

Principles for the Design of Web Applications

Anthony I. Wasserman

Center for Open Source Investigation (COSI)
Carnegie Mellon West
Moffett Field, CA 94035 USA
tonyw@west.cmu.edu

Abstract. The development of high-volume web applications draws on many principles and guidelines drawn from software engineering, human-computer interaction, and other aspects of system design. This paper identifies those principles and guidelines, with a focus on abstraction and modularity principles, using a content management system as an illustration of using high-level abstractions to create web applications.

1 Building Web Applications

The development of a successful web application (website) involves many different kinds of design, including functional design, software architecture, business process or workflow design, user interface design, and database design. More than any other type of application, web applications require effective integrated use of software engineering practices and interaction design principles.

In the decade since the first web applications were created, mechanisms and tools for the design and development of such websites have evolved very substantially. While early websites might have had a handful of server-side actions, today's sites routinely include JavaScript (ECMAScript) for client-side interactivity and checking, dynamically generated web pages, and a complex "back-end" that often includes extensive processing logic written in Java or a scripting language (Perl, PHP, or Python), distributed processing functionality in a J2EE container, a database management system, and specialized components for security, e-commerce, and management of web services.

Applications developed for high-volume use on the World Wide Web are highly complex, and illustrate the practical application of principles and guidelines for human-computer interaction. The most heavily used websites are characterized by high reliability, high availability, high security, and rapid interactive response. For web applications such as Google, Amazon.com, eBay, and Yahoo, these qualities are essential, and are a key reason why they are trusted by millions of users. Other web applications, such as those used for major sporting events and video streaming, are designed to support spikes in traffic.

2 Anthony I. Wasserman

One result of the vast experience in website development is that many aspects of this development have become systematized, allowing the development team of a new site to follow a well-understood set of principles. Many of today's tools for web application design incorporate some of these principles, effectively encoding the knowledge that has come from the collective experience of building so many sites. For example, it's extremely easy to create a robust e-commerce site *without writing any code at all*. The builder of such a site chooses a template, customizes some graphical elements and menu items, and defines the inventory with pricing. After a few additional steps, the application is up and running on the server site. (See, for example, CityMax, at <http://www.CityMax.com>)

One could make the case that the "coding free" version of a web application is very constraining, both in terms of the styles of user interaction and the site content. While that's true, it also should be noted that the ability to enable the "instant" creation of a site that meets the needs of at least 90% of all users represents a formidable achievement in software engineering and human-computer interaction. The design knowledge for building e-commerce web sites has been captured and embedded in the application builder and its server environment. This knowledge includes site and page templates for presentation of the site content, the functionality needed by an e-commerce site, and the infrastructure of the application as needed to assure acceptable performance, reliability, and security.

One might say that the creators of such an application builder have "boiled it down to a science," incorporating principles of interaction design (forms, navigation mechanisms, page layout), site architecture (server side processing, data management), and key application functionality (e.g., shopping carts, integration with payment mechanisms, security certificates, site registrations for express checkout).

These developers also know the traffic limits of sites created in this way. Dantzig's history of IBM's experience in building high volume web sites [1] describes their growing sophistication in site design over a five year period. Yen, Hu, and Wang have described a framework for effective web site design [2]. Menascé and Almeida [3] have defined a four level reference model that links together e-business requirements with capacity planning. Killilea [4] has focused on server configurations in his coverage of performance issues. Barish [5] has focused on scalability and performance issues for the Java Enterprise platform. (Other good sources of information on performance aspects of web application design are the annual meetings of the International Workshop on Web Site Evolution and the International Conference on Web Site Engineering.)

There is also a substantial body of literature of the human factors aspects and visual design of web applications. Van Duyne and colleagues [6] have identified important guidelines for site design. Nielsen has written a widely followed book on usability [7]. By contrast, though, there has been relatively little written about the software engineering design principles and scientific foundations that underlie this kind of software. Rosenfeld and Morville [8] have addressed both the aesthetics and the mechanics of web site design, but there are few other works that do so.

In summary, there is a significant challenge to unify design principles from the various disciplines that influence the development of web applications and their subsequent usefulness. This work cuts across all aspects of software development from requirements analysis through implementation (coding and deployment architecture) and testing, as well as across numerous aspects of human-computer interaction, including task flow analysis, user interface design, and web site responsiveness. If one views software engineering as the application of underlying scientific formulas, laws, and principles, then the longer term goal of this work is to create this scientific foundation that provides a solid basis for the systematic development of usable web applications, as well as expanded automated support for the engineering processes. The remainder of this paper identifies some of the unifying themes across software engineering and human-computer interaction as they apply to web application development as a starting point for creating an integrated design framework.

