Quality Attributes of Web Software Applications

Jeff Offutt, *George Mason University*

Web applications have very high requirements for numerous quality attributes. This article discusses some of the technological challenges of building today's complex Web software applications, their unique quality requirements, and how to achieve them.

The World Wide Web was originally designed to present information to Web surfers using simple sites that consisted primarily of hyperlinked text documents. Modern Web applications run large-scale software applications for e-commerce, information distribution, entertainment, collaborative working, surveys, and numerous other activities. They run on distributed hardware platforms and heterogeneous computer systems. The software that powers Web applications is distributed, is implemented in multiple languages and styles, incorporates much reuse and third-party components, is built with cutting edge technologies, and must interface with users, other Web sites, and databases.

Although the word "heterogeneous" is often used for Web software, it applies in so many ways that the synonymous term "diverse" is more general and familiar, and probably more appropriate. The software components are often distributed geographically both during development and deployment (diverse distribution), and communicate in numerous distinct and sometimes novel ways (diverse communication). Web applications consist of diverse components including traditional and nontraditional software, interpreted scripting languages, plain HTML files, mixtures of HTML and programs, databases, graphical images, and complex user interfaces.

As such, engineering an effective Web site requires large teams of people with very diverse skills and backgrounds. These teams include programmers, graphics designers, usability engineers, information layout specialists, data communications and network experts and data base administrators. This diversity has led to the notion of *Web site engineering*.[1]

The tremendous reach of Web applications into all areas of communication and commerce makes this one of the largest and most important parts of the software industry. Yet a recent National Research Council study[2] found that the current base of science and technology is inadequate for building systems to control critical software infrastructure. The President's commission on critical infrastructure protection reached

0740-7459/02/$17.00 © 2002 IEEE

Instead of "sooner but worse," it is often advantageous to be "later and better."

this same conclusion in the President's Information Technology Advisory Committee report.[3] This inadequacy is particularly severe in the novel, high-risk area of Web application software. Although Web software development uses cutting-edge, diverse technologies, little is known about how to ensure quality attributes such as Web application reliability.

Unique aspects of Web application software

Several factors inherent to Web development contribute to the quality problem. Developers build Web-based software systems by integrating numerous diverse components from disparate sources, including custom-built special-purpose applications, customized off-the-shelf software components, and third-party products. In such an environment, systems designers choose from potentially numerous components, and they need information about the various components' suitability to make informed decisions about the software's required quality attributes.

Much of the new complexity found with Web-based applications also results from how the different software components are integrated. Not only is the source unavailable for most of the components, the executables might be hosted on computers at remote, even competing organizations. To ensure high quality for Web systems composed of very loosely coupled components, we need novel techniques to achieve and evaluate these components' connections.

Finally, Web-based software offers the significant advantage of allowing data to be transferred among completely different types of software components that reside and execute on different computers. However, using multiple programming languages and building complex business applications complicates the flows of data through the various Web software pieces. When combined with the requirements to keep data persistent through user sessions, persistent across sessions, and shared among sessions, the list of abilities unique to Web software begins to get very long.

Thus, software developers and managers working on Web software have encountered many new challenges. Although it is obvious that we struggle to keep up with the technology, less obvious is our difficulty in understanding just how Web software development is different, and how to adapt existing processes and procedures to this new type of software.

Economic changes

We evaluate software by measuring the quality of attributes such as reliability, usability, and maintainability, yet academics often fail to acknowledge that the basic economics behind software production has a strong impact on the development process. Although the field of software engineering has spent years developing processes and technologies to improve software quality attributes, most software companies have had little financial motivation to improve their software's quality. Software contractors receive payment regardless of the delivered software's quality and, in fact, are often given additional resources to correct problems of their own making. So-called "shrink wrap" vendors are driven almost entirely by time-to-market; it is often more lucrative to deliver poor-quality products sooner than high-quality products later. They can deliver bug fixes as new "releases" that are sold to generate more revenue for the company. For most application types, commercial developers have traditionally had little motivation to produce high-quality software.

