Click - summative study: Post-study questionnaire

Click Feedback (Please be brutally honest!)

(1) While designing the application I felt "completely lost" and did not know how to proceed.
(1=hardly ever, 5=very often)
 1 2 3 4 5

(2a) I looked at the "To do" list (1=hardly ever, 5=very often).
 1 2 3 4 5

(2b) When I did look at the "To do" list I found it useful.
(1=strongly disagree, 5=strongly agree)
 1 2 3 4 5 not applicable

(2c) How did the "To do" list affect the way you worked on the application?
Can you remember any cases where it was particularly useful or distracting?
Do you have ideas for how to improve it?

(3a) I looked at the Sitemap (1=hardly ever, 5=very often).
 1 2 3 4 5

(3b) When I did look at the Sitemap I found it useful.
(1=strongly disagree, 5=strongly agree)
 1 2 3 4 5 not applicable

(3c) What does the Sitemap convey to you?
How did it affect the way you worked on the application?
Do you have ideas for how to improve it?

(4) Now I understand how to set up input validation for text fields (e.g. to require input of a valid e-mail address).

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5 not applicable

(5a) Now I understand how to define button action rules.

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5 not applicable

(5b) Now I understand why it was necessary to define a "Clear input fields" button action on some screens (e.g. the "Offer ride" screen).

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5 not applicable

(5c) Think about building an application that has input fields (such as text fields and checkboxes) on a number of different pages.

I would expect the input fields to automatically "remember" the data that the user has entered when s/he moves between pages (as opposed to automatically clearing the fields when the user goes to another page).

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5

(6a) Now I understand how to set up and use a *Dynamic table* component.

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5 not applicable

(6b) Now I understand the following option of the *Dynamic table* component:

Show edit link only for data records "owned" by "currently logged in user".

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5 not applicable

(6c) What determines whether or not a data record is "owned" by the currently logged in user?

(6d) Now I understand the relationship between a *Dynamic table* component and the *Dynamic text* component.

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5 not applicable

(7) Now I understand how to set up user-login controls (e.g. the login-protected "Offer rides" screen).

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5 not applicable

(8a) It was convenient to be able to both edit and test components within the same screen (i.e. the *Develop* view).

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5 not applicable

(8b) Was there anything you did *NOT like* about switching between editing and testing?
Any ideas how it could be improved?

(9) Overall, Click is easy to use.

(1=strongly disagree, 5=strongly agree)

1 2 3 4 5

(10a) What are the top 3 aspects of the Click tool you do *NOT like*?

(10b) What are the top 3 aspects of the Click tool you *like*?

(10c) Other comments or ideas for improvements:

(11) Your gender:

- female
- male

(12) Participant number (please ask us!):

G.10 Visualizations of Participants' Development Timelines

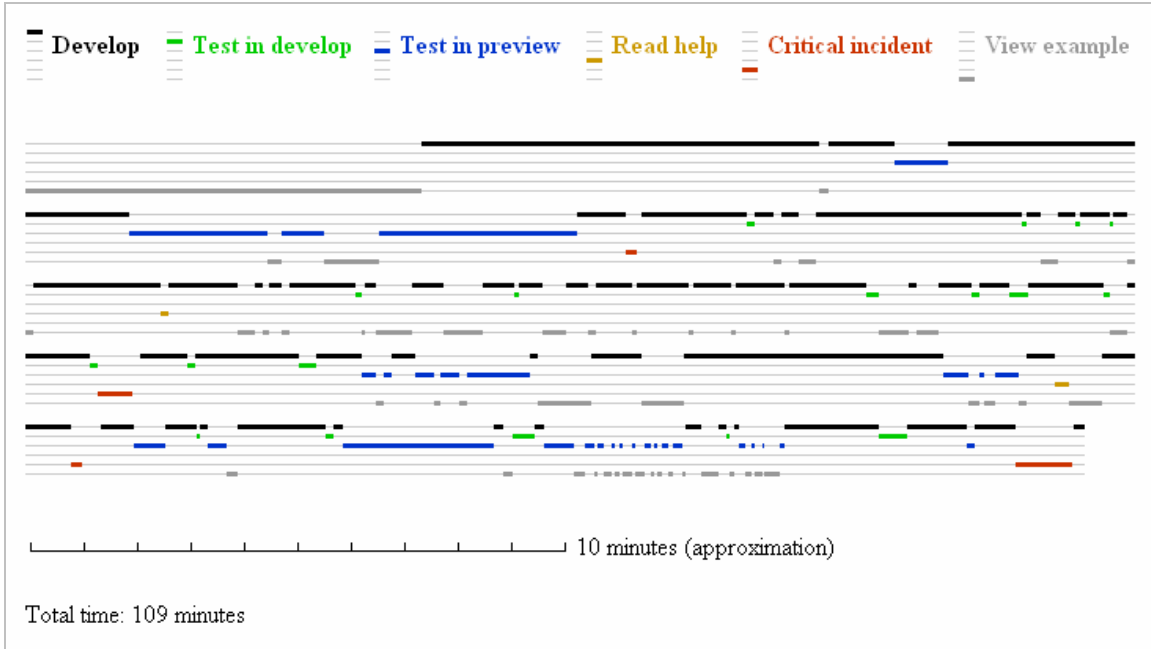


Figure 51: Click - summative evaluation: Visualization of development timeline from participant #1

