Chapter 8

End User Development of Web Applications

JOCHEN RODE[1], MARY BETH ROSSON[2]
and MANUEL A. PÉREZ QUIÑONES[3]
[1] *Virginia Polytechnic Institute and State University, jochenrode@gmail.com*
[2] *Pennsylvania State University, mrosson@ist.psu.edu*
[3] *Virginia Polytechnic Institute and State University, perez@cs.vt.edu*

Abstract. This chapter 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. Our 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 we focus on supporting the development of basic data collection, storage and retrieval applications such as online registrations forms, staff databases, or report tools.

First we analyze the factors contributing to the complexity of web application development through surveys and interviews of experienced programmers; then we explore the "natural mental models" of potential end-user web developers, and finally discuss our particular design solutions for lowering entry barriers, as embodied by a proof-of-concept development tool, called Click. Furthermore, we 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.

Key words. end user development, web applications

1. Introduction

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 chapter—contrast these scenarios:

Anna uses today's web tools	Anna uses tomorrow's web tools
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	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

(*continued*)

Henry Lieberman et al. (eds.), End User Development, 161–182.

(*Continued*)

Anna uses today's web tools	Anna uses tomorrow's web tools
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 particular needs, and the information is highly technical.	even seems that the tool reads her mind. It allows her toquickly 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.

The contrast shown in these two scenarios sketches out the challenges and motivation underlying the work we report here. Our goal is to understand 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.

2. Related Work

The ubiquity of the World Wide Web and the resultant ease of publishing content to a huge audience has been an important element in the expanding skills and expectations of computer users. However, today, most web pages built by end users simply present information; creation of interactive web sites or web applications such online forms, surveys, and databases still requires considerable skill in programming and web technology. Our preliminary studies indicate that these limitations in users' web development activities are not due to lack of interest but rather to the difficulties inherent in interactive web development (Rode and Rosson 2003). Many end users can envision simple interactive applications that they might try to build if the right tools and techniques were available. If web development becomes possible for a wider audience, we may see a greater variety of useful applications, including applications not yet envisioned or as Deshpande and Hansen (2001) put it: "release the creative power of people." For organizations that cannot afford a professional programmer, end-user development (EUD) may help to streamline workflows, increase productivity and client satisfaction.

Tim Berners-Lee designed the web as a collaborative tool (Berners-Lee 1996). However, his early vision was one of document sharing, and recognition of the web's potential as a platform for dynamic collaboration has been an emergent phenomenon, with the result that much of the web's infrastructure is ill-suited for application development. Currently, development of a typical web application requires knowledge not only of traditional programming languages like Java, but also technologies and problems specific to the web, for example HTML, JavaScript, CSS, HTTP, and cross-platform, cross-browser compatibility issues. When compared to "traditional" end-user programming

of single-user desktop applications, the sum of all these technological issues, the distributed nature of the web, and the highly volatile nature of requirements add the unique flavor to end-user development for the web (EUDWeb).

Our research mission is making web application development accessible to a broader audience. We are particularly interested in "informal web developers", people who have created a variety of web content, but who have not been trained in web programming languages or tools. We believe that these individuals are good candidates for end-user web programming—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.

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, modularized, scalable, and secure. In contrast, web-related research in end-user development centers on the idea of empowering nonprogrammer end users to autonomously create websites and web applications.

2.1. RESEARCH IN WEB ENGINEERING

The state-of-the-art in web engineering is the automatic generation of web applications based on high-level descriptions of data and application logic. Research ranges from a few full-scale processes like WebML (Ceri, Fraternali, Bongio 2000) to many light-weight code generators (e.g., Turau 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 a major outstanding research problem.

Research on tailorability (e.g., MacLean et al. 1990; Stiemerling, Kahler, Wulf 1997) has focused on techniques that allow software to be customized by end users. The underlying assumption in this work is that customizable systems may address a large fraction of end users' needs. In a previous survey of webmasters (Rode and Rosson 2003), we found that approximately 40% of the web applications envisioned by respondents could in fact be satisfied by five customizable generic web applications: resource reservation, shopping cart and payment, message board, content management, and calendar.

The analysis of web developers' needs has received only little attention in the web engineering literature. A survey conducted by Vora (1998) is an exception. Vora queried web developers about the methods and tools that they use and the problems that they typically encounter. Some of the key problems that developers reported include ensuring web browser interoperability and usability, and standards compliance of WYSIWIG editors. 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 et al. (2003) 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.