Web-based software, however, raises new economic issues. When I recently surveyed a number of Web software development managers and practitioners, I found that companies that operate through the Web depend on customers using and, most importantly, returning to their sites. Thus, unlike many software contractors, Web application developers only see a return on their investment if their Web sites satisfy customers' needs. And unlike many software vendors, if a new company puts up a competitive site of higher quality, customers will almost immediately shift their business to the new site once they discover it. Thus, instead of "sooner but worse," it is often advantageous to be "later and better." Despite discussions of "sticky Web sites" and development of mechanisms to encourage users to return,[4] thus far the only mechanism that brings repeat users to Web sites has been high quality. This will likely remain true for the foreseeable future.

In software development, a *process driver* is a factor that strongly influences the

process used to develop the software. Thus, if software must have very high reliability, the development process must be adapted to ensure that the software works well. When I surveyed the important quality process drivers for traditional software, developers always gave a single answer that stands far above the rest: time-to-market. But when I recently made the same survey of Web software development managers and practitioners, they claim that time-to-market, although still important, is no longer the dominant process driver. They see the three most important quality criteria for Web application success (and thus, the underlying software) as

1. reliability,
2. usability, and
3. security.

Additional important criteria include

4. availability,
5. scalability,
6. maintainability, and
7. time-to-market.

Of course, this is hardly a complete list of important or even relevant quality attributes, but it provides a solid basis for discussion. Certainly speed of execution is also important, but network factors influence this more than software does, and other important quality attributes such as customer service, product quality, price, and delivery stem from human and organizational rather than software factors. That said, the quality attributes I just listed track closely with those cited in other books and articles,[1, 5-8] suggesting wide agreement that successful Web software development depends on satisfying these quality attributes.

Reliability

Extensive research literature and a collection of commercial tools have been devoted to testing, ensuring, assuring, and measuring software reliability. Safety-critical software applications such as telecommunications, aerospace, and medical devices demand highly reliable software, but although many researchers are reluctant to admit it, most software currently produced does not need to be highly reliable. I have been teaching software testing in various forms for 15 years yet have always felt like I was selling something that nobody wants.

Many businesses' commercial success depends on Web software, however—if the software does not work reliably, the businesses will not succeed. The user base for Web software is very large and expects Web applications to work as reliably as if they were going to the grocery store or calling to order from a catalog. Moreover, if a Web application does not work well, the users do not have to drive further to reach another store; they can simply point their browser to a different URL. Web sites that depend on unreliable software will lose customers, and the businesses could lose much money. Companies that want to do business over the Web must spend resources to ensure high reliability. Indeed, they cannot afford not to.

Companies that want to do business over the Web must spend resources to ensure high reliability.

Usability

Web application users have grown to expect easy Web transactions—as simple as buying a product at a store. Although much wisdom exists on how to develop usable software and Web sites (Jakob Nielsen's text[9] being a classic example), many Web sites still do not meet most customers' usability expectations. This, coupled with the fact that customers exhibit little site loyalty, means unusable Web sites will not be used—customers will switch to more usable Web sites as soon as they come online.

Security

We have all heard about Web sites being cracked and private customer information distributed or held for ransom. This is only one example of the many potential security flaws in Web software applications. When the Web functioned primarily to distribute online brochures, security breaches had relatively small consequences. Today, however, the breach of a company's Web site can cause significant revenue losses, large repair costs, legal consequences, and loss of credibility with customers. Web software applications must therefore handle customer data and other electronic information as securely as possible. Software security is one of the fastest growing research areas in computer science, but Web software developers currently face a huge shortfall in both available knowledge and skilled personnel.

Designing and building Web software applications that scale well represents one of today's most interesting and important software development challenges.

Availability

In our grandparents' time, if a shopkeeper in a small town wanted to take a lunch break, he would simply put a sign on the front door that said "back at 1:00." Although today's customers expect to be able to shop during lunchtime, we do not expect stores to be open after midnight or on holidays. But that only works for "brick-and-mortar" stores. When customers can visit our stores online, 2:00 a.m. in North America is the middle of the afternoon in Asia, and national or religious holidays fall on different days in different countries. On the Web, customers not only expect availability 24 hours a day, seven days a week, they expect the Web site to be operational every day of the year—"24/7/365." Even a 10-minute downtime can be damaging; I recently purchased $50 worth of books online but switched companies because the first Web site gave a "missing file" error message. Ten minutes later, that Web site was operational again but had lost my sale. Although this was only one sale, many customers would never come back.