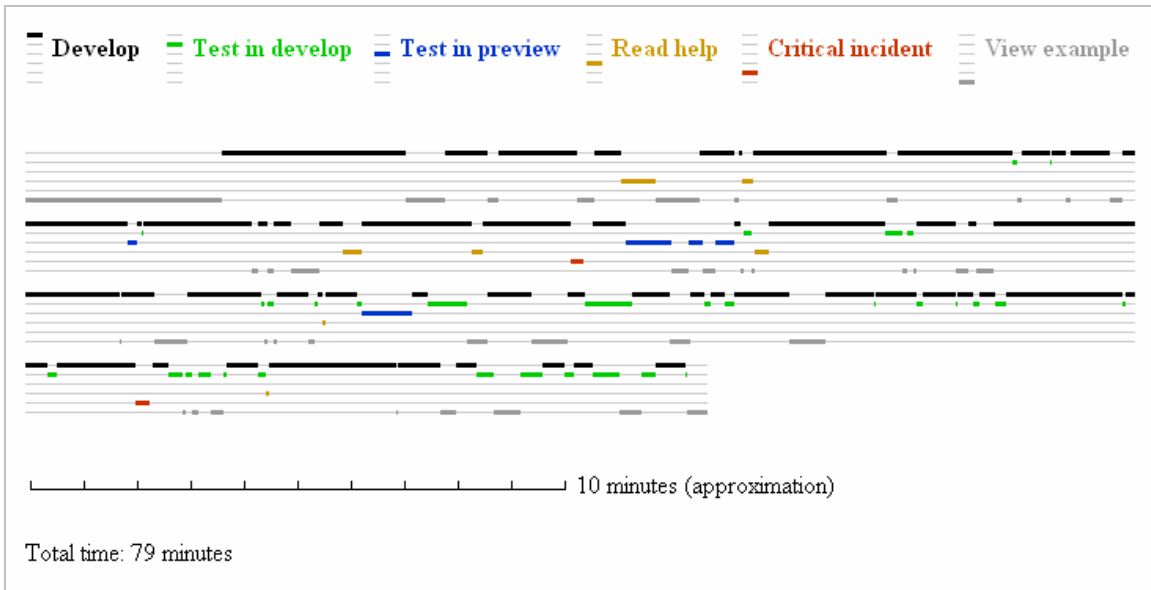


Figure 52: Click - summative evaluation: Visualization of development timeline from participant #2



Figure 53: Click - summative evaluation: Visualization of development timeline from participant #3