2.2. RESEARCH IN END-USER WEB DEVELOPMENT

Well before the development of the World Wide Web, end-user programming (EUP) of basic data management applications was a topic for academia and industry. Apple's HyperCard is an early example of a successful EUP tool. More recently, web development research projects such as WebFormulate (Ambler and Leopold 1998), FAR (Burnett, Chekka, Pandey 2001), DENIM (Newman et al. 2003), and WebSheets (Wolber, Su, Chiang 2002) have explored specific approaches to end-user programming of web applications. WebFormulate is an early tool for building web applications that is itself web-based and thus platform independent. FAR combines ideas from spreadsheets and rule-based programming with drag-and-drop web page layout to help end users develop online services. DENIM is a tool that can assist professional and nonprofessional web developers in the early stages of design with digital sketching of interactive prototypes. The WebSheets tool, 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.

2.3. COMMERCIAL WEB DEVELOPMENT TOOLS

The research community has devoted little effort to studying approaches and features found in commercially available web application development tools. There are a few notable exceptions including the aforementioned survey of web developers (Vora 1998) and the taxonomy of tools offered in (Fraternali 1999). Brief reviews of CodeCharge Studio, CodeJay, Microsoft Visual Studio, and Webmatrix from the perspective of productivity tools for programmers can be found in (Helman and Fertalj 2003).

3. A User-Centered Approach to Web Development Tools

Our review of existing tools and research literature indicates that there is great interest in supporting general web development needs, and that there is an emerging recognition that the tools used by professionals are not appropriate for less sophisticated users. However, very little work has been directed at understanding the requirements for web development *from an end users' perspective.* Thus we have adopted an approach that combines analytic investigations of features and solutions currently in use

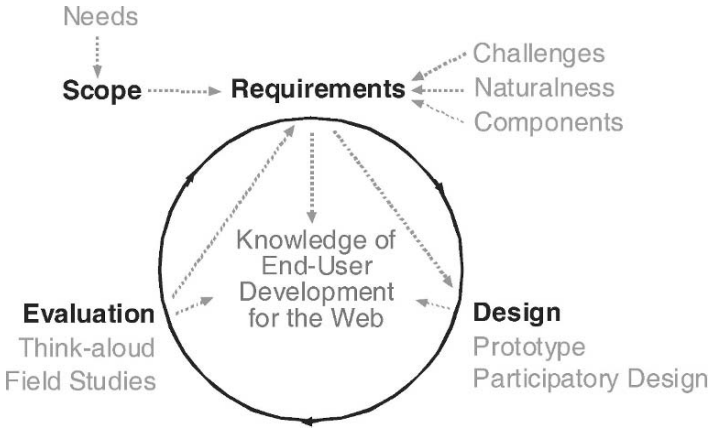

Figure 8.1. User-centered methods for building web development tools.

or under research with detailed empirical studies of end users' needs, preferences, and understanding of web development (Figure 8.1).

Although many tools for web development are already available, we do not yet know whether and how these tools meet the requirements of end users. For example, it is quite likely that professional development tools provide more functionality than is needed by nonprogrammers; we must understand what end users envision as web development projects, so that we can scope the supporting tools appropriately. At the same time, we must investigate how nonprogrammers think about the activities of web development, what concepts are more or less natural to them, what components or features they are able to comprehend.

Once we better understand the requirements for end-user web development tools, we can begin to explore techniques for meeting these requirements. This work takes place in two parallel streams, one aimed at developing and refining prototype tools, and the other at gathering detailed empirical evidence concerning the efficacy of the prototypes we build, along with comparative studies of other tools.

In the balance of the chapter we summarize our work on end-user web development. First we report a survey of sophisticated nonprogrammers (webmasters) that assessed their needs and preferences for web development; a complementary survey of programmer developers is also reported. We next describe our analysis of features present in existing web applications, as well as a usability analysis of commercial web development tools. We report two empirical studies of nonprogrammers that investigate how our target audience *naturally* thinks about typical web programming problems. We conclude with a brief description of our prototyping efforts in EUDWeb.

4. Needs Analysis for EUDWeb

Our first step toward defining a scope for our work in EUDWeb was to investigate the kinds of web applications our target population would like to build. Thus we conducted

an online survey of webmasters at our university (Virginia Tech). We reasoned that while some webmasters may have been professionally trained in web development, in a university environment most 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 we 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.

Our analysis of the survey responses (67 replies) indicated that approximately one third of end users' 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 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%. We were especially interested in the requests for applications involving basic data collection and management; such functionality seems quite reasonable to provide via an EUDWeb tool. Although this survey was a useful start, it was modest in size and restricted to one university computing population. Thus we are currently conducting a larger survey with the results expected for the first quarter of 2005 (excerpts of the results and a reference to the full results will be available at http://purl.vt.edu/people/jrode/publish/2005-05-webdevelopersurvey/).