Availability means more than just being up and running 24/7/365; the Web software must also be accessible to diverse browsers. In the seemingly never-ending browser wars of the past few years, some software vendors actively sought to make sure their software would not work under competitors' browsers. By using features only available for one browser or on one platform, Web software developers become "foot soldiers" in the browser wars, sometimes unwittingly. As an example, one major Web site at my organization uses "shockwave-flash," which is only compatible with MS IE and Netscape under Windows. Thus, the many Unix and Netscape users in my building cannot view their own Web site. To be available in this sense, Web sites must adapt their presentations to work with all browsers, which requires significantly more knowledge and effort on developers' part.

Scalability

A recent television advertisement showed a small group of young men and women nervously launching their Web site. The celebration started when the site got its first hit, but their faces quickly turned gray when the number of hits went into the thousands, then millions. As with a small corner store, as few as three or four people can create a commercial Web site but, unfortunately (or fortunately for the profit margin), virtually an unlimited number of customers can visit the Web site. We must therefore engineer Web software applications to be able to grow quickly in terms of both how many users they can service and how many services they can offer.

The need for scalability has driven many technology innovations of the past few years. The industry has developed new software languages, design strategies, and communication and data transfer protocols in large part to allow Web sites to grow as needed. Scalability also directly influences other attributes. Any programming teacher knows that any design will work for small classroom exercises, but large software applications require discipline and creativity. Likewise, as Web sites grow, small software weaknesses that had no initial noticeable effects can lead to failures (reliability problems), usability problems, and security breaches. Designing and building Web software applications that scale well represents one of today's most interesting and important software development challenges.

Maintainability

One novel aspect of Web-based software systems is the frequency of new releases, or the *update rate*. Traditional software involves marketing, sales, and shipping or even personal installation at customers' sites. Because this process is expensive, software manufacturers usually collect maintenance modifications over time and distribute them to customers simultaneously. For a software product released today, developers will start collecting a list of necessary changes. For a simple change, (say, changing a button's label), the modification might be made immediately. But the delay in releases means that customers won't get more complex (and likely important) modifications for months, perhaps years.

Web-based software, however, gives customers immediate access to maintenance updates—both small changes (such as changing the label on a button) and critical upgrades can be installed immediately. Instead of maintenance cycles of months or years, Web sites can have maintenance cycles of days or even hours. Although other

software applications have high maintenance requirements, and some research has focused on "on-the-fly" maintenance[10] for specialized applications, frequent maintenance has never before been necessary for such a quantity of commercial software.

Another ramification of the increased update rate has to do with compatibility. Users do not always upgrade their software; hence, software vendors must ensure compatibility between new and old versions. Companies can control the distribution of Web software to eliminate that need, though Web applications must still be able to run correctly on several Web browsers and multiple versions of each browser.

Another possible consequence of the rapid update rate is that developers may not feel the same need to fix faults before release—they can always be fixed later. I have seen no data to indicate this is happening.

Time-to-market

This has always been a key business driver and remains important for Web software, but it now shares the spotlight with other quality attributes. Most of the software industry continues to give priority to first to market. Given the other factors discussed here, however, the requirement for patience can and must impact the process and management of Web software projects.

Software researchers, practitioners, and educators have discussed these criteria for years, but no type of application has had to satisfy all of these quality attributes at the same time. Web software components are coupling more loosely than any previous software applications. In fact, these criteria have until recently been important to only a small fraction of the software industry. They are now essential to the bottom line of a large and fast growing part of the industry, but we do not yet have the knowledge to satisfy or measure these criteria for the new technologies used in Web software applications.

Technology changes

The commercial use of the Internet and Web has grown explosively in the past five years. In that time, the Internet has evolved from primarily being a communications medium (email, files, newsgroups, and chat rooms) to a vehicle for distributing information to a full-fledged market channel for e-commerce. Web sites that once simply displayed information for visitors have become interactive, highly functional systems that let many types of businesses interact with many types of users.

