As Easy as "Click": End-User Web Engineering

Jochen Rode[1], Yogita Bhardwaj[1], Manuel A. Pérez-Quiñones[1],
Mary Beth Rosson[2], and Jonathan Howarth[1]

[1] Virginia Polytechnic Institute and State University, Center for Human-Computer Interaction
3160 Torgersen Hall, Blacksburg, VA 24061
{jrode,yogitab,jhowarth}@vt.edu, perez@cs.vt.edu
[2] Pennsylvania State University, Information Sciences & Technology,
330 IST Building, University Park, PA 16802
mrosson@ist.psu.edu

Abstract. We are investigating the feasibility of end-user web engineering. The main target audience for this research is webmasters without programming experience – a group likely to be interested in building web applications. Our target domain is web-based data collection and management applications. As an instrument for studying the mental models of our audience and collecting requirements for an end-user web programming tool, we are developing Click, a proof-of-concept prototype. We discuss end-user related aspects of web engineering in general and describe the design rationale for Click. In particular, we elaborate on the need for supporting evolutionary prototyping and opportunistic and ad hoc development goals. We also discuss strategies for making end-user web engineering scalable and for encouraging end-user developers to continually increase their level of sophistication.

1 Introduction

Years after the introduction of CGI-scripting, the creation of a web application is still difficult, requiring a broad range of skills. Professional programmers develop the skills needed to create interactive web applications, but most nonprogrammers engaged in web development are limited to the creation of static websites. We believe that with the right tools and techniques even nonprogrammers may be able to develop web applications. By making web development possible for a wider audience, we may see a greater variety of useful applications being developed, including functionality not yet envisioned. For organizations unable or unwilling to hire professional programmers, end-user development may help streamline work flows and increase productivity and client satisfaction. Indeed, the WWW itself is an excellent example of what happens when technology becomes accessible to "the rest of us". In the words of Deshpande and Hansen [6], it is time for the web engineering community to "devise methods and processes to assist end users" which would help to increase the reliability of applications and "release the creative power of people."

Apart from empowering end users to pursue new goals, the web engineering community should also be considering how best to help novice developers create websites that are more secure, cross-platform-compatible, and universally accessible. User-friendly but "dangerously powerful" web programming languages like PHP [15] are becoming popular even among people who do not have the necessary training and experience to develop web applications of high quality. Harrison [10] calls this the

D. Lowe and M. Gaedke (Eds.): ICWE 2005, LNCS 3579, pp. 478–488, 2005.

"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. We choose the latter.

A good starting point is to focus on the needs and preferences of *sophisticated end users*, people who are experienced with web design in general but not (yet) with the programming required for interactive applications (e.g., input validation, database access, authentication). Our preliminary studies of university webmasters [18] indicates that a substantial fraction of these sophisticated end-users' web application needs are quite simple and similar. For instance, in one survey of Virginia Tech webmasters (n=67) we found that about one third of the applications described by these users are *basic data collection, storage, and retrieval applications* (such as service request forms, searchable publication databases, staff databases, and surveys). Another 40% of the requests could be satisfied through customization of five generic web applications (resource scheduling, shopping cart and payment, message board, content management, and calendar). Research on tailorability demonstrates that customizability is an achievable design goal (e.g., [12]). Diverse requests for more advanced applications comprised the remaining 25%. Based on these analyses, we have focused our efforts on tools for end-user development of web applications (EUDWeb) that revolve around basic data collection and management.

In the balance of this paper we first consider related work in web engineering and end-user development and discuss strategies for making web engineering easier for nonprogrammers. Then, we present our approach to EUDWeb, focusing on features and design rationale for Click (Component-based Lightweight Internet-application Construction Kit), our proof-of-concept EUDWeb tool [20].