2 Software Design Principles

There have been many previous efforts, including that of the author [9, for example], to identify the essential qualities of software design. More recently, the US National Science Foundation has sponsored a workshop and funded research projects on the theme of a Science of Design. The summary of the workshop, held in November, 2003, cited the need for a science of design, and enumerated goals for such an effort, including:

“To enable the design of software-intensive systems as dependable and fit to purpose as mechanical, civil, and other engineering systems typically are today; to develop rich behaviors at complex interfaces between software-intensive systems and their human and physical environments; to enable automated tools based on sound science to replace costly and error-prone human activities in software design; to leverage knowledge of design from other fields; to develop design methods for emerging technology platforms; to understand how to design software-intensive systems as social agents; to understand the characteristics of successful designs and how they are achieved; to learn how to represent designs at a much higher-level than code but nevertheless rigorously embody the key constraints; to discover how multi-disciplinary communities of designers and users can communicate to jointly advance design processes” [10]

Many of these goals depend on integration of design principles across domains of software engineering. Because web application development is software-intensive

and touches many different domains, it makes an excellent area for the exploration of these principles.

We now identify some basic design principles, drawing in part from [8], and showing how they fit in the context of web application development. We illustrate the application of some of these principles with examples from an open source content management system (CMS), Drupal (<http://www.drupal.org>), described by its creators as a “dynamic web site platform which allows an individual or community of users to publish, manage and organize a variety of content.” Drupal provides a site development environment with minimal coding, including weblogs, collaborative tools and discussion-based community software. The citizen journalist site, NowPublic (<http://www.nowpublic.com>), is an example of a site managed by Drupal.

2.1 Abstraction

Abstraction is a fundamental technique for problem understanding and solving. The psychological notion of abstraction permits one to concentrate on a problem at some level of generalization without regard to irrelevant low-level details. Abstraction is a common intellectual technique for managing the understanding of complex items, and is pervasive in software engineering.

In the requirements stage, methods and notations such as use case diagrams, scenarios, work flow models, and conceptual data models allow analysts to understand the problem independently of its implementation. At the design stage, user interface designers can think in terms of images, menus, buttons, text fields, and other visual design elements. Software designers can think in terms of major system functions and interfaces among them. Principles of information hiding, as described by Parnas [11], are a central notion of abstraction.

A CMS, such as Drupal, provides its users with an abstracted view of the basic functions of a web site, allowing the site administrator(s) to select components for use in a site managed by that CMS. Among the components provided in Drupal are user registration and login, static page creation, content aggregation (RSS), discussion forums, weblog (blog) creation, and site searching.

From a user interface perspective, the CMS provides visual design themes: templates that control the layout and format of the content. The site designer is presented with a collection of themes, which define the layout for a set of “blocks” on the page. The choice of theme determines the basic look-and-feel of the site, and can be used “as is”, or by adding some HTML code to add logos and other graphical elements within the constraints of the theme. One such theme is employed in the Drupal-based Spread Firefox site (<http://www.spreadfirefox.com>).

The important concept here, from the standpoint of design principles, is that the developer of the web application is able to work at a much higher level of abstraction than was previously possible. There is no need to write code to build a user

registration form, populate the registration database, or manage the subsequent user login process. All of these functions are bundled and automatically handled by the user registration and login component. Much the same is true for the other high level functions provided by the CMS. In the absence of a user interface designer, the developer can choose from a selection of site design layout themes, virtually all of which are superior to a design that could be created by someone without user interface design knowledge and experience.

2.2 Modularity and Architecture

Software architectures play a major role in determining the quality and maintainability of a system. The overall architecture of a software system has long been recognized as important to its quality (or lack thereof). An architecture's building blocks include units that carry out the system's behavior, connections that show how these units transmit or share information, and connections that show how the various program operations are activated, either sequentially or concurrently.

Web applications have a well-known and widely followed n-tier site architecture. Traditionally, the client side is a web browser that uses a network connection to a web server that directly processes requests for static HTML pages and passes along any requests that require additional server-side processing. The server side requests are often managed by an application server, which provides additional services for clustering, fail-over, and load balancing, in addition to managing connections to database management systems. Application servers are a central component of high volume content management systems, intranets, portals, and custom applications. With the appropriate architecture, these servers, as well as supporting database management systems, can be replicated and clustered to provide the desired levels of reliability and scalability.