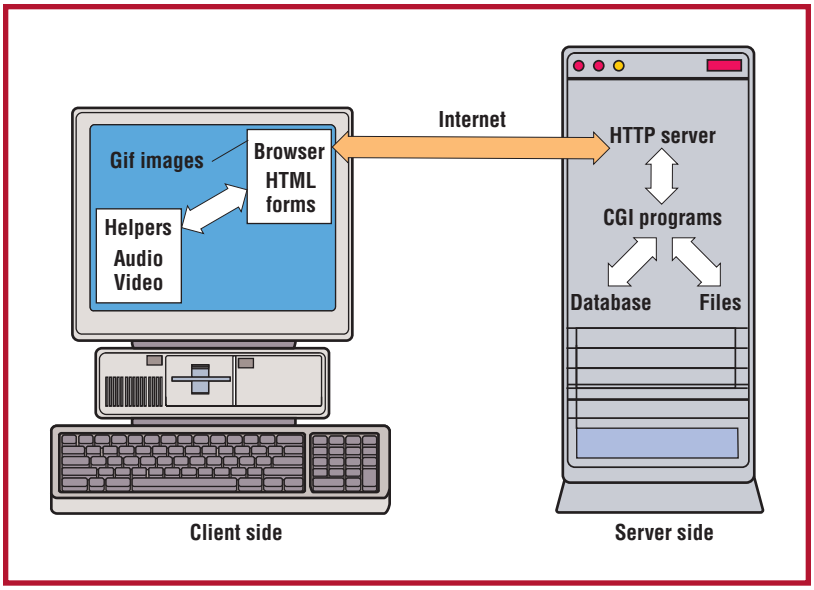

Figure 1. First-generation Web sites followed a client-server model that suffices to support simple page viewing and limited traffic but did not scale well.

These changes have had an enormous impact on software engineering. As the use of the Internet and Web has grown, the amount, type, and quality of software necessary for powering Web sites has also grown. Just a few years ago, Web sites were primarily composed of static HTML files, so-called "soft brochures," usually created by a single Webmaster who used HTML, JavaScript, and simple CGI scripts to present information and obtain data from visitors with forms.

Figure 1 illustrates the early Web, a typical client-server configuration in which the client is a Web browser that people use to visit Web sites that reside on different computers, the servers, and a software package called a Web server sends the HTML files to the client. HTML files contain JavaScripts, which are small pieces of code that are interpreted on the client. HTML forms generate data that are sent back to the server to be processed by CGI programs.

This very simple model of operation can support relatively small Web sites. It uses small-scale software, offers very little security, usually cannot support much traffic, and offers limited functionality. This was called a two-tier system because two separate computers were involved.

The Web's function and structure have changed drastically, particularly in the past 24 months, yet most software engineering researchers, educators, and practitioners have not yet grasped how fully this change affects engineering principles and processes. Web