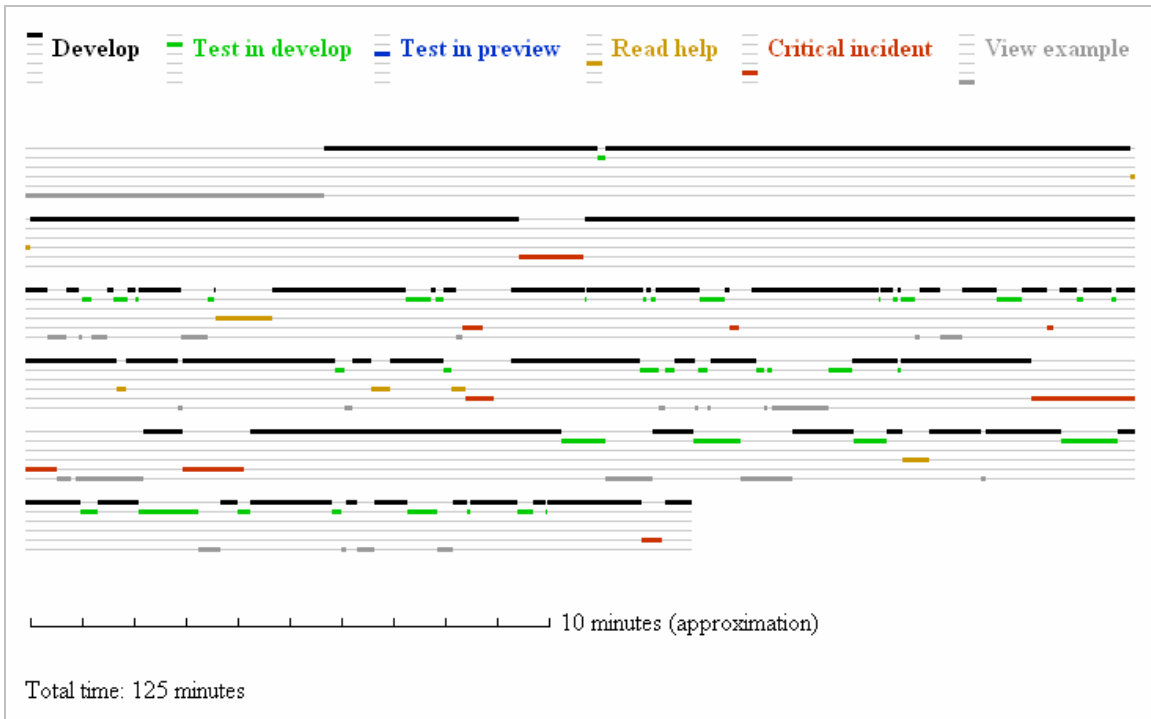


Figure 54: Click - summative evaluation: Visualization of development timeline from participant #4

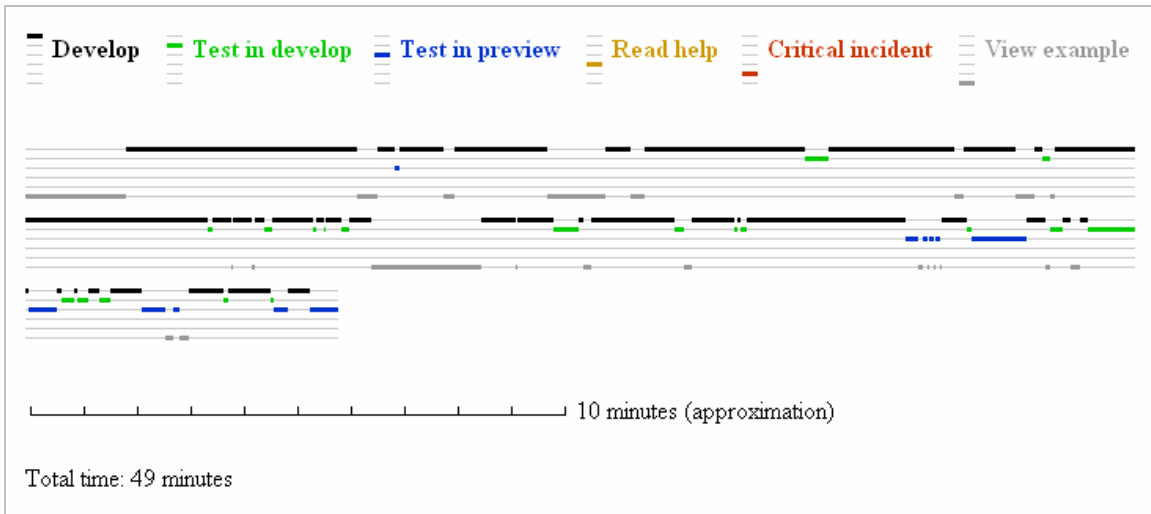


Figure 55: Click - summative evaluation: Visualization of development timeline from participant #5

