An End-User Development Perspective on State-of-the-Art Web Development Tools

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

[1] Center for Human-Computer Interaction
Virginia Polytechnic Institute and State University,
3160 Torgersen Hall, Blacksburg, VA 24061
jrode@vt.edu, jhowarth@vt.edu, perez@cs.vt.edu

[2] Pennsylvania State University, School of Information Sciences & Technology,
330D IST Building, University Park, PA 16802
mrosson@ist.psu.edu

Abstract

We reviewed and analyzed nine commercially available web development tools from the perspective of suitability for end-user development to compare and contrast alternative and best-of-breed approaches for particular problem areas within web application development (Getting Started, Workflow, Level of Abstraction, Layout, Database, Application Logic, Testing and Debugging, Learning and Scaling, Security, Collaboration, and Deployment). End-user development involves the creation of dynamic websites with support for features like authentication, conditional display, and searching/sorting by casual web developers who have some experience creating static websites but little or no programming knowledge. We found that current tools do not lack functionality, but rather have a variety of problems in ease of use for end users who are nonprogrammers. In particular, while many tools offer wizards and other features designed to facilitate specific aspects of end-user development, none of the tools that we reviewed supports a holistic approach to web application development. We discuss the implications of these problems and conclude with recommendations for the design of improved web development tools that would lower the entry barrier into web programming.

Keywords

end-user development, web development tools, web application development, web programming

Table of Contents

Introduction

Web publishing has ceased to be an activity for web professionals only. With increasing frequency, end users who are nonprogrammers are becoming web authors and creating websites for work-related and personal purposes. The level of sophistication in web design is increasing quickly; next to "flashy" online presentations, data-driven websites like online surveys, forms, and databases have become a major category of dynamic websites. Unfortunately, the web development tools provided for end users have not kept up with the fast pace. Tools like Macromedia Dreamweaver or Microsoft FrontPage enable nonprogrammers to autonomously create static websites, but they offer little if any help for developers without programming experience to create more sophisticated data-driven websites. For example, Macromedia Dreamweaver allows end users to specify dynamic features using a graphical user interface (GUI), but it fails to abstract non-trivial technical concepts such URL parameter passing or session management, which ultimately excludes a large potential user base.

In this paper, we report the results of a heuristic evaluation [17], of state-of-the-art web development tools according to how *usable* [18] they are for end users without programming experience. We adopt a holistic perspective on usability exemplified by this question: "Will an end user be able to use the tool to create a data-driven website of simple to medium complexity from start to finish?" Because the answer appears to be *no* for most current tools, we analyze specific features in an attempt to find the best-of-breed approaches, with the hope that these can form the basis of a more comprehensive solution.

The ultimate goal of the work reported here is to articulate the current approaches, metaphors, and solutions to commonplace challenges in web development that seem to be practical for end users who have little or no web programming experience. The work presented here is a first step towards providing tool designers with a source of inspiration and resources for the evaluation of potential design solutions. At the same time, we establish a framework of analysis that could be repeatedly applied to a different set of development tools.

In the balance of this paper, we first provide an overview of related work in the areas of web engineering, end-user web development, and commercial tools. Second, we describe our methodology for selecting and evaluating the tools and explain and motivate the dimensions of analysis and particular areas that we focus on during the evaluation. Third, we report our findings organized by web development problem area. We conclude with recommendations for designers of web development tools.

Throughout the paper we use the terms end user, nonprogrammer, and developer interchangeably to refer to a prototypical member of our target audience, namely a casual web developer who has some experience creating static websites but little or no programming knowledge.

Related Work

Two complementary domains of research and practice – *web engineering* and *end-user development* – have focused on methods and tools that could better support the web development needs of both programmers and nonprogrammers. Research in the domain of web engineering concentrates on making web professionals more productive and the websites that they produce more usable, reusable, 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.

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 (e.g. WebML [5]) to many light-weight code generators (e.g. [10], [29], [32]). 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. [14], [27]) 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 [21], 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 [30] 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 [8] proposes a taxonomy for web development tools that suggests some of the major dimensions of web development tasks. For example, he categorizes available web tools into visual HTML editors and site managers, HTML-SQL integrators, web-enabled form editors and database publishing wizards, and finally, web application generators.

Newman and Landay [16] 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.

In prior work we have also surveyed and interviewed web developers about the methods and tools that they use and the problems that they encounter during the development

process [22]. We concluded that web developers want more support for building secure and cross-browser compatible applications. The challenges of integrating many different technologies (HTML, CSS, JavaScript, ASP, SQL, etc.) and the difficulties in debugging web-based applications are also prominent issues. Finally, we learned that current tools often do not sufficiently support developers' tendencies and habits. For example, developers attempt to get the work done quickly and disregard issues such as accessibility if it requires too much time. In addition, developers prefer learning from and working with examples rather than using reference documentation.

Research in End-User Web Development

Well before the rise of the web, end-user programming (EUP) of basic data management applications has been a topic in academia and industry. Apple's HyperCard [2] is an early example of a successful EUP tool. More recently, web development research projects such as WebFormulate [1], FAR [3], DENIM [16], and WebSheets [31] have demonstrated specific approaches to end-user programming of web applications. WebFormulate is an early tool for building web applications that is itself web-based and thereby 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 by supporting 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.

In this paper, we will occasionally characterize problems using the terminology introduced by Green's cognitive dimensions framework [11]. This analysis framework was developed in the context of empirical studies of both programmers and nonprogrammers as they attempted to use a variety of languages and tools. To account for problems observed, the framework introduces a number of problem areas for programming languages including *viscosity*, the resistance to local changes to the code; *hidden dependencies*, dependencies between modules that are not readily recognizable by the programmer; *abstraction level*, new high-level concepts that need to be learned by the programmer; and *premature commitment*, having to make a decision based on information that is not yet known.

Commercial Web 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 [30] and the taxonomy of tools offered in [8]. Brief reviews of CodeCharge Studio, CodeJay, Microsoft Visual Studio, and Webmatrix from the perspective of productivity tools for programmers are can be found in [13].

In the area of web development, it seems that industry is more advanced than the research community — this is one of the main motivations for the investigation we report in this paper. The majority of research and rationale behind the design and development of commercial tools, however, is not publicly available and thus is difficult to incorporate into our analysis. Thus, one of our contributions is a framework for comparing and contrasting

commercial tools that we expect to be useful to others who seek to understand and categorize approaches and design decisions in commercial web development tools.

Method and Scope of Tool Evaluation

Expert Review of Web Development Tools

The ultimate goal of the work reported here is to identify currently used approaches, metaphors, and solutions to commonplace challenges in web development, which are practical for nonprogrammers.

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. This selection of features was grounded in previous work that analyzed existing data-driven websites [21]. To make these features more concrete and to convey our assumptions about a likely end users' goals and activities, we have constructed a reference scenario (Figure 1). As a supplement to the scenario narrative, we also provide a persona [6] for an informal web developer, Mark, who serves as a concrete proxy for the envisioned target audience (Figure 2). In the scenario, Mark would like to build what we feel is a typical example of a data-driven website – an online employee database similar to the vision depicted in Figure 3. These scenario-based analysis techniques enable specific points to be made about features and their usability implications [25], although at the same time they may promote conclusions that should be qualified by the hypothetical usage context.

Reference Scenario: Helena Inc. Online Employee Database

Mark, the marketing manager of the interior design company Helena Inc., would like to provide an online employee directory as a resource for customers. Visitors of the web site could browse and search the directory for employee information including full name, job title, picture, and contact email. Helena Inc.'s employees could use the same web site internally. Once logged on, an employee could also access more protected information such as employee phone numbers. In addition, Mark would have authorization to add new information and edit existing employee profiles.

Features of the website described in this scenario:

- Authentication (login) and authorization (3 levels: public, employee, admin)
- Add, edit, delete data record (with confirmation)
- Input validation (when adding or editing records)
- Conditional display (only show logout link when logged on)
- Browse/Datagrid view (alternating line color)
- Overview-Detail (clicking on name in overview leads to detail page)
- Sort, Search/Filter
- Record-set paging
- Upload (image) file
- Display uploaded image

Figure 1: Reference Scenario – Helena Inc. online employee database

Persona: Mark, the informal web developer

Although Mark is a businessman by trade, he enjoys the opportunities and challenges of web design. Helena Inc.'s web presence had been designed and developed by a web consultant three years ago. Mark has gradually taken over the website's maintenance and has been updating content and occasionally creating new pages by copying and modifying existing ones. Knowing that his boss will not see his idea for an employee database as a priority and will refuse the budget for outsourcing it to a professional developer, he decides that he will try to create this new piece of functionality on his own. Mark is familiar with basic web concepts such as pages and links and also has a vague idea of the web publishing process, but he has no programming experience.