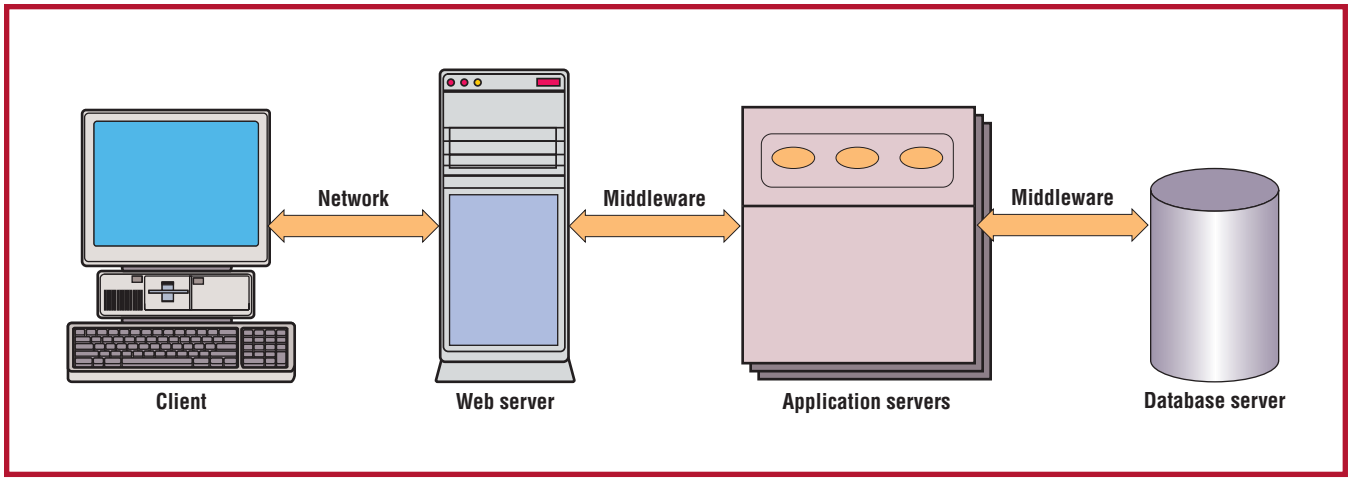

Figure 2. Modern Web sites generally follow an N-tier model that, by separating presentation from business logic, supports much greater application complexity, higher traffic, and stronger site security.

sites are now fully functional software systems that provide business-to-customer e-commerce, business-to-business e-commerce, and many services to many users. Instead of referring to *visitors* to Web sites, we now refer to *users*, implying much interaction. Instead of Webmasters, large Web sites must employ Web managers leading diverse teams of IT professionals that include programmers, database administrators, network administrators, usability engineers, graphics designers, security experts, marketers, and others. This team uses diverse technologies including several varieties of Java (Java, Servlets, Enterprise JavaBeans, applets, and Java Server Pages), HTML, JavaScript, XML, UML, and many others. The growing use of third-party software components and middleware represents one of the biggest changes.

The technology has changed because the old two-tier model did not support the high quality requirements of Web software applications. It fails on security, being prone to crackers who only need to go through one layer of security on a computer that is, by definition, open to the world to provide access to all data files. It fails on scalability and maintainability because as Web sites grow, a two-tier model cannot effectively separate presentation from business logic, and the applications thus become cumbersome and hard to modify. It fails on reliability: whereas previous Web software generations relied on CGI programs, usually written in Perl, many developers have found that large complex Perl programs can be hard to program correctly, understand, or modify. Finally, it fails on availability because hosting a site on one Web server imposes a bottleneck: any server problems will hinder user access to the Web site.

Figure 2 illustrates current Web site software. Instead of a simple client-server model, the configuration has expanded first to a three-tier model and now more generally to an "N-tier" model. Clients still use a browser to visit Web sites, which are hosted and delivered by Web servers. But to increase quality attributes such as security, reliability, availability, and scalability, as well as functionality, most of the software has been moved to a separate computer—the application server. Indeed, on large Web sites, a collection of application servers typically operates in parallel, and the application servers interact with one or more database servers that may run a commercial database.

The client-server interaction, as before, uses the Internet, but middleware—software that handles communication, data translation and process distribution—often connects the Web and application servers, and the application and database servers. New Web software languages such as Java are easier to modify and program correctly and permit more extensive reuse, features that enhance maintainability, reliability, and scalability. The N-tier model also permits additional security layers between potential crackers and the data and application business logic. The ability to separate presentation (typically on the Web server tier) from the business logic (on the application server tier) makes Web software easier to maintain and to expand in terms of customers serviced and services offered. Distributed computing, particularly for the application servers, allows the Web application to tolerate failures and handle more customers, and allows developers to simplify the software design.

Newer design models[11,12] have extended these goals even further. Java Server Pages (JSPs) and Enterprise JavaBeans (EJBs) let developers separate presentation from logic, which helps make software more maintainable. To further subdivide the work, developers can create a software dispatcher that accepts requests on the Web server tier, then

forwards the request to an appropriate hardware/software component on the application tier. Such design strategies lead to more reliable software and more scalable Web sites.

Of course the technology keeps changing, with the latest major addition to the technology being Microsoft's .NET. As of this writing, it is too early to say what type of affect .NET will have, although it does not seem to provide additional abilities beyond what is already available.

Clearly, modern Web sites' increased functionality creates a need for increasingly complex software, system integration and design strategies, and development processes. This leads to two exciting conclusions.

- One of the largest and fastest-growing software industry segments finds itself in dire need of the high-end software engineering practices and processes that researchers and educators have been developing and teaching for years.
- The new models for Web-based software production and deployment require that we adapt or replace many of the research solutions available now.

These conclusions imply that we need significant research progress, significant education, and significant training in diverse software engineering areas. The software that drives the Web has become a critical part of the world's infrastructure. Although Web software's immaturity poses significant risk to both industry and government, it also represents an opportunity for software engineering researchers and educators.