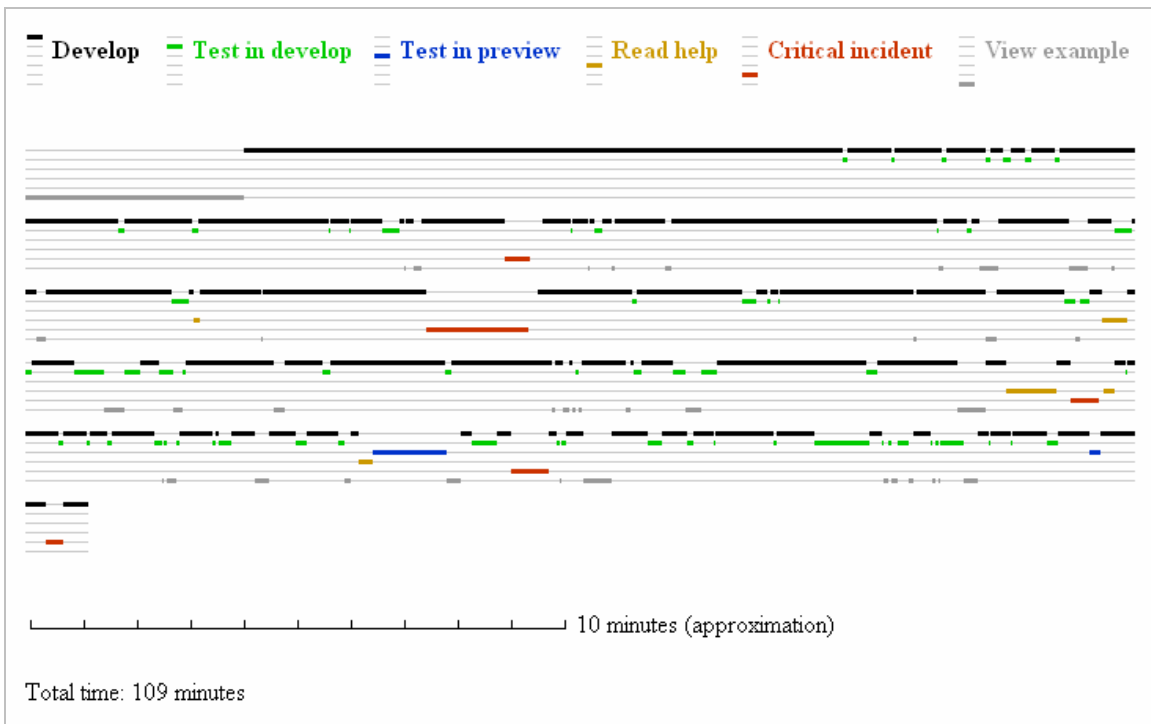


Figure 56: Click - summative evaluation: Visualization of development timeline from participant #6

Appendix H Grant Information

This dissertation is based upon work supported by the National Science Foundation under Grant No. 0353309.

Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of the National Science Foundation.

Appendix I Publications and Presentations

I.1 Peer-reviewed Full-length Conference Papers

Rode, J., M. B. Rosson (2003). Programming at Runtime: Requirements & Paradigms for Nonprogrammer Web Application Development. *Proceedings of Symposium on Visual Languages and Human-Centric Computing (VL/HCC'03)*. Auckland, New Zealand. October 28-31

Rode, J. M.B. Rosson, M. A. Pérez-Quiñones (2004). End users' Mental Models of Concepts Critical to Web Application Development. *Proceedings of Symposium on Visual Languages and Human-Centric Computing (VL/HCC'04)*. Rome, Italy. October 26-29

Rode, J., Y. Bhardwaj, M. A. Pérez-Quiñones, M. B. Rosson and J. Howarth (2005). As Easy as "Click": End-User Web Engineering. *Proceedings of International Conference on Web Engineering*, Sydney, Australia. July 27-29

Rosson, M. B., J. Ballin, J. Rode and B. Toward (2005). Designing for the Web revisited: A Survey of Casual and Experienced Web Developers. *Proceedings of International Conference on Web Engineering*, Sydney, Australia. July 27-29

Rosson, M. B., J. Ballin and J. Rode (2005). Who, What, and How? A Survey of Informal and Professional Web Developers. *Proceedings of Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*. Dallas, Texas, USA. September 20-24

I.2 Book Chapter, Abstract, Technical Reports

Rode, J., M.B. Rosson, M. A. Pérez-Quiñones (2005). “End-User Development of Web Applications.” End-User Development. H. Lieberman, V. Wulf, F. Paterno, Eds. Kluwer. (to be published in 2005)

Rode, J. (2004). Nonprogrammer Web Application Development. Doctoral Consortium. *CHI 2004*. Vienna, Austria. April 24-29

Rode, J., M. B. Rosson and M. A. Pérez-Quiñones (2002). The Challenges of Web Engineering and Requirements for Better Tool Support. Technical Report #TR-05-01. Virginia Tech Computer Science

Rode, J., J. Howarth, M. A. Pérez-Quiñones and M. B. Rosson (2004). An End-User Development Perspective on State-of-the-Art Web Development Tools. Tech Report #TR-05-03. Virginia Tech Computer Science

Vita

Jochen Rode was born close to and grew up in Germany's capital city Berlin experiencing life on both sides of the legendary wall. While enjoying the thrill of occasionally failing physics demonstrations at Berlin's Heinrich Hertz high school he had his first professional encounters with information technology by working as a programmer and computer networks technician for small businesses ranging from civil engineering to funeral services. In a slightly more cheerful environment and parallel to his studies in Business and Information Technology at the University of Applied Sciences (FHTW) he worked for "Stadt und Land" – Berlin's largest real-estate company as a system administrator, programmer, and support staff for everything that had wires and everyone who had problems with it.

In 1998 he came to the United States of America as a Fulbright scholar to work towards world peace and a Masters degree at the Computer Science department of Virginia Tech. After escaping the joys of a seven day work week by finishing the program, he experienced people's frustrations with information technology while working as a usability consultant and web application developer at Virginia Tech's IT department. Motivated by the shortcomings of computer's user interfaces, he returned to Academia as a Ph.D student in search of a cure for too much spare time and the problem of end-user software development.