5. Challenges Faced by Web Developers

As a second source of input to requirements development, we surveyed sophisticated developers regarding the challenges, tools, and processes within the domain of web application development. Our rationale was simple: issues that are troublesome for experienced developers may be insurmountable hurdles for novices.

We surveyed 31 experienced web developers and subsequently conducted in-depth interviews with 10 additional developers (still focusing on a university computing context). On average, the 31 respondents rated themselves just above the mid-point on a scale from "no knowledge in web application development" to "expert knowledge". Their self-reported years of experience in web application development were approximately equally distributed between "*less than a year*" and "*more than 5 years*." The 10 interview participants rated themselves in an average of 5.1 (SD = 1.3) on a scale from 1 (no knowledge) to 7 (expert knowledge). The average self-reported experience of the interview participants is somewhat higher than the mean experience of the survey participants which was only 4.3.

In both the survey and interviews we asked the developers to rate a list of potential web development problems and issues. Their responses are summarized in Figure 8.2. As the figure suggests, none of the concerns were considered to be particularly "severe";

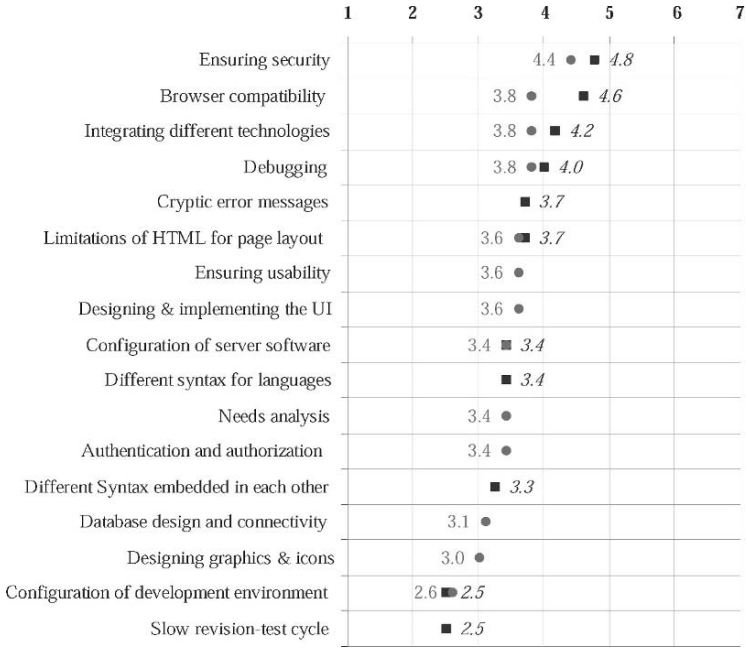

Figure 8.2. Ratings of problems in web application development (1 = not a problem at all; 7 = severe problem). The square markers are the mean ratings from the survey (value to right marker in italics; $N = 31$). The round markers are ratings from a pre-interview questionnaire (value to left of marker; $N = 10$). To facilitate comparison, the survey responses have been scaled from a 1–5 scale to a 1–7 scale.

most mean ratings were in the middle or lower half of the scale. This underscores the expected expertise of these respondents, who seem to be generally confident in their abilities to design and implement a range of web applications.

The top-rated issue in both the survey and interviews was 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 "reasonably" secure application or to assess whether and when an application is secure. Web developers are not confident about these procedures and therefore are concerned.

The inconsistencies between different browsers, versions and platforms are another source of complaints of web developers. According to our respondents, compatibility problems are also a major reason for *avoiding* the development of advanced user interaction designs using Javascript, CSS2, or Flash. Such techniques are seen as ways to improve the user experience but very risky for applications that must run on a variety of different platforms. A similar concern was the complexity of web engineering technologies: while classical desktop applications typically use 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, Java applets, Active X).

The resulting complexity leads to code that is hard to develop and maintain. Given these complexity and compatibility issues, it is not surprising that debugging was also acknowledged as an important concern.

The developers we surveyed were generally confident in their ability to solve web engineering problems, but even so acknowledged moderate concerns for the issues we probed. Several implications we drew from the survey were that EUDWeb tools should provide extensive support for security management; that cross-platform compatibility should be as transparent as possible; that the tools should automate integration of different web technologies; and that a debugger should be a basic service.

6. Cataloguing Key Components of Web Applications

In response to the needs analysis described earlier, we have chosen one particular genre of web applications as the focus of our research in EUDWeb: software that enables basic data collection and management. Once we had decided to concentrate on this specific class of applications, we needed a clear understanding of what components, concepts, and functionality would be needed to implement such software. We wanted to obtain a naturalistic sample of web components associated with this type of web application, so we gathered and analyzed the structure of database-centric websites that already existed at our university.