2 Related Work

Two complementary domains of research and practice – *web engineering* and *end-user development* – have focused on methods and tools for the creation of web applications. Research in web engineering has concentrated on making web professionals more productive and the websites that they produce more usable, reusable, modularized, scalable, and secure. In contrast, research on end-user development for the web has attempted to empower nonprogrammers to autonomously create websites and web applications. Within the web engineering community, one research focus has been on *model-based approaches* to the design of hypermedia systems (e.g., [23], [5]). While these top-down design approaches address many problems related to productivity, consistency, security, and platform-independence, they normally assume a high-level of abstraction unsuited for nonprofessional web developers. The approach we advocate can be seen as an alternative to model-based, top-down development. Another focus of research and practice is *tools* that assist web developers in becoming more productive. Many powerful CASE/RAD tools have been developed for experienced developers like Web Models' WebRatio [25], IBM's Rational Web Developer for WebSphere Software [11], or Microsoft's Visual Web Developer 2005 [13]. 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. In 2004 we reviewed selected state-of-the-art web development tools designed for end users [21] such as Microsoft FrontPage or Macromedia Dreamweaver. 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 end-user 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. Fraternali's and Paolini's observation about available web tools [8] seems equally true today as it did five years ago: "…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 possibilities of web application development by end users have only recently become a topic of research. WebFormulate [1] is a tool for building web applications that is itself web-based. FAR [4] combines ideas from spreadsheets and rule-based programming with drag-and-drop web page layout to help end users develop online services. The WebSheets tool [26], although currently limited in power, uses a mix of programming-by-example, query-by-example, and spreadsheet concepts to help nonprogrammers develop fully functional web applications. Although the prior EUDWeb work has investigated many particular aspects and opportunities of web development, we are not aware of any research that has approached the problem in a holistic manner, starting with the needs of developers, analyzing the barriers and then prototyping tools. This is what we strive to provide.

3 Web Engineering for End Users

Web engineering is complex, but many aspects of its current complexity are not inherent to the process. For data management applications, much of what makes web development difficult is what Brooks [3] has termed "accidental complexity"–barriers introduced by the supporting technology rather than the problem at hand. Examples of accidental complexity and corresponding hurdles for web developers include [19]:

- Ensuring security;
- Handling cross-platform compatibility;
- Integrating technologies (e.g., HTML, CSS, JavaScript, Java, SQL); and
- Debugging web applications.

Our empirical studies of nonprogrammers' intuitions about web programming has yielded an even longer list of concerns, for example the stateless nature of HTTP and the necessity for explicit session management, parameter passing between pages or modules of an application, input validation, and establishing and managing database connections [18]. If web development is to become "end-user friendly", *accidental complexity must be eliminated or hidden as much as possible.*.

A common approach to creating processes and tools that better match end users' goals and expectations is to make the tools specific to the users' problem domain [14]. We have adopted this perspective: Rather than looking for an EUDWeb "silver bullet" that is as general and powerful as possible, our approach is to *identify classes of*

needs and build tools that target these domains specifically, while at the same time planning for extensibility as users' requirements grow and evolve.

Empirical studies of professional programmers have shown that software developers do not always follow a systematic design approach. Sometimes programmers develop top-down; sometimes they "dive into" details and build parts of an application bottom-up [7]. Furthermore, software developers rarely construct an application in one step but rather perfect it incrementally through evolutionary prototyping [22]. Although there are few if any empirical studies of novice web developers, our informal observations and interviews lead us to believe that this phenomenon extends to these more casual developers just as much (if not more so). Therefore, EUDWeb tools should *support or even encourage opportunistic behavior*.

A critical tradeoff for every end-user development tool is the relationship of usability and expressiveness. Ideally a tool's complexity will be 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 our EUDWeb design goals is to make the effort required more proportional to the complexity of the problem at hand. We advocate a *"gentle slope of complexity"* [12], arguing for tools that adapt and grow with users' needs in a *layered* fashion. For the Agentsheets simulation tool, Repenning and Ioannidou [16] 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. We recommend a similar approach for EUDWeb.

We turn now to a presentation and discussion of the Click prototype, which was built as a demonstration of these EUDWeb requirements and recommendations.

4 Click: A Prototype End-User Web Engineering Tool

We are developing Click [20] as an EUDWeb prototype that is specifically targeted at end users who want to develop web-based data collection, storage and retrieval applications. Before we discuss Click's main contributions in detail, we will briefly illustrate how an end-user developer might use it to create a web application.