Figure 2: Persona – Mark, the informal web developer

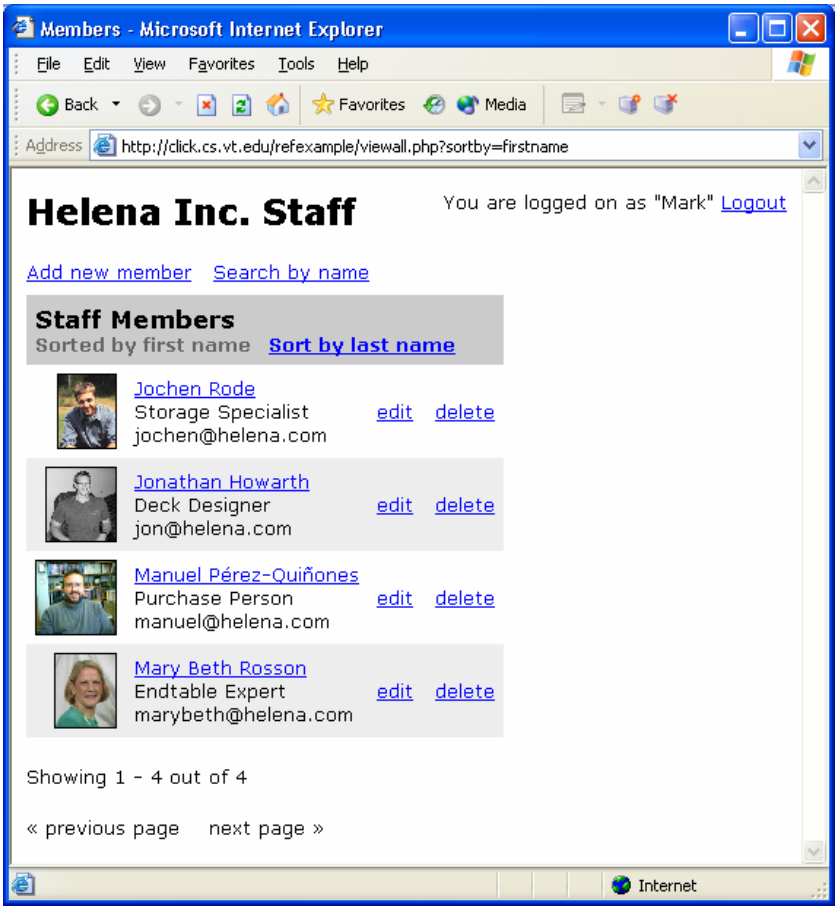

Figure 3: Reference implementation of Helena, Inc. employee database

Tools Reviewed

Due to the large number of commercial web development tools currently available, we had to make a selection for our review process. We selected tools based on both their apparent market dominance (e.g. Dreamweaver MX 2004, FrontPage 2003) and their potential sophistication (e.g. CodeCharge Studio, Visual Web Developer 2005). Although we did not use a formal method of selection, we followed many literature references and conducted several web searches based on combinations and permutations of "web application development", "rapid application development", "end-user", "nonprogrammer", and "tool."

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, form-centric tools, and website-centric tools.

Below, we describe the three categories and give examples of tools that fit into them. We reviewed only one tool in the database-centric and form-centric categories because tools in these categories do not provide the functionality required to develop our target class of applications (i.e., as exemplified by the reference scenario in Figure 1).

Database-centric Tools

Database-centric tools are primarily intended to help end users put databases online for viewing and editing purposes. These tools often automatically generate an online interface for a database and provide few if any options for customization of that interface. From an online application development perspective, these tools have limited functionality. However, tools in this category do generally provide end users with a convenient and efficient method for sharing data in databases.

As a state-of-the-art example of a database-centric tool, we reviewed FileMaker Pro 7 (FMP). This tool is a database that stores information in local files, and as a result, does not require the configuration of a database server. FMP has wizards that help end users set up tables, and four modes (browse, find, layout, and preview) and three views (form, list, table) for populating and displaying those tables. The web publishing wizard makes it easy to activate the built-in web server and put databases online. Overall, the tool allowed us to rapidly create a database and make it available through the web. The major limitation is functionality; the tool simply does not have some of the features that would be needed to develop the application in the reference scenario (Figure 1). In particular, the tool does not allow the end user to significantly customize the look and feel or conditionally display data. We reference FMP in the remainder of the paper primarily for its positive design features.

Form-centric Tools

Developing forms, particularly surveys, is a common requirement of end users, and several tools have been developed to support this task. These form-centric tools help end users create forms for collecting data; they also help in storing the resulting data and retrieving it for analysis purposes. Much like database-centric tools, form-centric tools are specialized tools that contain a subset of the functionality required for end-user web application development of the kind exemplified by our reference scenario (Figure 1).

We reviewed Quask FormArtist (FA) as an example of a commercial online form development tool. FA begins by displaying a wizard that has the following options: design, deploy, notify, fetch data, and analyze. Design involves creating forms with a pixel-based graphical user interface (GUI) editor. Condition editors allow end users to validate input and make branching decisions based on the content of fields. The condition editor generates statements in a proprietary scripting language that cannot be edited directly. FA generates either ASP or PHP code, so deploying a survey involves putting the necessary files on an appropriate server; a wizard facilitates this process. The tool also provides functionality for retrieving data and analyzing it. The tool is well designed, and we were able to quickly develop and deploy a survey. Similar to FMP, the main limitation of this too was functionality. For instance, it would not be possible to implement any of the data management features in the reference scenario such as overview/detail relationships or searching, sorting, and filtering. Although FA is not intended for the development of a full web application, we reference it throughout this paper to illustrate design decisions.

Several other form-centric applications exist; SurveyMonkey.com and FormSite.com are two examples of popular web-based tools. Microsoft InfoPath, a desktop application, is one of the most advanced form-creation tools of which we know. InfoPath, however, has no web deployment option and creates forms that can only be filled in by other users of Microsoft InfoPath.

Website-centric Tools

We categorize software applications whose primary purpose is assisting the user with the creation of static or dynamic websites as website-centric tools. Tools in this category allow users to create web pages, specify behaviors, and deploy the resulting pages online. These tools have support for integrating existing data sources such as databases or XML files and sometimes feature built-in database wizards that help users create new databases. Note that within the group of website-centric tools, there is a considerable difference between tools targeted at programmers and tools targeted at nonprogrammers. We decided to include tools intended for programmers because we expected them to offer features that would be useful to nonprogrammers if they could be made more accessible. Although the lines dividing programmer versus nonprogrammer tools are blurry, we have used the distinction as a heuristic in classifying the seven tools analyzed in the website-centric category

Tools Targeted at Programmers

Many web application developments tools are clearly too advanced for nonprogrammers, but instead are designed as productivity tools for programmers. They automate many common tasks such as generating an HTML data table from database fields or getting data from a form, but they require programming to link together the various pieces of an application. Because they require programming, they are particularly flexible and allow for the creation of a practically limitless variety of applications. These tools also often facilitate learning by allowing the developer to view and edit automatically generated code. In addition, many of these tools encourage good programming practices because they help to enforce structure and organization.

As examples of applications within this category, we reviewed Microsoft's Visual Web Developer (VWD) and YesSoftware's CodeCharge Studio (CCS). These two tools share

features that separate them from the other website-centric tools. For example, both have interfaces that are similar to those of integrated development environments for programming with frames that contain property-value pairs for the currently selected object and status frames that contain compilation and other low-level messages. Also, both tools rely heavily on the tool user's understanding of technical terminology (e.g. "data source", "class").

Tools Targeted at Nonprogrammers

The website-centric tools for nonprogrammers are intended to allow end users with little or no programming knowledge to create data-driven websites. Not surprisingly, the majority of the tools that we reviewed were in this category, including H.E.I. Informationssysteme RADpage (RAD), Instantis Sitewand (SW), Macromedia Dreamweaver MX 2004 (DW), Macromedia Drumbeat (DRB), and Microsoft FrontPage 2003 (FP). These tools take a variety of approaches to enabling end-user web application development. In general, these tools are similar in that they facilitate the creation of common web page components and allow end users to increase the complexity of their applications as they become more experienced.

List of Tools

Table 1 lists of the nine tools that we reviewed. The table lists the name of the tool, the abbreviation that we use in this paper, and a short summary of its focus and features.

Tool	Description
Database-centric	
FileMaker Pro 7 (FMP)	FMP is primarily intended to be a database tool, but it has a web publishing feature that allows users to easily put their databases online for viewing and editing purposes. FMP includes its own built-in web server that requires minimal setup and can be turned on and off through the user interface. The online interface is very similar to the tool's interface and is thus easy to use, but not customizable. FMP is not intended to support complete end-user programming.
Form-centric	
Quask Form Artist (FA)	FA is a desktop-based tool for creating multi-page forms and online surveys. Although it is not a full web application builder, it stands out for its ease of use. FA uses a pixel-based layout and provides wizards for many tasks. Simple behavior can be specified via forms, but more sophisticated behavior such as if-then branching logic requires the use of a built-in high-level scripting language (some wizards are provided). Forms are deployed via a wizard to either the Quask Test Server or any other server that can support the generated ASP or PHP code.

Website-centric Programmer	
Microsoft Visual Web Developer 2005 Beta (VWD)	VWD is a desktop-based WYSIWYG and ASP.NET 2.0 code editor. Layout is defined visually via drag-and-drop. High-level server controls such as the GridView control allow the end user to visually develop large portions of the website's behavior, but much still needs to be specified manually as ASP.NET code. Along with the 2.0 version of ASP.NET, this tool is currently in the beta-stage. VWD is the future version of the web component of Microsoft's Visual Studio. A built-in web server that does not require manual setup facilitates rapid testing of applications. It is most likely to appeal to web programmers as a productivity tool.
YesSoftware CodeCharge Studio (CCS)	CCS is a desktop-based web application builder that supports a large number of languages including ASP, ASP.NET, ColdFusion, JSP, Perl, and PHP. It has an application builder wizard that greatly simplifies the development of an application based on a database schema. CCS includes productivity features such as a large number of predefined themes and the ability to easily set up security groups and permissions. Although it is possible to generate a web application with the wizard, the tool is primarily intended for users with programming experience.
Website-centric Nonprogrammer	
H.E.I. Informations-systeme RADpage (RAD)	RAD is a web-based web application builder. Although its UI is somewhat unappealing, much of the layout can be specified within the tool using its small set of HTML layout tools, but RAD advocates the use of an external web editor for the "finishing touches." Its major strength is the definition of the application's behavior, which can be fully specified using a form-based UI. RAD introduces HeiTML, a language that builds on the syntax of HTML and offers high-level tags to perform operations such as displaying data in table format or sending emails. The developer can also edit HeiTML manually to gain more flexibility. As soon as the developer specifies a behavior on a page, RAD shows a live view of the page similar to how a visitor would see it.
Instantis SiteWand (SW)	SW is a web-based visual rapid application development tool. End users define the layout using an external web editor. After the page layout is uploaded, the behavior can be specified using the concept of engines, which process incoming form data and communicate with associated data tables. Engines can be combined to produce sophisticated applications. All the behavioral specification is form based. Although the tool is easy to use for very simple applications such as email feedback forms, the use of proprietary technical jargon makes it difficult for nonprogrammers to use it for more sophisticated applications.
Macromedia Dreamweaver 2004 MX (DW)	DW is a desktop-based WYSIWYG web editor marketed to programmers and nonprogrammers alike, although its features are not beginner-friendly. DW has a large number of features including templates, site management support, and a CSS editor, but it fails in guiding beginners. Wizards are only available for a small number of features. The concept of server behaviors allows the technically savvy user to generate ASP, PHP, ColdFusion code.