We used Google and its filtering capabilities (e.g., "filetype:asp site:vt.edu") to find applications in use at our institution. Using file extensions that indicate dynamic content (.asp, .aspx, .php, .php3, .cfm, .jsp, .pl, .cgi) we were able to find a large number of cases. We disregarded simple dynamic websites (scripting only used for navigation, header and footers, no database) and focused on those applications that were close to the needs expressed by the end-user 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. We reviewed the applications that were publicly accessible and constructed a list of concepts and components found within these basic web applications. The essence of this analysis is shown in Table 8.1.

The components, concepts and functions that we 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). Of course, commercial Web development tools already offer much of the high-level functionality listed in Table 8.1, but most of these tools are not aimed at nonprogrammers. We expect the list of elements to change and grow along with our knowledge about web applications and the progress of technology. We see this list as simply a start towards a functional requirements list for EUDWeb tools.

7. Analysis of State-of-the-Art Tools

Some of the most active work on EUDWeb is in commercial web development tools. Thus as yet another source of requirements, and to better ground our research in related

Table 8.1. Components, concepts and functionality of basic web applications.

Concept or function	Description
Session management	Maintaining state information when going from page to page, "fixing" HTTP's connectionless nature
Input validation	Validating user inputs for increased usability but also for increased security to preventing hacker attacks
Conditional output	e.g. only show "Hi John!" when John is logged in
Authentication and authorization	Restricting who can use the web application and which features can be used by whom
Database schema	The structure of the tables holding the data
Database lookup	e.g. resolving a user-ID to a full name
Overview-detail relationships	e.g. showing a list of employees on one web page and when clicking on the name, the details on another
Normalization & use of foreign keys	Addressing data redundancy issues
Uniqueness of data records	Being able to identify each record even though the data be the same
Calculating database statistics	e.g. showing the number of registrations in online registration database
Search	e.g. finding a person's e-mail in a staff database

work, we reviewed nine commercial web development tools. 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 (Rode et al. 2005).

7.1. OVERVIEW OF TOOLS ANALYSIS 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 (Table 8.1). To make these features more concrete and to convey our assumptions about a likely end-users' goals and activities, we had 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.

7.2. USABILITY FINDINGS

What does the ideal web application development tool look like? We 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-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 (2000) 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."

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 FP 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, none of the tools that we reviewed would work without 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) is looking for needs to provide 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 stepwise exposing the inner workings of the wizards and forms. Ideally, following the concept of the "gentle slope" (MacLean 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".

8. End Users' Understanding of Web Development

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. 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 "mental model" of the end-user developer. 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" 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 our target audience and how detailed are they? We report two studies in an attempt to this question. The studies adapt the methods of Pane, Ratanamahatana, and Myers (2001), who designed a "natural" programming language for children by first studying how children and adults use natural language to solve programming problems. In our variation of their method, we investigate how nonprogrammers naturally think about the behavior of web applications. The findings from this work have guided the design of our prototype EUDWeb tool, as will be discussed later.

8.1. 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 employed when solving web programming problems.

8.1.1. *Participants and Study Procedures*

We recruited participants for a two-part paper and pencil study. Participants were asked to label screen elements and to specify the application behavior. We created a simple web application (member registration and management) for the study. Ten participants were sampled randomly from organizational 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).

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 labelling instructions included a sample labelled image (a room with objects), including nested items. This first phase of the study was intended to inform us about the language our audience uses to reference visible screen elements (left side of Figure 8.3).

Next, they were allowed to explore the application until they were comfortable with how it worked. After the 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. The seven tasks were illustrated by concise instructions that were designed to guide the user without biasing their response—for example,

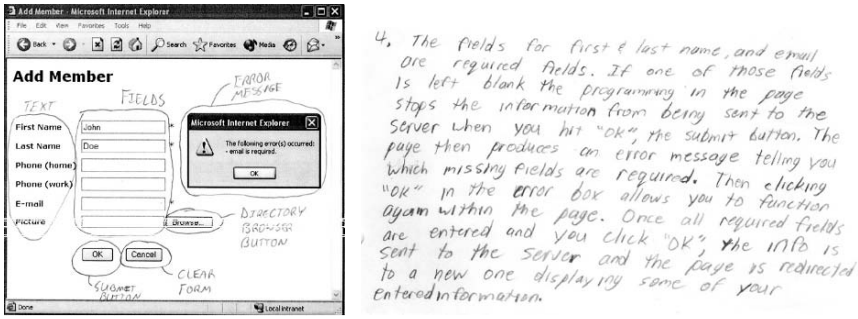

Figure 8.3. Annotated screenshot and a participant's description of the behavior of the "Add Member" dialog (MMODELS-1).

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 application was available for reference. Participants wrote responses using screenshots and blank paper (right side of Figure 8.3). We 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.