4.1 Developing Web Applications with Click

To construct a web application, an end user 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 1 shows Click being used to define a "Register" button that (ultimately) will save user-entered data into a database and display another web page. 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).

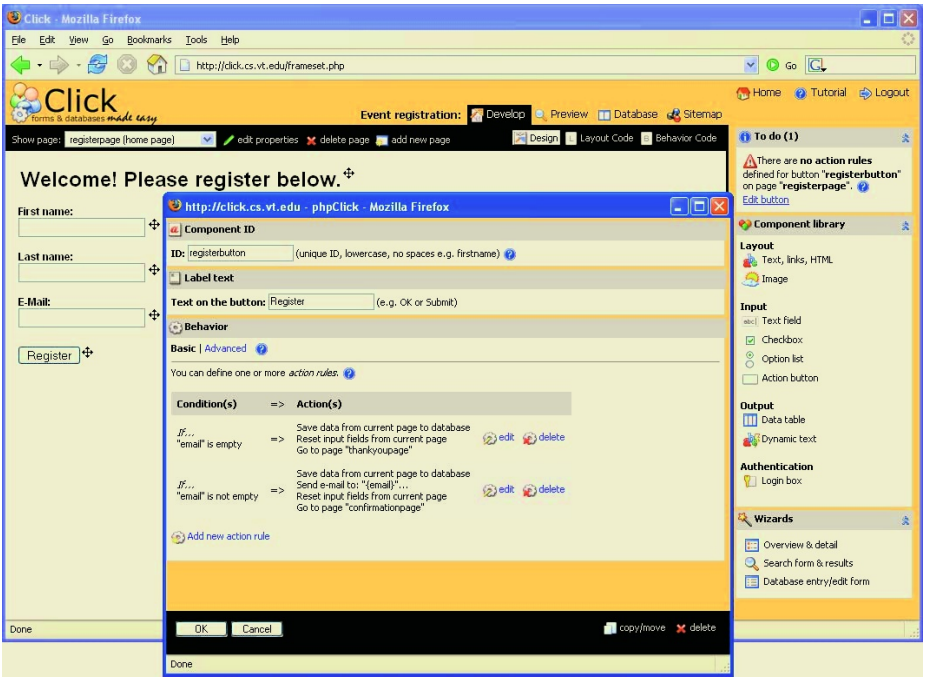

Fig. 1. Defining a "Register" button and associated action using the form-based UI of Click

4.2 Hiding Unnecessary Complexity

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 (Figure 2).

Separately, 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 (Figure 3). These functions are designed to be understandable by novice programmers who want to go beyond the dialog/form-based facilities.