	Users can edit in design or code view.
Macromedia Drumbeat 2000 (DRB)	DRB is a desktop-based WYSIWYG web application editor with many leading edge end-user friendly features. The screen layout is pixel-based, which simplifies positioning. DRB includes a database engine, but can also connect to external data sources. It automatically creates HTML, CSS, and ASP or JSP code, but does not allow the end user to view or edit this code. DRB has been discontinued by Macromedia and certain features have been merged into DW (negatively affecting ease of use).
Microsoft FrontPage 2003 (FP)	FP is a desktop-based WYSIWYG web editor primarily targeted at nonprogrammers. Predefined templates and themes are included, which help end users get started quickly. FP provides data-driven features that work in conjunction with Microsoft FrontPage Server Extensions and Microsoft SitePoint Server's web parts. Wizards are available to help automatically generate ASP/ASP.NET code.

Table 1: Tools reviewed for the purpose of this investigation

Web Development Problem Areas

We have organized the discussion of the reviewed tools into the general problem areas listed in Table 2. We identified these problem areas through an initial high-level review of several of the tools. Most of the concerns in these areas resulted from taking the perspective of an end-user developer wanting to accomplish design goals similar to those found in the reference scenario (Figure 1). However, some of the concerns are not only relevant for developers but also for the visitors of the website that is being developed. For example, in many situations it is important that web developers understand and implement security options and strategies, so that visitors to their sites can securely conduct transactions or submit information.

Problem area	Description
Getting Started	Support provided during the initial phases of development
Workflow	General process for creating a web application
Level of Abstraction	Abstractions provided by the tool, e.g. components, wizards
Layout	Defining the look of the web application
Database	Creating and accessing a database
Application Logic	Defining the behavior of the web application
Testing and Debugging	Helping to identify and solve errors
Learning and Scaling	Support learning and growing beyond the "ceiling" of the tool
Security	Level of security of produced web applications
Collaboration	Supporting multiple developers
Deployment	Assisting in moving a ready application into production mode

Table 2: Web development problem areas

Results

We present the results of our tools review organized by particular web development problem areas (Table 2) in an attempt to facilitate comparison. Within each problem area, we describe specific approaches and tradeoffs associated with those approaches that we encountered while attempting to implement or determine how to implement parts of the reference scenario. Our reference scenario (Figure 1) required a tool for developing the look and feel and specifying the behavior of the application, a database to store employee information, and an application server to make the employee directory available online.

We reviewed the tools independently and made notes based on the problem areas. We then met to compare notes and discuss the approaches that we observed. We have different web programming backgrounds, and the joint review of notes was particularly insightful because it allowed us to merge our different perspectives.

Getting Started

Understanding Available Options

Getting started is often one of the most difficult steps in using a web development tool because some amount of knowledge or research is necessary to understand the various web configuration options that a tool supports. Some tools such as SW are web-based and provide few, if any, configuration options, thereby trading flexibility for ease of use. Desktop tools, on the other hand, often provide a variety of language, application server, and database options at the expense of a relatively steep learning curve. For example, CCS supports six languages (ASP.NET, ASP, ColdFusion, JSP, PHP, and Perl), five application servers (BEA WebLogic, IBM WebSphere, Microsoft IIS, Macromedia Coldfusion, and Netscape Enterprise Server), and five databases (IBM DB2, Microsoft Access, Microsoft SQL Server, MySQL, and Oracle) (Figure 4). A developer with an understanding of these technologies can use them to create powerful and efficient applications, but end users new to web programming are likely to be overwhelmed.

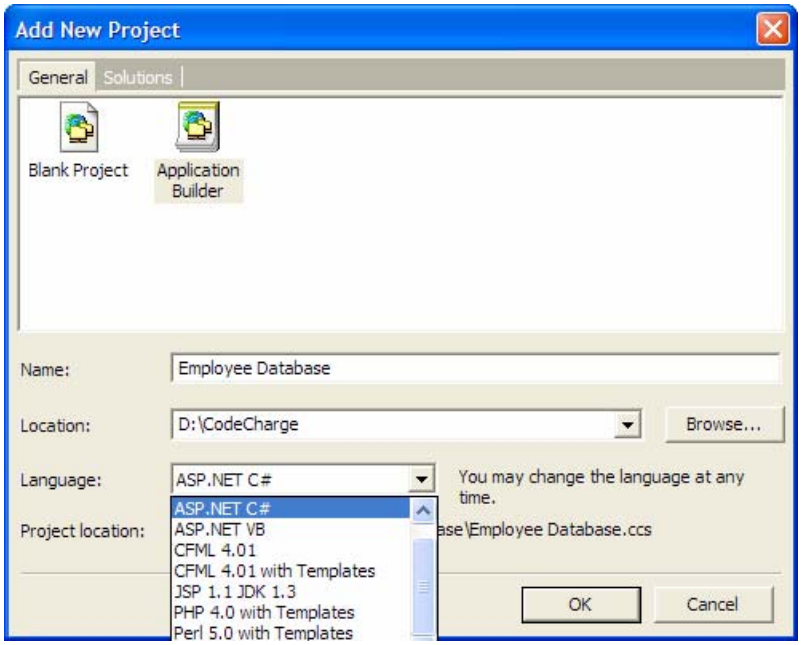

Figure 4: Variety of languages supported by CCS

Installation and Configuration of Tools and Servers

The installation of the tools that we reviewed was a straightforward process, but the installation of the associated database and application servers was considerably more difficult. DRB, although otherwise beginner-friendly, requires the user to install the Java Development Kit, IBM WebSphere application server and the IBM DB2 database engine. Complex preparatory activities like this are not only confusing to beginners but may quickly deflate whatever initial motivation they have to use the new web development tools [4].

Some tools simplify the process by integrating necessary server support. FMP, for example, provides a built-in production web server that can be controlled through the tool interface. A downside of built-in production web servers is that they are generally not intended to support large numbers of simultaneous users and thus set an upper limit on scalability.

VWD integrates a zero-configuration test server that is started automatically after the tool is launched. The tools that did not have built-in servers require the configuration of external database and application servers. While external servers provide more flexibility, they also require that the end users either learn how to configure and maintain them or have access to a managed hosting service.

Initial Use

A key aspect of a tool is how well it supports the end-user's initial attempt to develop an application. Some tools such as FA have a wizard that guides the end user through the creation of a project while others like DW take a more unstructured approach and provide the user with an empty workspace. The wizard approach is particularly effective for novices because it guides them through steps that they may otherwise skip or intentionally omit. Wizards, however, can also be unfavorable if they are too lengthy or detailed or if they require too much premature commitment. DRB, for example, asks a large number of

questions and requires users to select among options that may not be understood until later in the development cycle. In addition to wizards, examples can be a particularly effective teaching tool. CCS provides a number of applications that can be customized by the end user; in fact, it would be possible to customize a pre-defined directory application to serve as our employee directory.

Workflow

Workflow is concerned with the process that developers follow to create a web application. The tools that we reviewed take one of two basic approaches: some begin with the database and generate the necessary code based on the database schema, while others begin with the design of the site and add to the database as necessary. The application builder of CCS, for example, assumes that the database design is complete and uses a wizard to determine which tables and fields to use to create the application. Such an approach improves data design because it requires developer to plan ahead. It has the drawback, however, that many aspects of the application may not be known *a priori* and must be discovered as the application is developed. Having to update the database and continually regenerate the application is tedious and time-consuming.

A further complication is that the software applications with which our target group of end users is most likely familiar do not require the sort of analysis and planning assumed by tools like CCS. Mark, the marketing manager in our scenario is probably most familiar with office productivity applications such as word processors or spreadsheets, which are intended to facilitate rapid design changes. Tools that allow the end user to design the site first, such as SW or FA, support a more incremental process in which the user can focus on his or her end goals (how the site should look and operate) and count on the system to update the database automatically as components that use the database are added or removed.

Level of Abstraction

At the lowest level of abstraction is a tool that provides no built-in components and requires the end user to manually code the necessary components. At low levels of abstraction, the end user is intimately familiar with the functioning of components and how they communicate with one another. Few end users, however, have the time or the programming expertise to build such components. Thus it is not surprising that the tools in our survey all provide some level of abstraction. The tools with lower levels of abstraction provide smaller components that need to be combined by the end user. For example, DW provides “repeated regions” and dynamic text objects that can be combined to format and present the contents of a database table. Less abstract components like these have the advantage that they are relatively customizable, but have the disadvantage that they require more time to configure and use.