8.1.2. *Study Findings*

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); however, they made little attempt to describe how "hidden" operations are accomplished. Given these users' background in web authoring, we were not surprised to find that they used terms common in WYSIWIG web editors to label screen elements. 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 constructs such as variables and loops in the behavior specifications. Where looping constructs are required (e.g., authenticating a user), the participants specified one iteration, seeming to expect that it would apply (i.e., be repeated) as necessary.

We were particularly interested in how these users described web-specific data processing—e.g., client-server interaction, HTML generation, the web's stateless nature, and so on. Only three users 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 or manages data. We thought that the search and sort tasks might evoke informal descriptions of algorithms, but most participants focused on a result (e.g., what the user sees next in a table) rather than on how a data listing was 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.

8.2. MENTAL MODELS OF SPECIFIC WEB DEVELOPMENT TASKS

One problem with our study of concepts and language for web programming (MMODELS-1) was the generality of the problem-solving it required: we 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 rather predictable) tendencies in end users' mental models. We wanted to find out how end users would think about the specific components and features we had catalogued in our analysis of existing database-centric web applications (e.g., input validation, database lookups, overview-detail relationships). We carried out a second study [MMODELS-2] to explore these issues (Rode, Rosson, Pérez-Quiñones 2004). Our goal was to determine how end users *naturally think* about *typical concerns* in web application development.

We are concerned with naturalness in this sense: by studying the "stories" that nonprogrammer webmasters can generate about how a specific programming feature works, we hope to develop approaches for supporting this feature that will be intuitive to this user population.

We wanted to begin our investigation with programming concerns that are commonly addressed by web developers when creating a web application. Thus we selected a set of concerns that appeared frequently in an earlier analysis of 61 existing web applications (as discussed before). As experienced web developers, we also relied on our personal experiences to judge what programming concerns are most important in web development. Because many, if not most web applications work with databases, many of the programming issues are database-related.

8.2.1. *Participants and Study Procedures*

We recruited 13 participants (7 female, 4 male, and 2 who were later eliminated because they did not match our target audience) 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) but very little or no programming background.

The screening survey did not question participants for their experience with databases. However, during the interview 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 (our 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 our interviews suggest that the level of database understanding is novice to intermediate rather than expert.

The goal of this study was to better understand how webmasters who do not know how to program are able to imagine how a range of computational processes might take place "inside" an interactive web application. Probing naive expectations of this sort is a challenge, because the facilitator must provide enough information 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, we presented and asked questions about nine scenarios (for full list of scenarios see Rode, Rosson, Pérez-Quiñones 2003), each describing one or more programming concerns related to a fictional web application—an online video library system.

Each scenario consisted of a mock-up of a screen shot, a short paragraph explaining what the mocked-up screen depicts, and a series of questions. As an example, Figure 8.4 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. Most of the questions begin with the words: "What do you think the web site must do to . . . "; we hoped that

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 8.4. Scenario 1 of 9 as shown to each participant.

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; the facilitator 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 interviews took place in a one-on-one setting in a private atmosphere. Verbal responses were voice recorded for later analysis.

Not all users answered all questions. Sometimes a participant responded simply that "I have no idea" rather than attempting an explanation. In such cases we encouraged participants to give a "best guess", but occasionally we were forced to continue without an explanation. In general we were sensitive to participants' comfort level, and if an interviewee conveyed or said that they were feeling "stupid" we quickly moved on to another question.

We analysed the study in the following manner. First, we transcribed the recorded interview for each participant (focusing on analysis questions, and excluding unrelated remarks). If participants had made sketches we used those to understand and annotate their remarks. Second, in a separate document we listed the eleven 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, we 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, we summarized the results for each development concern by referring back to this document, and when necessary the transcribed interviews or even the original recordings.

8.2.2. *Study Findings*

In the balance of this section, we discuss what we have learned from our participants regarding their understanding of specific aspects of web application development and how these findings have influenced our 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.

Input validation. The typical approach of defining an input mask using patterns or placeholders (as used by many existing tool, e.g. Microsoft Access) seems to be an appropriate abstraction for end users. Certainly, this result is unsurprising in light of the fact that ten participants had previous database experience and were familiar with this notion.

Conditional output. Although, the concept of "if-then" branching was frequently mentioned informally, the exact implementation (in particular when and where if-then

rules would be applied) did not appear trivial to most participants. This suggests that while an EUD 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 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 (of course, this may not be the case 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 often was 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. We 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 expose the details of the implementation.

8.3. IMPLICATIONS FOR EUDWeb RESEARCH

From a methodological point of view we 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. We are considering a refinement of the approach that has a more "graduated" set of scenarios and questions. For example, we 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. We are also considering the use of examples as a prop in the discussions: for example, when a nonprogrammer states that they have no idea how a process takes place, we might present them with two plausible alternatives (where one is "correct") and ask them to evaluate the differences.