Click's pre-defined components have been selected based on several analyses of both existing web applications and end users' mental models of interactive web programming [18]. The components provide the functionality needed to implement a typical data storage and retrieval application (e.g. a data table, dynamic text output). Click has been designed to make session management, authentication, database management, and so on, relatively automatic and invisible, so that only minimal learning is required. For example, by default, all data entered by a user on a web page persist even after the page has been submitted, so that the web application can continue to refer to and use this data at any point in time (we found that end users assume that once information has been entered, the system should "know" about it).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1DTD/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="16" Y="17" Z="61">
  <prop:Text><h1>Welcome! Please register below.</h1></prop:Text>
</com:HtmlText>
<com:InputText ID="firstname" X="14" Y="61" Z="66" Columns="20" Rows="1"
TextMode="SingleLine" DbFieldName="data:firstname" InputRequired="false"    Val-
ueType="Characters" MinValue="1" MaxValue="30">
  <prop:Label><b>First name:</b><br /></prop:Label>
  <prop:ErrorMessage>Please enter between 1-30 characters.</prop:ErrorMessage>
</com:InputText>
<com:InputText ID="lastname" X="14" Y="113" Z="67" 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 1-50 characters.</prop:ErrorMessage>
</com:InputText>
<com:InputText ID="email" X="14" Y="165" Z="68" 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="13" Y="223" Z="70" On-
Click="registerbutton_runActions" />
</com:Form>
</body>
</html>
```

Fig. 2. The "Layout code" view for the screen seen in Fig. 1

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

Fig. 3. The "Behavior code" view for the screen seen in Fig. 1

A recent review of state-of-the-art web development tools [21] revealed that one problem is that such tools rarely provide "holistic guidance" for developers, instead expecting them to know the exact steps required to implement a web application. Click does not attempt to predict and interrupt a developer's workflow in the way an "intelligent" software agent might do since the risk and costs of false guesses would

likely be high [17]. However, Click maintains a non-intrusive "To-do" list (see upper right of Figure 1) that keeps track of the developer's progress and gives recommendations about possible or required future tasks. 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.

Furthermore, Click is able to automatically create new database fields and web pages if the developer refers to them in a rule (for example in a `saveToDatabase` or `goToPage` action). This eliminates the interruption and distraction caused when a programmer must pause his or her problem-solving process to set up supporting structures.

The layout feature of Click allows developers to place components using drag-and-drop and absolute positioning. This allows for quick prototyping and shields novice web developers from the difficulties of HTML table-based or CSS-based layout. More advanced users can edit the layout code directly and add HTML and CSS code in order to gain more control over the presentation.

Instead of exposing the developer to the page-submit-cycle metaphor typically found in web development tools, Click implements an event-based programming model similar to that found in ASP.NET or Java Server Faces (or programming tools for desktop applications such as Visual Basic). In this model, buttons and links have event handlers whose actions can be defined either by completing a Click form or by using the high-level PHP functions mentioned earlier (`sendEmail, goToPage`, etc.) The developer is shielded from the details of passing parameters to a page via HTTP's GET or POST methods, receiving and validating these inputs, and so on.

When the developer selects the *Sitemap* tab, Click automatically generates and displays a graphical representation of the application as it has been defined so far (using AT&T's Graphviz library [2]). Figure 4 shows an example of a sitemap for a "ride board" application. 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). Solid lines show the control flow while dashed lines show the data flow (between pages and database tables). Besides providing a general overview, the sitemap helps developers to discover under-specification such as unreferenced pages or database tables (for example table "users" in Figure 4).

Finally, Click recognizes that EUDWeb will rarely occur solely on an individual level but rather that it is a collaborative process [14]. As a web-based system, Click easily supports multi-user projects. Each web application under development can have one or more "developers". Each of these developers can log into Click and modify the application. Because Click offers different layers of complexity and power (as we will describe later), one possible scenario is that a more novice developer asks an advanced colleague to extend Click by writing a custom component or behavior.