Tools such as VWD provide more abstract components like a GridView, which creates a web-viewable table from a database table (Figure 5). The user need only specify the desired fields from the database table, and VWD generates the necessary code. RAD provides components at both levels of abstraction; dbdisplay components are equivalent to GridView components while dbselectlink components deal with links within a single

column within a data table. As an additional example of abstraction, SW uses engine components, which process requests and return results.

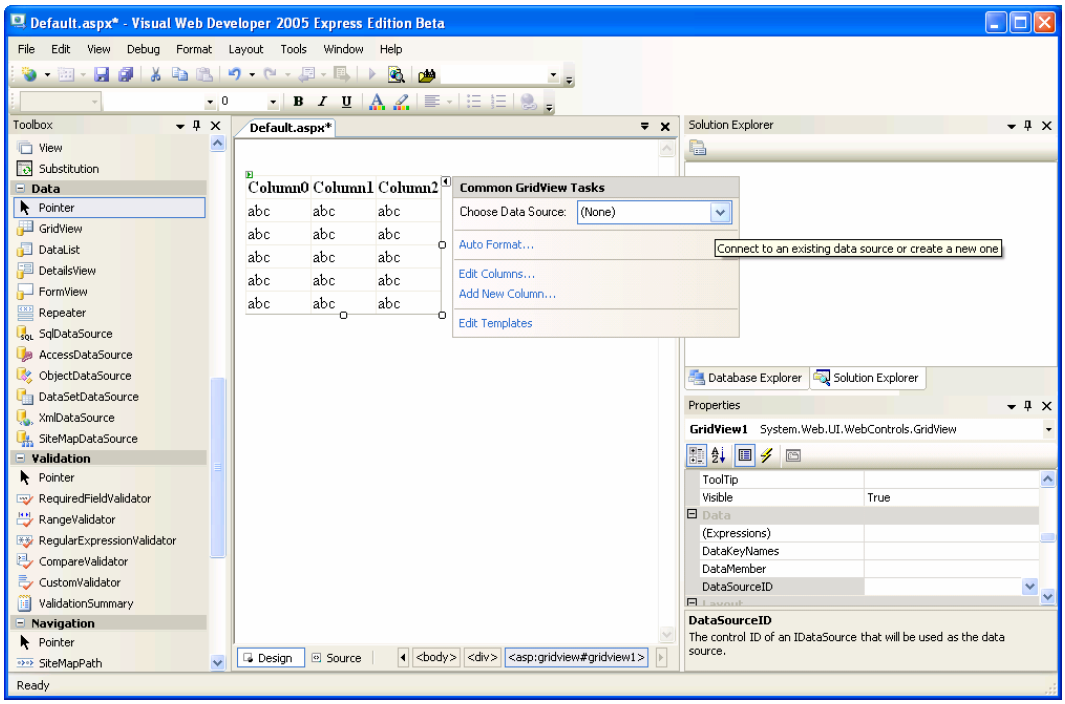

Figure 5: GridView component in VWD

Layout

Stand-alone Tool versus Reliance on an External Layout Editor

One of the most basic differences between the tools that we reviewed is the extent to which they help the end user to design the page layout. At one extreme are tools like DRB that encapsulate all necessary visual layout functionality within the tool and make it very difficult or even impossible to change the design outside of the tool. At the other extreme are tools like SW that do not have integrated layout capabilities and rely entirely on the developers' use of an external web editor. The remaining tools use a solution that is a compromise between the two extremes. RAD, although it includes a rudimentary WYSIWYG editor, is primarily concerned with helping end users implement behavior and advocates the use of an external WYSIWYG editor to "polish the look." Tools such as DW and FP allow the user to fully design the page layout and also work well with external tools as necessary.

HTML-flow-based Positioning versus Pixel-based Positioning

Because most web applications make heavy use of HTML, the layout editors of most tools that we reviewed are HTML-flow-based, meaning that the user needs some understanding of constructs like HTML tables to create a nontrivial layout. In informal usability sessions with novice web developers, we observed this to often be a significant problem [20].

However, as tools like DRB or FA show, an HTML-flow-based layout editor is not the only possible solution. DRB's layout editor, for example, is completely pixel-based (Figure 6). Users can use standard drag-and-drop techniques to position components on the screen at pixel accuracy. An optional snap-to-grid feature assists positioning. DRB then automatically creates the HTML code necessary to implement the layout (making heavy use of HTML tables and spacer images). We were surprised to discover that, although CSS2 (Cascading Style Sheets) positioning is now widely supported, no tool uses it for its implementation. We speculate that the reason for this is an effect of the timing of this standard and the development time of the tools as well as limitations of fixed layouts for deployment in heterogeneous environments.

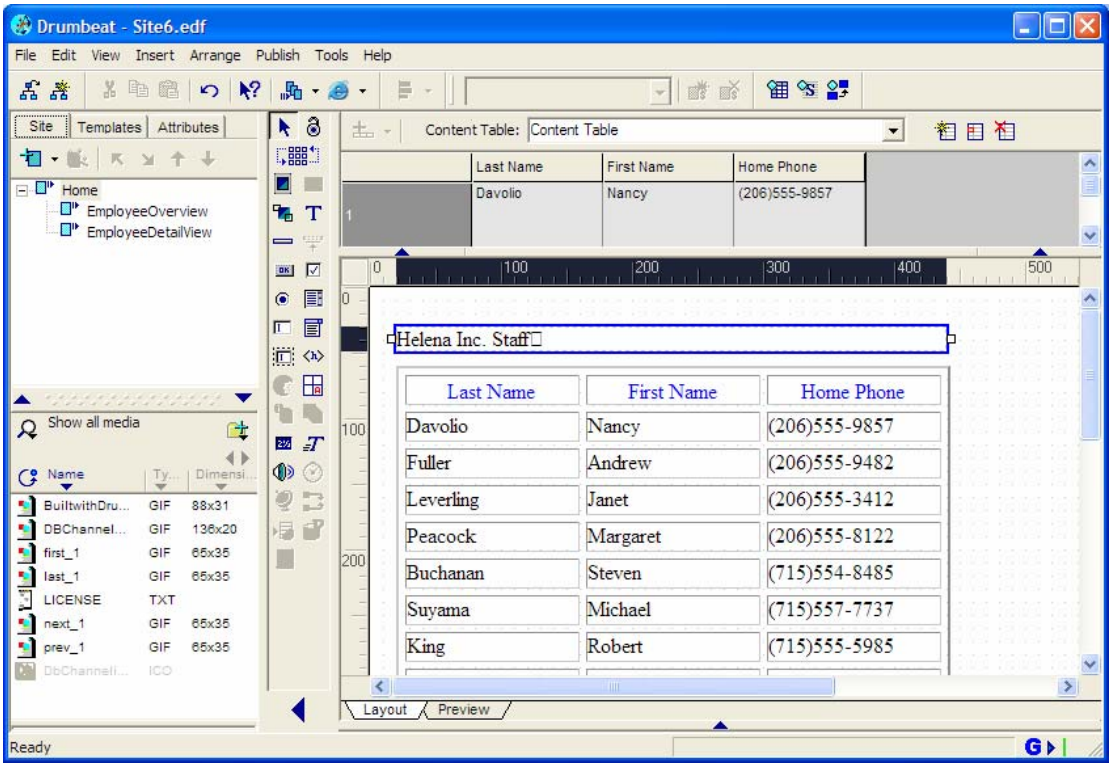

Figure 6: Pixel-based layout in DRB

Templates, Themes, and Skins

Some tools such as DW assist the user in implementing a consistent look and feel across different web pages. Templates, an approach pioneered by DW and now available in many tools, allow the user to create one or more layout templates on which new web pages can be based. This functionality can be used, for example, to implement a common header and footer or to create a pleasing but complex layout which inexperienced web developers can reuse. Any change to the template can automatically be applied to all dependent web pages, thereby simplifying global changes. VWD provides layout inheritance, a concept similar to templates.

Yet another approach to ease the design of attractive and consistent looking pages is the concept of themes. Themes are pre-defined and often user-customizable style sheets that

determine attributes such as fonts and colors. Typically, tools supporting themes such as CCS make it fairly easy to change the theme of any page at any point in time. Finally, some tools make it possible to change the look of individual components. So-called “skins” or “schemes” can be used to modify the look but maintain the behavior of components such as the menu control in VWD.

Database

Database Creation and Connection

All the website-centric tools that we reviewed include support for a database that contains the data to be displayed or manipulated by the web application. The degree of integration, however, varies from tools like DRB, in which the end user is able to define a database schema and add and edit data within the tool, to DW, in which the end user is only able to define and use a connection to an already existing database. While the latter may be sufficient for users who already have a database that they simply need to publish online, it may create a difficult problem for an end user starting from scratch.

We have observed considerable differences in the user interface used for specifying connections to existing databases. FP provides a positive example because it creates a database in a desired directory and creates an appropriate connection to the database. A less positive example is the DW database wizard, which from the perspective of a nonprogrammer, uses too much technical jargon (e.g. “connection string”, “OLE DB provider”) and provides unhelpful error messages (e.g. “an unidentified error has occurred”).

The SW approach to the specification of a database schema is different from all the other tools that we have reviewed because it automatically derives the data schema from the form elements contained in an associated web page.

DRB also provides a unique feature in that it handles binary data, such as images, as “first-class” field types. DRB even shows a thumbnail for each image contained in a database table.