We 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.

Based on our results so far, we would characterize a "prototypical" end-user web developer in the following way. He or she:

- 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 (if under-specification is pointed out to them),
- Naturally uses a mix of declarative language (e.g. constraints) and procedural language (e.g. if-then rules) to describe behavior, while being unclear about where and when these rules 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 checking),
- Understands the utility of advanced concepts (e.g. unique key fields, normalization) but is unlikely to implement them correctly without guidance,
- Thinks in terms of sets rather than in terms of iteration (e.g. show all records that contain "abc"),
- Imagines a spreadsheet table when reflecting on data storage and retrieval.

The type of mental models study we 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 our current work.

9. Prototyping and Evaluating EUDWeb Tools

Prototyping is an integral part of our research on EUDWeb. First, it has helped to assess the feasibility of design ideas; second, prototypes serve as research instruments to support observational studies (e.g., end-user debugging behavior); and third, we are hoping to soon discover new requirements for EUDWeb through participatory design with the users of our prototype system in an ecologically valid manner. We have explored many different paths, including extensions to a popular web development tool (Macromedia Dreamweaver) to offer web application features more suitable to end users and implementing an online tool using Macromedia Flash (Rode and Rosson 2003). Although tools like Dreamweaver and FrontPage have substantial extension APIs, we found the inflexibility in controlling the users' workflow as the main hindrance to adopting this approach. Using Flash itself as a platform solves many layout and WYSIWYG issues but presents the problem that most users still want to produce HTML-based web sites. From many informal user studies we have learned that the web development tool that users envision is typically "Word for Web Apps", expressing a preference for a desktop-based tool that embraces the WIMP, drag-and-drop, and copy-and-paste metaphors, offers wizards, examples and template solutions, but yet lets the developer see and modify the code "behind the scenes." Our current approach uses an HTML/JavaScript/PHP-based online tool that is integrated with a database management system (MySQL). Figure 8.5 shows a screenshot from Click (Component-based Lightweight Internet-application Construction Kit), our most recent prototype (Rode et al. 2005). In the depicted scenario, the developer defines the behavior of a button component in a declarative way, that is, upon pressing the "Register" button the application would redirect the user to the web page "confirmationpage" if and only if the e-mail field is not left blank. Click distinguishes itself from other state-of-the-art EUDWeb tools in that it fully integrates the process of modelling the look and feel, component behavior, database connections, publishing and hosting, while working on a high level of abstraction appropriate for nonprogrammers.

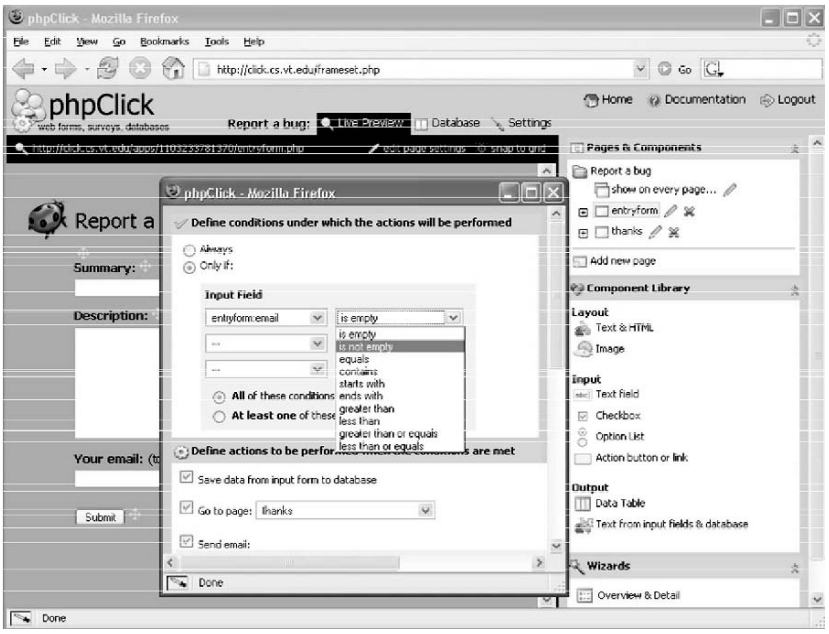

Figure 8.5. Screenshot of Click showing definition of a button's behavior.

Click implements what we call "design-at-runtime", applying Tanimoto's concept of liveness (Tanimoto 1990). This concept builds from the ideas of direct manipulation (Shneiderman 1983) and the "debugging into existence" behavior (Rosson and Carroll 1996) studied in professional programmers. At 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 programming and runtime modes. That is, the application is always usable to the fullest extent that it has been programmed, and when its boundaries are tested, the environment provides useful feedback suggesting next steps for the developer to take.