In its simplest form, the site structure is 3-tiered, with the presentation layer on the client side, the business logic embodied in code processed by the application server, and the data managed by the database management system. This three-tier model is supported directly by Struts (<http://struts.apache.org>), which uses a variant on the model-view-controller (MVC) architectural design pattern.

Design patterns [12, 13] are yet another example of allowing the designer to work at a higher level of abstraction than was previously the case. With Struts, for example, the developer writes a configuration file (in XML) to organize the different actions of the architecture into user inputs (Action Forms), system activities (Action Mappings), and output page selection (Action Forwards).

Design patterns apply not only to functional and structural components, but also to elements of the user interface. Martijn van Welie has compiled a collection of patterns in interaction design, covering many common patterns of web site and GUI design, including navigation mechanisms, site structures, searching, menu structures,

and basic page types (<http://www.welie.com>). He has also written about mechanisms for organizing pattern languages [14].

Plug-ins are another important architectural contribution. Plug-ins were introduced as components of professional design and publishing tools, such as Adobe Photoshop and QuarkXPress (XTensions). The developers of these tools created extensible architectures that allow third parties to extend the functionality of the basic tool by creating new components that work within the tool's architectural framework. Plug-in architectures are now found in a broad variety of software components, including the Eclipse development platform and application frameworks (<http://www.eclipse.org>) and the Firefox browser (<http://www.mozilla.org>).

In summary, there is now a significant body of knowledge about the ideal structure of software systems and web applications, building upon first principles of abstraction and modularity. Knowledge and use of these successful structures is a key principle for creating new web applications.

2.3 Logging for Analysis and Testing

Traditional software packages can be instrumented internally or externally to trace their execution flow, invocation of other software components, and other performance measures. There are numerous tools for performance tuning, load testing, GUI testing, real-time analysis, and related measures. Many such tools are valuable in the development and testing of web applications. The typical application server, e.g., WebLogic Server or JBoss, produces detailed log files that are valuable for studying system performance, identifying errors, and determining general patterns of use.

Beyond that, though, a web application can be extensively tested and evaluated from the perspective of its users. An HTTP server, such as Apache, produces the Common Logfile Format, a standard log that shows details of every request made from a web browser accessing the application (site). Basic analysis tools, such as Analog (<http://www.analog.cx>) and Webalizer (<http://www.mrunix.net/webalizer>), provide information on traffic by browser, time of day, and originating domain, as well as counts of page viewings, errors, and searches.

While this information is useful for finding errors and scaling the hosting resources, it is possible and often more valuable to collect higher level information. For example, it is useful to understand such items as the amount of time that a user spends on the site, which pages have been viewed, the sequence of pages navigated by a user, the time taken to complete a task, login and registration information, and many more related items. Operators of e-commerce sites not only want to know which products were viewed, but also which items were placed into shopping carts and ultimately purchased; at the same time, they also want to know when shopping carts were abandoned. Advanced log analysis tools, such as 3DStats (<http://www.3dstats.com>) and WebTrends (<http://www.webtrends.com>) can provide this data.

Such information provides web application developers with rapid, even realtime, feedback on usage patterns for their site(s). For example, traffic data and user behavior can validate the success of a marketing campaign or the popularity of a new product on an e-commerce site. Amazon.com, for example, is able to use this data to maintain a dynamically updated list of their most popular products in various categories. Problems can be found and fixed.

The underlying principle is that multiple levels of logging data is essential to continuous improvement of the web application, from both the standpoint of application reliability, security, and efficiency, as well as the perspective of the user experience. Mechanisms for collecting and analyzing this data should be built into the ongoing operation of the site.

2.4 Drupal and the Design Principles

The Drupal CMS adheres quite well to these principles. Modularity and abstraction are well supported, since additional functionality and content can be added to a Drupal-based site without access to or knowledge of the underpinnings of the system (an HTTP server, PHP, and either the MySQL or PostgreSQL DBMS). The underlying site infrastructure is hidden, and the site administrator(s) can enable various functional components above and beyond those provided by default. In other words, the CMS has functional modules that implement abstractions such as content aggregation, blogging, and site searching.

Drupal's open architecture allows third-party development of add-on modules. Among the more than 100 such modules (see <http://drupal.org/project/Modules>) are those for display of ad banners and for e-commerce. These modules can be freely downloaded and integrated with other modules, sharing the same look-and-feel theme.

The site administrator has access to logs that show various system events, including user sessions and security violations. These logs afford a task-oriented view of user behavior at a higher level of abstraction than that provided by the logs created by the HTTP server and the DBMS.