The handling of late changes to the database schema is a problem area that some tools address better than others. This issue becomes a major problem when pages have been automatically generated by a wizard based on an initial database schema and subsequently customized by the user, but later need to be updated to handle database changes.

Accessing the Database

The lingua franca for communicating with the database is SQL - even for nonprogrammer tools. However, most tools also offer a query builder feature that allows the user to specify a query. CCS, for example, provides a visual query builder that shows relationships among tables and has options for showing SQL statements and the actual values that are returned by the query (Figure 7).

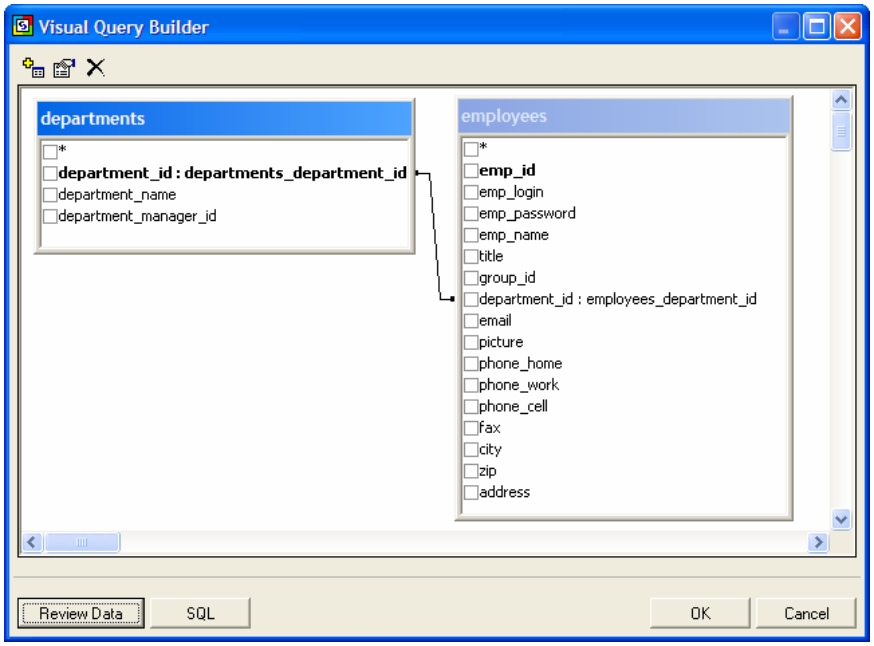

Figure 7: Visual Query Builder in CCS

None of the tools that we reviewed offered an outstanding solution to the problem of specifying complex Boolean conditions. The state of the art, even though sometimes masked behind buttons and dropdown fields, is still the use of nested parenthesis along with the operators "and" and "or". Research has shown that nonprogrammers often do not use "and" and "or" in their Boolean interpretation [19]. Therefore, we see the creation of a powerful, yet nonprogrammer-friendly condition builder as an opportunity for groundbreaking research.

In addition to building the actual SQL query, there are issues associated with getting the necessary values for the query. Web applications have a potentially large number of places from which they receive information. For example, values may be stored in form data sent by another page, in the query string, in cookies, or even in server or session variables. The variety of data sources may be overwhelming to and end user new to web application development.

The tools in our review take a variety of approaches to giving end users access to these various data sources. When using DW, for example, an end user must define one or more recordsets (DW's term for queries) for each web page that should communicate with the database. A recordset, defined individually for each web page, can use "URL parameters", "Form variables", "Session variables" and other parameters as filters. Such an approach is flexible, but the heavy use of technical jargon makes it not very novice-friendly. DRB and RAD, on the other hand, use the concept of the current row (a virtual pointer to the current data row of interest) as a high-level abstraction in referring to a particular row in the database; this hides from users the internal use of keys and id fields to select a particular record. While interesting, it remains to be shown by user testing whether this is more intuitive than informing end users about the need for appending *id* information to *edit* or *detail* links.

Application Logic

Session Management

Due to the connectionless nature of the HTTP protocol, most web applications need to handle the problem of remembering the application's state for a given visitor between requests for web pages. Some tools expect the developer to manually implement session management, while other tools and technologies like VWD/ASP.NET automatically add a persistence layer. For example, VWD text box controls remember the text that the web page user has entered even after the form containing the text fields has been submitted, a very useful behavior that a nonprogrammer is likely to expect.

Actions

An important step in the process of creating a web application is defining what actions to take when certain conditions are true. Tools like FP, which are comparatively novice-friendly in this regard, expose high-level actions such as validating form data and displaying error messages, sending form data to an email address, and adding form data to a database. DW offers predefined server behaviors such as insert, update, and delete record (Figure 8).

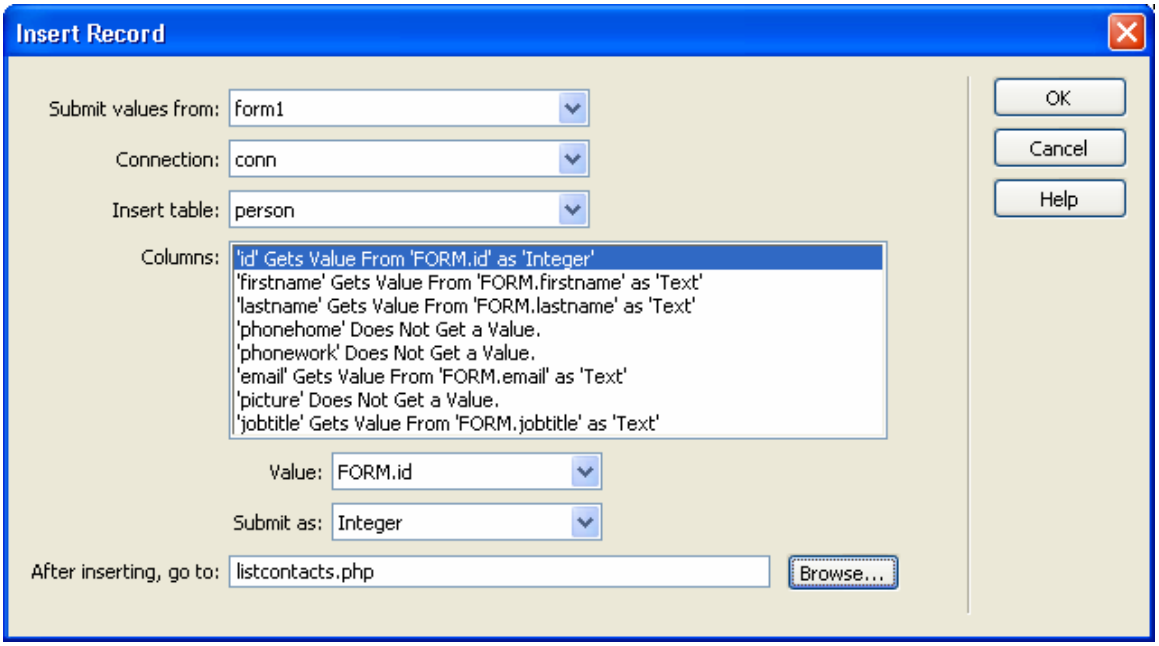

Figure 8: Insert Record Server Behavior in DW

FP and DW associate actions with the HTML <form> tag. Other tools like RAD tie actions to submit buttons. SW offers an end-user friendly interface to defining the conditions under which actions should be executed. SW lets the developer define integer values that represent the evaluation order of conditions, for example to define conditional storage actions (Figure 9).