An important design goal for Click is to support evolutionary prototyping, to allow the developer to easily change virtually every aspect of the web application at any point in time For example, in the figure the user is updating the behavior of a submit button while in the midst of testing her application. This can be contrasted to most state-of-the-art tools that require significant "premature commitment", as Green (1989) might call it. For example, in many tools the database schema has to be fully defined before the application is implemented and is difficult to change after the fact.

Finally, Click provides several layers of programming support. While novices can customize existing applications or work with a predefined set of components and actions, more advanced developers can manually edit the underlying code which is based on HTML, PHP, and the event-driven PRADO framework (Xue 2005). Click strives to expose a "Gentle slope of complexity" as advocated by MacLean et al. (1990).

9.1. EVALUATION AND LESSONS LEARNED

Click has been released as an open source tool (http://phpclick.sourceforge.net) and may soon be formally released to the Virginia Tech computing community in an effort to elicit important requirements for EUDWeb through large-scale participatory design and as an instrument for field studies (see Figure 8.1). In the course of developing Click, we have also begun to carry out a series of usability evaluations, gathering feedback from representatives of our target audience. Although many usability issues are left to be resolved, Click already addresses many problems. For example, Click completely hides session management (all inputs entered on one web page are available at any later point in time), integrates the database management within the tool (as the developer creates a new input field on screen a matching database field is created), and allows the developer to design the layout in a true WYSIWYG fashion without having to revert to "tricks" like using HTML tables for alignment.

10. Summary and Conclusions

We have described the initial phases of a user-centered approach to understanding and supporting EUDWeb. From investigating end users' needs we 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. These types of applications are also a *feasible* target considering that most web applications are simple—at least conceptually speaking. While currently the implementation of any non-trivial, secure, and cross-platform compatible web application requires expert knowledge, it does not have to be this way. Most of what makes web development difficult is not inherent complexity but rather an accumulation of many technical challenges. Concerning the main challenges in web application development, experienced web developers mentioned the issues of ensuring security, cross-platform compatibility, the problems related to integrating different web technologies such as Java, HTML, PHP, Javascript, CSS, SQL, and, the difficulties of debugging distributed applications.

Many web applications are quite similar on a conceptual level. By analysing existing applications we have compiled a list of frequently used components, functions, and concepts such as session management, search, and overview-detail relationships (Table 8.1). The web development process will become much easier and more accessible to nonprogrammers when tools integrate these concepts as building blocks on a high level of abstraction rather than requiring low level coding.

Much progress has been made by commercial web development tools. Most of the end-user tools that we reviewed do not lack functionality but rather ease-of-use. We also found that while many tools offer wizards and other features designed to facilitate specific aspects of end-user development, few (if any) take a holistic approach to web application development and integrate layout, styling, behavioral description, data modelling, publishing, and maintenance tasks. The "ideal" tool for end-user web

developers would provide ease-of-use with the appropriate abstractions, absence of jargon, a library of examples and templates, wizards for complicated tasks and take a holistic approach by integrating all aspects of web development. Finally, such a tool would also support developers' growing needs and knowledge, offer power and flexibility by allowing the integration of user-defined and automatically-created code.

Understanding how end users naturally think may help us design tools that better match their expectations. In two studies we found, that end users frequently only have vague ideas of how web applications works behind the scenes, and that end users expect many aspects such as session management or search to work "out-of-the-box." However, the nonprogrammers we have observed, generally did not have problems to think on an abstract level about the concepts behind web application development and for example easily understood concepts like "if-then" branching although being unable to say where and how it would be implemented.

We have begun constructing and evaluating an EUDWeb tool prototype called Click that supports end user web application development from start (requirements elicitation through application prototyping) to finish (deployment and maintenance). We are confident that a tool like Click will soon make the "tomorrow" of the introduction scenario and end-user development for the web a reality.

Acknowledgements

We thank Julie Ballin and Brooke Toward for their roles in development and administration of a large-scale survey of web developers; Yogita Bhardwaj, and Jonathan Howarth for helping develop Click, our EUD prototype, and conducting a review of existing web tools; Betsy and Erv Blythe for supporting the idea of EUDWeb within Virginia Tech's IT department, and last but most definitely not least, B. Collier Jones, Jan Gibb, Kaye Kriz and Dr. Andrea Contreras for their valuable feedback and support throughout our research.

References

Ambler, A. and Leopold, J. (1998). Public Programming in a Web World. *IEEE Symposium on Visual Languages*, Nova Scotia, Canada: 100–107.