While Drupal remains a work in progress, the functionality currently available through the base CMS and the add-on modules (akin to plug-ins) is such that one can build significant web applications with minimal coding. Drupal is an excellent example of a software system that is built on well-proven underpinnings and that follows good design principles on its own, therefore simplifying the task of creating new web applications.

Beyond Drupal, numerous other web applications, such as SugarCRM (<http://www.sugarcrm.com>), are following the same principles, creating an open source software application that builds on proven components, such as the Apache HTTP server and the MySQL RDBMS.

The installation of these applications invokes a script that creates the database schema needed by the application, populating it as needed. Both the Drupal CMS and the SugarCRM application are written in a scripting language, PHP (<http://www.php.net>) [15, 16] in these examples, but there are numerous other languages that could alternatively be used. Web pages containing PHP script have a php extension instead of an htm or html extension. Two lines are added to the configuration file for the Apache HTTP server to indicate that the PHP interpreter should be invoked for those pages with the php extension. (Similarly, Java Server Pages are denoted with a jsp extension.)

Another approach to web application development may be found with Ruby on Rails [17], a new framework that builds on the Ruby programming language [18]. The Rails framework builds upon a specialized web server and overcomes a major inefficiency of traditional web applications. The Basecamp project management system (<http://www.basecamp.com>) is built on this framework.

An important aspect of all of these tools is that they make effective reuse of major software components and also enable future tools to be built upon the foundation that they have established. The net result of reusing these mature components will be growing quality of the applications, less need to write new code, and reduced time for creating both prototypes and production versions of web applications.

3 Conclusion

Abstraction and modularity are key software engineering principles that are central to the design of web applications. Patterns for site architectures, software components, and interaction design provide a powerful foundation for the effective creation of usable and efficient web applications. Logging provides valuable information on both application performance and user behavior. When new web applications are built upon proven components that follow these principles, the development time for new applications is drastically reduced, while the quality of the application is increased. The domain of web application design shows numerous areas where user interface design interacts with these software engineering principles, making it an excellent area for further study of integrated methods and processes involving these domains, as well as a basis for a science of design.

References

1. Dantzig, P.: Architecture and Design of High Volume Web Sites. In: The Fourteenth International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July, 2002, pp. 17-24.
2. Yen, B., P. Hu, and M. Wang: Towards Effective Web Site Designs: A Framework for Modeling, Design Evaluation and Enhancement. Proceedings. IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005, pp. 716-721.
3. Menascé, D.A. and V.A.F Almeida: Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning. Prentice Hall, 2000.
4. Killelea, P.: Web Performance Tuning, 2e. O'Reilly, 2002
5. Barish, G.: Building Scalable and High-Performance Java Web Applications using J2EE Technology. Addison-Wesley, 2002.
6. Van Duyne, D.K., J. Landay, J.Hong: The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Addison-Wesley, 2002.
7. Nielsen, J. Designing Web Usability: the Practice of Simplicity. New Riders, 1999.
8. Rosenfeld, L. and P. Morville: Information Architecture for the World Wide Web: Designing Large Scale Web Sites, 2e. O'Reilly, 2002.
9. Wasserman, A.I.: Toward a Discipline of Software Engineering, IEEE Software, November, 1996, pp. 23-31
10. Sullivan, K.J.: Science of Design: Software-Intensive Systems -- A National Science Foundation Workshop. Airlie, Virginia, November, 2003. <http://www.cs.virginia.edu/~sullivan/sdsis/workshop%202003.htm>
11. Parnas, D.L.: On the Criteria to be used in Decomposing Systems into Modules, Communications of the ACM, 15, 12 (December, 1972), pp. 1053-1058.
12. Gamma, E., R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Object-Oriented Software Architecture. Addison-Wesley, 1995.
13. Alur, D., D. Malks, and J. Crupi. Core J2EE Patterns: Best Practices and Design Strategies, 2e. Prentice-Hall, 2003.
14. Van Welie, M. and G. van de Veer: Pattern Languages in Interaction Design: Structure and Organization. In: Human-Computer Interaction – INTERACT'03, Zürich, September , 2003, pp. 527-534.
15. Lerdorf, R. and K. Tatroe. Programming PHP. O'Reilly, 2002.
16. Williams, H.E. and D. Lane. Web Database Applications with PHP and MySQL, 2e. O'Reilly, 2004.
17. Thomas, D. and D. Heinemeier Hanson. Agile Web Development with Rails: a Pragmatic Guide. Pragmatic Bookshelf, 2005.
18. Thomas, D., C. Fowler, and A. Hunt. Programming Ruby: the Pragmatic Programmer's Guide, 2e. Pragmatic Bookshelf, 2004.