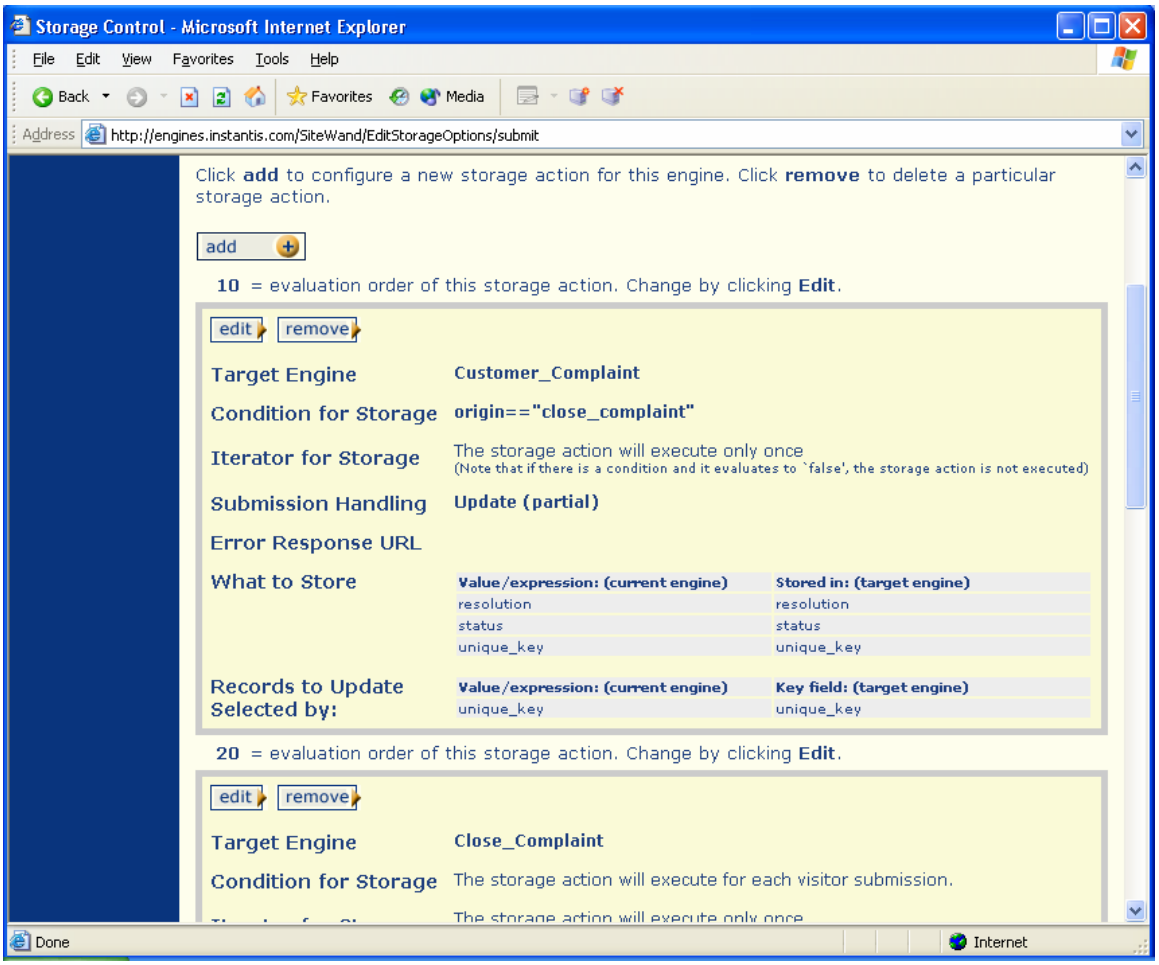

Figure 9: In SW each action has an associated evalution order (e.g. 10 or 20 in the picture above)

Displaying Repeated Information

Often, web applications must retrieve, present, and manage a large set of records of a similar format. These records need to be displayed in a consistent fashion. The majority of the tools in our review provide the concept of repeating elements. DW, for example, has the concept of regions; repeated regions delineate segments of a page that should be repeated for each row in a recordset. Repeating elements are often used to create overview-detail pages that consist of one overview page that displays summaries of multiple items, with each item having a link to a page that contains more details. Our reference scenario (Figure 1) uses the overview-detail pattern to display a list of employees; each row has an "edit" link that allows a visitor to view and change the detailed data. As an example, DW offers, although somewhat hidden within its menu hierarchy, a "Master Detail Page Set" wizard that guides the user through the process of creating a set of connected server behaviors.

Testing and Debugging

Iterating between Building and Testing

Through our own informal observations, we have found that nonprogrammer end users use an incremental approach to constructing an application; they construct small pieces and then test them before continuing with development [20]. The continuous testing helps to reassure end users and lets them know that their application is functioning as expected. The tools that we reviewed take three basic approaches to providing end users with the ability to test their applications.

The first approach, exemplified by FP, is an explicit preview mode. This approach requires that the end-user put files on a test application server either manually or through an option in the interface to view them.

RAD uses the second approach, which we refer to as design-at-runtime or programming-at-runtime [21]. With this approach, the application is completely functional as it is being constructed. For example, if an end user clicks on a button in the design view and an action has been defined for that button, the button will trigger the action. Similar to the automatic recalculation feature in spreadsheet applications, design-at-runtime allows for the most immediate feedback. Tanimoto has coined the term *liveness* to describe how a software development tool gives semantic feedback [28]. This approach seems to be most consistent with our observations of end users' incremental development style, as it enables a seamless transition between development and testing of pages and features. A possible tradeoff is that developers may be encouraged to take a more "piecemeal" view of the work that they are doing (i.e., focusing on it as a feature by feature extension) and fail to appreciate the overall structure of the application.

The third approach is a blend of the first two approaches. DW uses a live data view, which allows end users to view pages with dynamic data in the design view. DW accomplishes this by sending a temporary copy of the current page to the application server and then displaying the results. Live data view does not simulate a fully functioning application because interface elements such as hyperlinks or buttons are not functional.

Identifying and Recovering from Errors

Web development tools take three basic approaches to helping end users address the problem of (potential) errors.

The first approach is to help users to avoid errors. Some tools like DRB constrain the options they present, by making them context-sensitive; such tools may also prevent direct access to the underlying implementation by the developer, which in effect, eliminates the chance for syntax errors.

The second approach is to build support for identifying errors into the tool. RAD helps users identify errors as they are defining the behavior and suggests solutions (Figure 10). This approach may not work well for tools that require an external application server (e.g. FP, DW) because they have limited abilities to detect errors; the execution of the code is so highly dependent on the configuration of the server.

The third approach is a built-in debugger that allows end users to step through the execution of a section of code. None of the tools in our review include such a debugger, but other existing web development tools such as Zend Studio (www.zend.com) have complete debuggers. A debugger has the potential to be very powerful, but it requires some understanding of how programs execute and, as a result, may be difficult for

nonprogrammer end users to effectively apply. However, if debuggers tailored to novice programmers can be developed (e.g. using visual representations and interaction techniques) [15], they may offer important opportunities to learn through exploration of an existing code base [24].

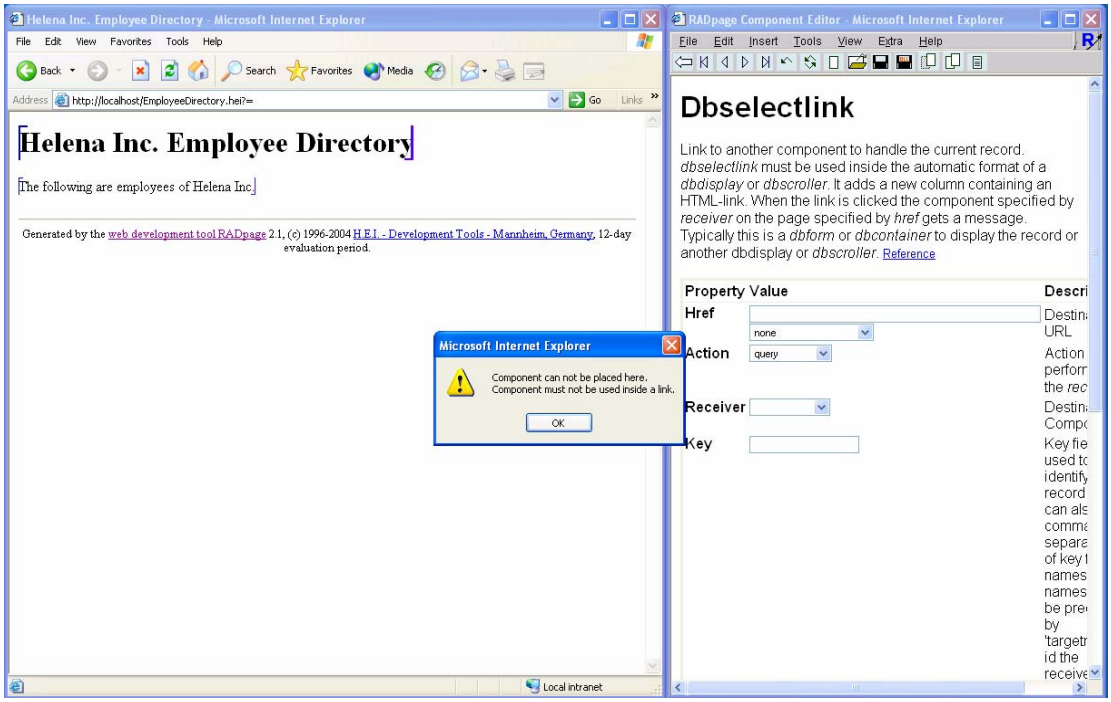

Figure 10: Error message in RAD while defining a behavior

Learning and Scaling

Viewing Code

Viewing code produced by a tool is useful for helping end users learn programming syntax and semantics. The tools in our review use a variety of approaches to displaying code. One approach used by RAD is to show code by component – hovering the mouse cursor over a component shows its HeiTML implementation in a popup window. Such an approach helps end users isolate code segments, so that they are not overwhelmed by the code surrounding the component. DW provides a split screen approach that displays the design view and the code view. Code in the code view that corresponds to selections in the design view is highlighted, which helps end users develop mappings between the two. A third approach used by VWD is a feature that hides sections of code for ease of view; for example 50 lines of HTML table code can temporarily be displayed as an ellipse ("…").

Editing Code

A major issue with editing code is whether or not the tool allows the end user to directly modify the code. Some tools like DRB have a hidden implementation that the user can only modify through the interface. While such an approach simplifies the process of application creation for end users, it also places a firm limit on what is possible to do with the tool. The

tools that do allow for the direct editing of code either use a proprietary language like RAD's HeiTML or a standard language such as PHP. In either case, tools that produce code similar to what a programmer might write manually can simplify code generation and editing and help to teach end users good programming practices. Tools that maintain the format of user-written code such as VWD are more likely to be preferred by the end-user over tools that reformat code such as older versions of DW [22]. Another feature that simplifies editing is an indication of where the end-user should insert custom code. DW provides comments in the code for this purpose.

Separation of Layout and Behavior

The tools in our review take two approaches to integrating the layout and behavior code. DW and FP, for example, integrate the layout accomplished via HTML and the behavior code in the same file. Such an approach helps end users see how the layout and behavior interact and is generally more direct and novice-friendly, but the mixture of layout and behavioral specifications can also lead to complicated text files that are difficult to read and modify. VWD and CCS use another approach that separates the layout from the behavior. This model that keeps the "code behind" the view is targeted at programmers and creates less complex text files. End-users, however, may have trouble understanding how the behavior specified in one file interacts with the layout specified in another file.