Berners-Lee, T. (1996). "WWW: past, present, and future." *IEEE Computer* **29**(10): 69–77.

Burnett, M., Chekka, S.K. and Pandey, R. (2001). FAR: An End user Language to Support Cottage E-Services. *HCC—2001 IEEE Symposia on Human-Centric Computing Languages and Environments*, Stresa, Italy: 195–202.

Ceri, S., Fraternali, P. and Bongio, A. (2000). "Web Modeling Language (WebML): A Modeling Language for Designing Web Sites." *Computer Networks* 33(1–6): 137–157.

Deshpande, Y. and Hansen, S. (2001). "Web Engineering: Creating a Discipline among Disciplines." *IEEE MultiMedia* **8**(2): 82–87.

Fraternali, P. (1999). "Tools and Approaches for Developing Data-Intensive Web Applications: A Survey." *ACM Computing Surveys* **31**(3): 227–263.

Fraternali, P. and Paolini, P. (2000). "Model-Driven Development of WebApplications: The Autoweb System." *ACM Transactions on Information Systems* **28**(4): 323–382.

Green, T.R.G. (1989). Cognitive dimensions of notations. In A. Sutcliffe & I. Macauley (eds.), *People and Computers IV*. Cambridge: Cambridge University Press.

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

MacLean, A., Carter, K., Lövstrand, L. and Moran, T. (1990). User-Tailorable Systems: Pressing Issues with Buttons. *ACM CHI 1990*: 175–182.

Newman, M., Lin, J., Hong, J.I. and Landay, J.A. (2003). "DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice." *Human-Computer Interaction* **18**: 259–324.

Pane, J.F., Ratanamahatana, C.A. and Myers, B.A. (2001). "Studying the language and structure in non-programmers' solutions to programming problems." *International Journal of Human-Computer Studies* **54**(2): 237–264.

Rode, J. and Rosson, M.B. (2003). Programming at Runtime: Requirements & Paradigms for Non-programmer Web Application Development. *IEEE HCC 2003*. Auckland, New Zealand. Oct. 28–31.

Rode, J., Rosson, M.B. and Pérez-Quiñones, M.A. (2003). Participant Instructions. http://purl.vt.edu/people/jrode/publish/2003-09-interviews/instructions.pdf

Rode, J. Rosson, M.B. and Pérez-Quiñones, M.A. (2004). End users' Mental Models of Concepts Critical to Web Application Development. *IEEE HCC 2004*. Rome, Italy. Oct. 26–29.

Rode, J., Bhardwaj, Y., Rosson, M.B., Pérez Quiñones, M.A. and Howarth, J. (2005). Click: Component-based Lightweight Internet-application Construction Kit. http://phpclick.sourceforge.net

Rode. J., Howarth, J., Pérez Quiñones, M.A. and Rosson, M.B. (2005). An End-user Development Perspective on State-of-the-Art Web Development Tools. Virginia Tech Computer Science Tech Report #TR-05-03.

Rode, J., Bhardwaj, Y., Pérez-Quiñones, M.A., Rosson, M.B. and Howarth, J. (2005). As Easy as "Click": End-User Web Engineering. *International Conference on Web Engineering*. Sydney, Australia. July 27–29.

Rosson, M.B. and Carroll, J.M. (1996). "The reuse of uses in Smalltalk programming." *ACM TOCHI* **3**(3): 219–253.

Rosson, M. B., Ballin, J., Rode, J. and Toward, B. (2005). Designing for the Web revisited: A Survey of Casual and Experienced Web Developers. *International Conference on Web Engineering*. Sydney, Australia. July 27–29.

Shneiderman, B. (1983). "Direct Manipulation: A Step Beyond Programming Languages." *IEEE Computer* **16**: 57–60.

Stiemerling, O., Kahler, H. and Wulf, V. (1997). How to Make Software Softer—Designing Tailorable Applications. *Symposium on Designing Interactive Systems 1997*, 365–376.

Tanimoto, S. (1990). "VIVA: A Visual Language for Image Processing." *Journal of Visual Languages and Computing* **1**(2): 127–139.

Turau, V. (2002). A Framework for Automatic Generation of Web-based Data Entry Applications Based on XML. *17th Symposium on Applied Computing*, Madrid, Spain, ACM: 1121–1126.

Vora, P.R. (1998). Designing for the Web: A Survey. *ACM interactions* (May/June): 13–30.

Wolber, D., Su, Y. and Chiang, Y.T. (2002). Designing Dynamic Web Pages and Persistence in the WYSIWYG Interface. *IUI 2002*. Jan 13–16. San Francisco, CA, USA.

Xue, Q. (2005). PRADO: Component-based and event-driven Web programming framework for PHP 5. http://www.xisc.com/