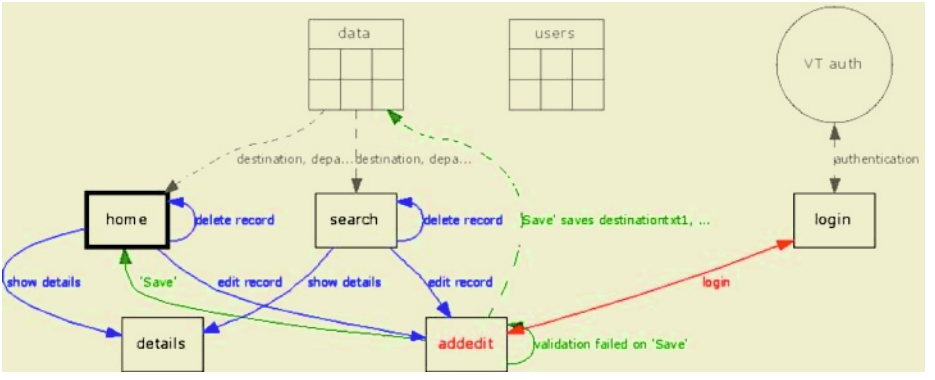

Fig. 4. A sitemap automatically generated by Click

4.3 Evolutionary Prototyping and Opportunistic Development

Supporting iterative and opportunistic development is a key design requirement for Click. Contrary to common code-generation approaches that make late changes to the user interface or behavior expensive to implement, *Click allows modifications to the layout, behavior, and database schema at any point in time*. Moreover, *changes take effect immediately*, thereby facilitating a rapid build-test cycle. We call this paradigm *design-at-runtime*. The design-at-runtime concept builds on the ideas of direct manipulation [24] and on the "debugging into existence" [22] observed for professional programmers working in a prototyping context. In its core it is similar to the automatic recalculation feature of spreadsheets. A critical piece of the concept is that the user is able to develop and use the application without switching back and forth between design and runtime modes (Click additionally provides an explicit preview-only mode; this requirement was discovered through formative usability evaluation). 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 whose behavior has not yet been defined. At this point the user is guided through a dialog to define the missing behavior. This interleaving of development and use continues until the whole application has been defined and tested. Of course, the usefulness of working with live data instead of placeholders at design time has been realized before. In Macromedia Dreamweaver MX, 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 support full use of an application–for example, hyperlinks do not work in this mode.

4.4 A Gentle Slope of Complexity

Tools for end users often have a low ceiling with respect to expressiveness. There is a natural tendency to hide complexity to improve usability, but the cost is often a concomitant loss of power. We hope to make EUDWeb highly expressive, and to provide a gentle learning curve to even greater power and functionality.

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

Fig. 5. Layers of Click's programming support that illustrate a "gentle slope of complexity"

Click's design provides several layers of programming support (see Figure 5).

At Layer 1, developers may customize existing web applications; ease-of-use is high but trades off with flexibility (assuming that existing applications are not complete matches). 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 (Figure 2; this is comparable to hand-editing HTML). The predefined high-level functions may be modified by editing the behavioral code (Layer 5; see Figure 3). 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), developers may access the component-based PRADO framework [27], 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 [9]. 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 7 (not yet implemented), experienced developers have full access to the capabilities of PHP. To gain ultimate flexibility, Click can export the full application code so that it may be used stand-alone on a separate web server.

We do not expect all users to take advantage of all layers. Rather, we 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. We also do not see these layers as a "natural progression" for developers as they gain experience. More probably the use of these features will be quite opportunistic and vary on an individual basis.

The layers summarized in Figure 5 are specific to Click but future web development tools may implement similar facilities, perhaps leaving out, changing or introducing new layers. Our intention is for Click to have a *gentle slope of complexity*: offering features and flexibility that grow proportionally with the developer's needs.

4.5 Implementation and Evaluation

Click is implemented in an object-oriented manner using PHP Version 5 [15]. MySQL Version 4 provides the database layer and the PRADO framework [27] provides an underlying extensible component model. PRADO exposes an event-driven

programming model similar to that of ASP.NET and JSP and cleanly separates layout from behavior. Click uses many open-source third-party components such as the HTMLArea WYSIWYG editor and a JavaScript-based drag-and-drop library and is itself freely available as open-source software (for references see [20]).

Click is still under development. A series of three formative evaluation sessions (4-6 participants each) has shown that novice web developers can implement a basic 3-page conference registration website within about one hour of time. Although many usability problems are left to be resolved, Click appears to facilitate the first steps into web engineering. However, we still have to evaluate how Click supports the construction of more complex projects. Just now we have begun to evaluate how novice developers manage when asked to autonomously implement non-trivial software (such as an online ride board application) from start to finish.