Extending Applications

The major issue with learning and scaling is how well end users are able to use a tool to develop their applications as their experience and understanding increases. Tools that have a "hard ceiling" require end users to leave the environment and support of the tool to create more complicated layouts or behaviors. For example, RAD requires that end users move to an external web editor if they wish to refine the look or layout of an application. In comparison, tools with a soft ceiling allow the end-user to perform increasingly complicated tasks within the tool. These tasks might involve creating more complicated layouts and behaviors by directly editing code or, in tools such as DW, extending the architecture at different levels by creating custom server behaviors (Figure 11), or writing custom plug-ins that integrate into DW's user interface using HTML and DW's extended JavaScript API.

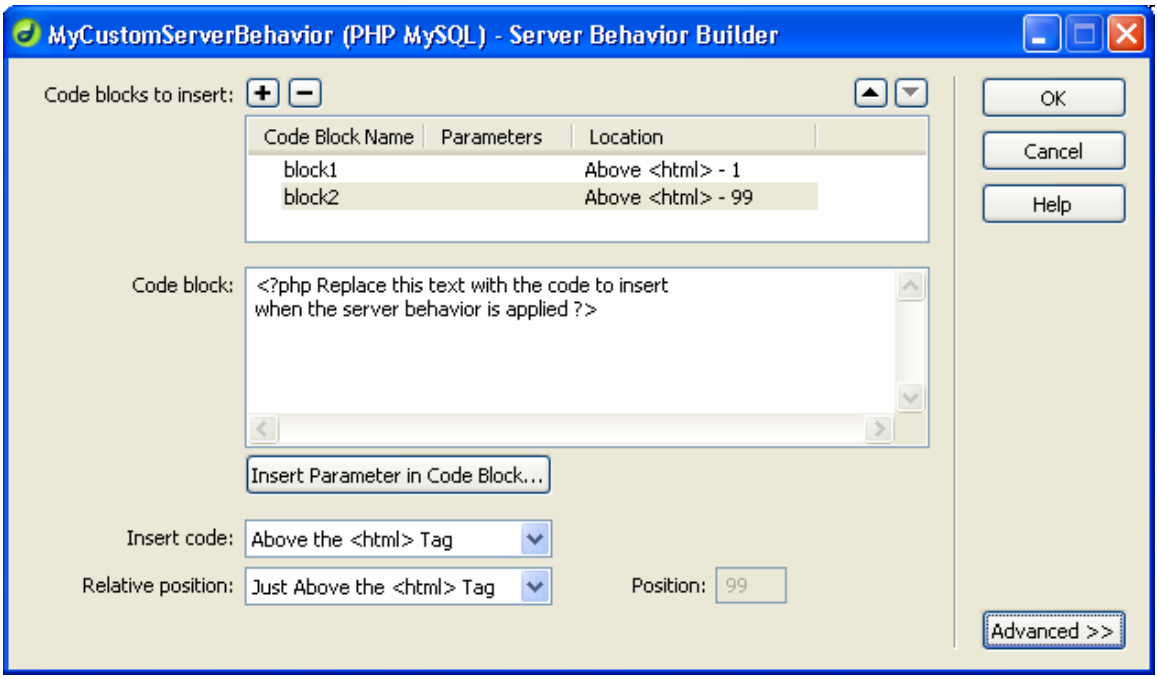

Figure 11: Custom Server Behavior Builder in DW

Security

While EUP tools empower less technically sophisticated users to autonomously create applications, unfortunately they may also empower end users to create security risks [12]. Security can be considered on many different levels including the physical access, operating system, server software, and application software levels. We focus on two particular types of security located at the server and application software levels: visible security, the security provided through login screens and authorization rules, and invisible security, the security built into the code to make applications less vulnerable to attacks that use loopholes in standards and implementations (e.g. SQL injection attacks).

Visible Security

Many of the tools that we reviewed have a predefined model for abstracting access rules and helping end users implement them. For example, DW features a server behavior called "Log in user." Prior to using this component, the developer must manually create a login web page that contains two form fields, one for the user ID and one for the password. The "Log in user" configuration dialog then asks the developer to identify the database connection, table, and fields that should be used to validate user input. In the case of DW and CCS, the developer must manually set up the database structure to contain user ID, password, and access-level information. In comparison, VWD/ASP.NET 2.0 works on a higher level of abstraction by offering a predefined Login control ("control" is ASP.NET terminology for "component") that already contains two text fields for the user ID and password. Furthermore, VWD stores the security information in an encoded form in a separate XML file rather than in a database table that the developer defines. VWD also advocates the use of the ASP.NET Web Site Administration Tool, which can be used to easily define access rules for each web page. Another powerful abstraction found in

VWD/ASP.NET 2.0 is the so-called LoginView control, a component that can render different content dependent on whether nobody is logged in (anonymous), a user is logged on, or a user belonging to a particular role or group is logged on. To employ this feature, the developer must define three templates that are rendered in place at runtime.

Another issue with visible security is input validation. Many types of attacks can be prevented by verifying that input provided by visitors has certain characteristics. The combination of VWD and ASP.NET, for example, offers a number of validation controls such as the RequiredFieldValidator or RangeValidator. A validation component defines an error message text and is bound to a particular input component and condition that defines valid input. One major advantage of this approach is flexibility; the error text can be placed anywhere on the page. Input validation not only improves the usability of a website, but also makes the code more resistant against attacks, a topic relating to invisible security. In VWD, if the developer wants an overview of the validation rules defined for a particular web page, he or she must click on all existing validation controls, which can be a tedious process. SW, on the other hand, displays all rules for all input components of a web page in tabular format on one screen (Figure 12).

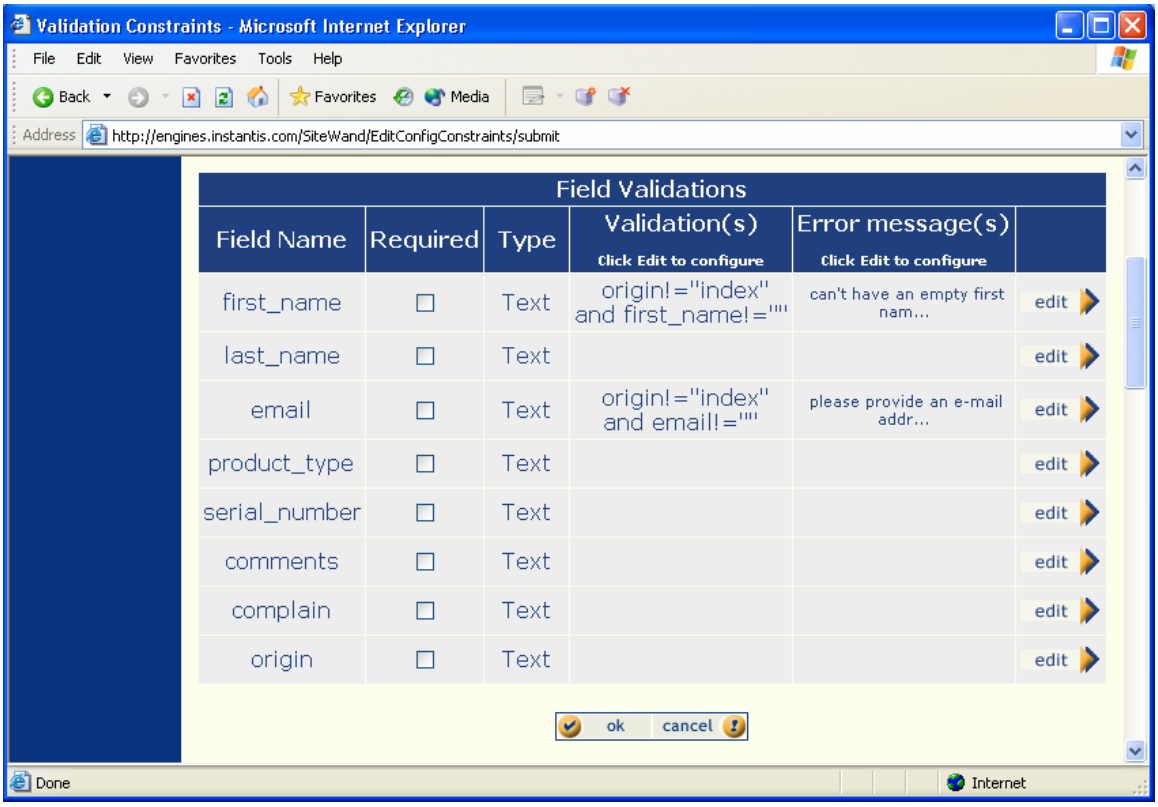

Figure 12: SW displays a screen summarizing all input validation rules and error messages

Invisible Security

Another aspect of security is the capability of an application to stand up against hacker attacks that abuse vulnerabilities within the web application's code. An example often found in the early days of eCommerce is faking the values of hidden HTML form fields to

gain access to private information, or to "adjust" the price of items for purchase. Although there are numerous techniques to avoid these vulnerabilities (e.g. using sessions instead of hidden form fields), nonprogrammers are unlikely to be aware of these and will expect their tools to create secure code. Unfortunately, under certain conditions code produced by DW is vulnerable to SQL injection techniques because the contents of HTTP GET variables are passed straight into a SQL query without being validated, escaped, or surrounded by quotes. Although we have not performed a security review of the code produced by each tool, it seems clear that tool creators should pay more attention to this issue because of end users' likely assumptions that the tools will be doing the "right thing."

Note that some tools already implement good security practices. RAD, for example, only passes pointers to information that has been securely stored within a session on the server, instead of passing database ID fields in clear text through HTTP GET requests. Using techniques like this, EUP web development tools can automatically provide security without requiring the end user to explicitly request it. A potentially important security feature that we did not find in any tool is an alert or information feature that informs the user of risks such as storing social security and credit card numbers or other private information.

Collaboration

Our scenario discusses the development of a web application by a single individual. Many situations in the realm of web development, however, call for collaboration among a number of people [26]. Collaboration can take a variety of forms from simple file sharing to simultaneous editing of the same file. Many of the tools in our review have built-in support for simple forms of collaboration. DW, for example, has support for a "check out" or file-locking feature. DW also has design notes that allow end users to document implementation rationale and decisions. SW provides a "secondary login", a feature that is slightly above the file-level approach of DW. The login allows a user to view the data stored by engines but not change the engines' configurations.

Higher level collaboration support (e.g. assigning roles, setting up review or acceptance workflows) was not a built-in feature in the tools in our review; we had to analyze the tools to determine how we would use them collaboratively. More advanced features similar to but not as "novice-hostile" as the popular CVS [7] would certainly be beneficial to a group of developers who would like to collaboratively construct an application, but they also have the potential to make the development process much more complicated.

Deployment

Deploying a web application has the potential to be a very difficult process for end users. Many of the tools in our review provide features to simplify the deployment process. Some tools such as FMP use a built-in server, which makes deploying applications relatively simple. The major issue with built-in servers is that they are not intended to handle large numbers of simultaneous users. Other tools like FA provide a publicly accessible test server. Upon completion of an application, end users can simply upload their application to a server that has been correctly configured. Another approach used by RAD and SW is to integrate the tool and the public server. Because these tools are web-based, the application is developed on a server and requires no deployment.

Discussion and Conclusions

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 VWD 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. Fraternali's and Paolini's observation about available web tools [9] 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 otherwise comparatively novice-friendly FP, 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.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 Mark, the informal web developer from our reference scenario (Figure 1, Figure 2). The tool that Mark is looking for needs to provide multiple reference examples, well-guided but short (unlike DRB) wizards, an integrated zero-configuration web server for testing purposes, and good support during the deployment phase of the application. Also, as Mark becomes more familiar with the capabilities of the tool and his applications become more ambitious, the tool should help him learn by exposing the inner workings of the wizards and forms.

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

Web development problem area	Recommended Solutions for Tools Targeted at End-users
Getting Started	• Avoid technical jargon for startup options (e.g. non-technical descriptions of underlying required technologies) • Provide wizards (with minimal premature commitment) • Provide example solutions • Provide templates
Workflow	• Take a holistic approach to web application development • Allow for gradual construction of the database
Level of Abstraction	• Provide high-level components such as data tables but also lower-level components for flexibility • Make components customizable, skinable
Layout	• Include the layout editor in the tool • Pixel-based editors are simpler than HTML-flow based editors • Provide templates and themes that can be applied site-wide
Database	• Allow for the creation of a new database including schema from within the tool or through a connection to an existing external database • Facilitate late changes to database schemas • Support populating and editing the database from within the tool • Provide a visual or form-based query builder
Application Logic	• Make session management transparent • Provide predefined high-level actions such as add, update, delete record, go to page, and send email • Offer wizards to create commonly used design patterns such as overview-detail or repeating regions
Testing and Debugging	• Facilitate fast iteration between building and testing, e.g. by using design-at-runtime • Avoid syntax errors by constraining the development UI • Provide context-sensitive error messages
Learning and Scaling	• Allow for viewing code parallel to design (e.g. DW split view) • Allow for viewing code by component (e.g. RAD) • Allow developer to edit automatically generated code or provide hooks or placeholders for custom code • Reintegrate custom modifications made by the end user into the automatically generated code (challenging research issue)

	• Document automatically generated code
Security	• Provide predefined user/permissions management and high-level security components (e.g. VWD Login control) • Provide high-level validation features for input components • Generate secure code (e.g. check inputs, SQL commands)
Collaboration	• Facilitate collaborative development by offering a file check-out or versioning system • Implement levels of access (e.g. develop, modify data, etc.)
Deployment	• Provide a built-in zero configuration test server, whether as a local server (e.g. VWD) or a remote server (e.g. FA) • Provide a built-in production server (e.g. FMP) or easy to use deployment wizard

Table 3: Guidelines for tools targeted at nonprogrammers derived from our tools review

We have argued that ease of use is currently an important problem for web application development tools intended for use by nonprogrammer end users and briefly discussed features that would improve ease of use. There are two major challenges, however, involved in designing a tool that includes these features: maintaining flexibility while limiting the amount of necessary up-front technical knowledge and recognizing and integrating user-generated and user-modified code segments into the overall workflow. Addressing these challenges will result in better tools.

The existing tools represent a good start, but they still leave much to be desired for supporting end-user development. Conceptually, web application development is not that complex of a process; tools should further reduce the complexity and not contribute to it.

Future Work

Although we are confident that the conclusions presented here are thought-provoking for tool designers, we are well aware of the limitations imposed by our informal evaluation methodology and therefore recommend more rigorous investigations of current tool designs. Observing tool use via case studies is an obvious choice for gathering results with ecological validity, although the data obtained in this fashion will necessarily be imprecise. Scaled usability sessions focused on particular tools or approaches would provide a complementary perspective and could be used to compare the efficacy of competing tool designs and services.

In our own research lab, we are in the process of building a prototype nonprogrammer web development tool that joins the best-of-breed approaches under a consistent user interface [23]. Through evolutionary prototyping and participatory design, it may also become an instrument for the elicitation of ever-changing user requirements.

References

[1] Ambler, A. and J. Leopold (1998). Public programming in a web world. *Visual Languages*, Nova Scotia, Canada.

[2] Apple Computer Inc. (1987). *HyperCard User's Guide*.

[3] 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.

[4] Carroll, J. M. and M. B. Rosson (1987). The paradox of the active user. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, ed. J. M. Carroll, p. 80-111, MIT Press, Cambridge, MA.

[5] Ceri, S., P. Fraternali and A. Bongio (2000). "Web Modeling Language (WebML): A modeling language for designing web sites." *Computer Networks*, 33(1-6): 137-157.

[6] Cooper, A. (1999). *The Inmates are Running the Asylum*, Indianapolis, IN, SAMS.

[7] CVS (2004). Concurrent Versions System. http://www.gnu.org/software/cvs/

[8] Fraternali, P. (1999). Tools and approaches for developing data-intensive web applications: A survey. *ACM Computing Surveys*, 31(3): 227-263.

[9] Fraternali, P., and P. Paolini (2000). Model-driven development of web applications: The Autoweb system. *ACM Transactions on Information Systems*, 28(4): 323–382.

[10] Gaedke, M., D. Schempf, 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.

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

[12] Harrison, W. (2004). The dangers of end-user programming. *IEEE Software*, 21: 5-7.

[13] Helman, T. and K. Fertalj (2003). A critique of web application generators. *Information Technology Interfaces (ITI)*, Cavtat, Croatia.

[14] MacLean, A., K. Carter, L. Lövstrand and T. Moran (1990). User-tailorable systems: Pressing issues with buttons. *Proceedings of CHI 1990*, 175-182.

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

[16] Newman, M., J. J.I. Lin, J. Hong and J. A. Landay (2003). DENIM: An informal web site design tool inspired by observations of practice. *Human-Computer Interaction*, 18: 259-324.

[17] Nielsen, J. (1992). Finding usability problems through heuristic evaluation. *Proceeding of CHI 1992*, 373-378.

[18] Norman, A. D. (1990). *The Design of Everyday Things*. New York, Double Day.

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

[20] Rode, J. and J. R. Howarth (2004). Usability evaluation of phpClick. http://phpclick.sourceforge.net/

[21] Rode, J. and M. B. Rosson (2003). Programming at runtime: Requirements and paradigms for nonprogrammer web application development. *IEEE HCC 2003*, Auckland, New Zealand.

[22] Rode, J., M. A. Pérez-Quiñones and M. B. Rosson (2004). The challenges of web engineering and requirements for better tool support. Submitted to *International Journal on Software Tools for Technology Transfer*.

[23] Rode, J., Y. Bhardwaj, M. B. Rosson, M. A. Pérez Quiñones, J. Howarth (2004). CLICK: Component-based Lightweight Internet-application Construction Kit. http://phpclick.sourceforge.net/

[24] Rosson, M. B. and J. M. Carroll (1996). The reuse of uses in Smalltalk programming. *Transactions on Human Computer Interaction*, 3(3): 219-253.

[25] Rosson, M. B. and J. M. Carroll (2002). *Usability Engineering: Scenario-based Development of Human-Computer Interaction*, San Francisco, CA: Morgan Kaufmann Publishers.

[26] Rosson, M.B., J. Ballin and H. Nash (2004). Everyday programming: Challenges and opportunities for informal web development. *IEEE HCC 2004*, Rome, Italy. Oct. 26-29.

[27] Stiemerling, O., H. Kahler and V. Wulf (1997). How to make software softer: Designing tailorable applications. *Symposium on Designing Interactive Systems*, 365-376.

[28] Tanimoto, S. (1990). VIVA: A visual language for image processing. *Journal of Visual Languages and Computing*, 1(2): 127-139.

[29] Turau, V. (2002). A Framework for automatic generation of web-based data entry applications based on XML. *17th Symposium on Applied Computing*, Madrid, Spain.

[30] Vora, P. R. (1998). Designing for the Web: A survey. *ACM interactions*, (May/June): 13-30.

[31] Wolber, D., Y. Su and Y. T. Chiang (2002). Designing dynamic web pages and persistence in the WYSIWYG interface. *IUI 2002*, San Francisco, CA.

[32] Zdun, U. (2002). Dynamically generating web application fragments from page templates. *17th Symposium on Applied Computing*, Madrid, Spain.