5 Summary of Contributions and Future Work

We have discussed the opportunities and challenges of EUDWeb and argued that supporting end users in web application development is not only a promising and important opportunity, but also a realistic endeavor. Our Click prototype demonstrates that we can provide high-level functionality that helps even nonprogrammers develop fully functional web-based data collection, storage, and retrieval applications. As an alternative to web engineering approaches that address problems in a top-down, model-based way, we support the natural tendencies of developers to work in a more opportunistic fashion. Also, we recognize that novice developers are likely to handle concrete representations (such as components on screen) more easily than abstract models of an application (e.g., content, navigation, or presentation models). Finally, we advocate web development tools that expose functionality in a layered fashion to facilitate a gentle slope of complexity.

Much work needs to be done before we can claim that end-user web engineering is a reality. We must validate the efficacy of the concepts of design-at-runtime and gradual introduction to layers of functionality. We must also continue to analyze and develop components best suited for the needs and skills of our target audience. The work we have presented here is an early step into the promising future of end user web development and we hope that other research will follow.

References

1. Ambler, A., J. Leopold (1998). Public Programming in a Web World. Visual Languages, Nova Scotia, Canada.
2. AT&T (2005). Graphviz – Graph Visualization Software. http://www.graphviz.org/
3. Brooks, F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. Computer Magazine. April 1987
4. Burnett, M., S. K. Chekka, R. Pandey (2001). FAR: An End user Language to Support Cottage E-Services. HCC – 2001 IEEE Symposia on Human-Centric Computing Languages and Environments, Stresa, Italy.
5. Ceri, S., P. Fraternali, A. Bongio (2000). Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. Computer Networks 33(1-6): 137-157.
6. Deshpande, Y., S. Hansen (2001). Web Engineering: Creating a Discipline among Disciplines. IEEE MultiMedia 8(2): 82-87.

7. Détienne, F. (2002). Software Design – Cognitive Aspects. Springer.
8. Fraternali, P., and P. Paolini (2000). Model-driven development of web applications: The Autoweb system. ACM Transactions on Information Systems, 28(4): 323–382.
9. Gaedke, M., C. Segor, H.W. Gellersen (2000). WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. 2000 ACM Symposium on Applied Computing (SAC 2000), Villa Olmo, Como, Italy.
10. Harrison, W. (2004). From the Editor: The Dangers of End-User Programming. IEEE Software 21(4): 5-7.
11. IBM (2005). IBM Rational Web Developer for WebSphere Software. http://www.ibm.com/software/awdtools/developer/web/
12. MacLean, A., Carter, K., Lövstrand, L., Moran, T. (1990). User-Tailorable Systems: Pressing Issues with Buttons. ACM. Proceedings of CHI 1990: 175-182.
13. Microsoft (2005). Visual Web Developer. http://lab.msdn.microsoft.com/express/vwd/
14. Nardi, B. (1993). A Small Matter or Programming – Perspectives on End User Computing. MIT Press. Cambridge, Massachusetts, USA, London, England.
15. PHP (2005). PHP: Hypertext Preprocessor. http://www.php.net/
16. Repenning, A. and A. Ioannidou (1997). Behavior Processors: Layers between End-Users and Java Virtual Machine. IEEE VL 1997. Capri, Italy. Sep. 23-26
17. Robertson, T. J., Prabhakararao, S., Burnett, M., Cook, C., Ruthruff, J.R., Beckwith, L., Phalgune, A. (2004). Impact of Interruption Style on End-User Debugging. ACM Conference on Human Factors in Computing Systems, Vienna, Austria, April 2004
18. Rode, J., M. B. Rosson (2003). Programming at Runtime: Requirements & Paradigms for Nonprogrammer Web Application Development. IEEE VL/HCC 2003. Auckland, NZ.
19. Rode, J., M.B. Rosson, M. A. Pérez-Quiñones (2002). The challenges of web engineering and requirements for better tool support. Virginia Tech Computer Science Tech Report #TR-05-01.
20. Rode, J., Y. Bhardwaj, M. Pérez-Quiñones, M.B. Rosson, J. Howarth (2005). Click: Component based Lightweight Internet-application Construction Kit. http://phpclick.sourceforge.net
21. Rode, J., J. Howarth, M. Pérez-Quiñones, M.B. Rosson (2004). An End-User Development Perspective on State-of-the-Art Web Development Tools. Virginia Tech Computer Science Tech Report #TR-05-03.
22. 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.
23. Schwabe, D., G. Rossi, S.D.J. Barbosa (1996). Systematic Hypermedia Application Design with OOHDM. ACM Hypertext '96, Washington DC, USA.
24. Shneiderman, B. (1983). Direct Manipulation: A Step Beyond Programming Languages. IEEE Computer. 16: 57-69.
25. Web Models (2005). WebRatio. http://www.webratio.com
26. Wolber, D., Y. Su, Y. T. Chiang (2002). Designing Dynamic Web Pages and Persistence in the WYSIWYG Interface. IUI 2002. Jan 13-16. San Francisco, CA, USA.
27. Xue, Q. (2005). The PRADO Framework. http://www.xisc.com