Planning for the future

One more important issue remains: the lack of engineers skilled in Web software development. A recent study from the US National Research Council found that the current base of science and technology is inadequate for building systems to control critical software infrastructure.[2] Much of Web software is built from existing systems and involves complicated analysis to effectively compose and integrate these components into systems. Yet the combination of a shortage of skilled IT engineers and the large number of IT jobs means companies often resort to hiring engineers who have less skills and education than desired. As a small example, US universities graduate approximately 25,000 bachelors in computer science every year, but industry recently estimated that it needs more than 200,000 IT professionals.[2]

Even with the current economic downturn, the university output is not enough. If universities could double the production of computer scientists, we still could not put a dent in the need. (Most economists and business leaders currently believe last year's many layoffs and bankruptcies in the e-commerce sector resulted from temporary problems, and expect significant growth in the near future. I optimistically accept this prognosis; if it is wrong, then this article will be irrelevant anyway.) We can only meet this need by

- retraining experienced engineers to work with the new technology
- applying knowledge and technology to increase efficiency, thereby reducing the number of engineers needed and
- finding ways to let people with less education and skills contribute

We are already seeing some progress in all three of these directions:

- Training classes and university courses in Web software engineering technologies are increasing and experiencing very high enrollment. The Web software engineering courses at George Mason University are over-subscribed every semester, with some students being non-degree professionals seeking to improve their marketability.
- New textbooks, tools, languages, and standards are emerging to make Web software engineering knowledge more accessible and easier to learn, use, and deploy. For example, several XML innovations in the past year have made it a more useful and accessible language, while new development tools and refinements in the standards for JSPs and EJBs allow more software to be created with less effort.
- Automated technologies have recently allowed nonprogrammers to contribute more to Web software development. When HTML was first introduced, an HTML writer needed to fully know the language and be proficient with a text editor to create Web pages. Recent tools provide point-and-click ability to create

Although Web software's immaturity poses significant risk to both industry and government, it also represents an opportunity for software engineering researchers and educators.

Web pages that can even be enhanced with dynamic HTML and JavaScripts while requiring very little knowledge of HTML and programming.

Achieving the high quality requirements of Web software represents a difficult challenge. Although other segments of the software industry have already mastered some of these, such as the need for reliability in telecommunications and network routing, aerospace, and medical devices, they have typically done so by hiring the very best developers on the market, using lots of resources (time, developers, and testing), or relying on old, stable software and technologies.

Unfortunately, these solutions will not work for Web applications. There are simply not enough of the "best developers" to implement all of the Web software needed today, and few but the largest companies can afford to invest extra resources in the form of time, developers, and testing. Finally, it should be obvious that old, stable software technologies will not suffice as a base for the Web—it relies on the latest cutting-edge software technology. Although the use of new technology involves some risk, it allows us to achieve otherwise unattainable levels of scalability and maintainability.

Indeed, technological innovations in just the past three years have greatly advanced the field, both in breadth and in depth. And research progresses: New conferences appear almost monthly, and traditional conferences feature more tracks and papers on Web software issues. Every new PhD student, it seems, wants to write a thesis on some aspect of Web software engineering, and more textbooks and classes teach Web software application material. When George Mason University offered a graduate course in Web software engineering in Fall 2000, the class immediately became popular, with a large waiting list.

Web software engineering presents challenging and unique research problems. We currently lack the knowledge to create Web software of sufficient complexity or quality and that can be updated quickly and reliably. Although Web software engineering has significant differences from traditional software engineering, we can adapt much of what we already know to understanding and resolving these differences. Not only are we making extraordinary progress, we are also bringing much research of the past 20 years to fruition. Indeed, this is an exciting time to be a software engineer.

Acknowledgments

This work is supported in part by the US National Science Foundation under grant CCR-98-04111.

References

1. T. A. Powell, *Web Site Engineering: Beyond Web Page Design*, Prentice Hall, Upper Saddle River, N.J., 2000.
2. F. B. Schneider, *Trust in Cyberspace*, National Academy Press, Washington, D.C., 1999.
3. President's Information Technology Advisory Committee, *Information Technology Research: Investing in our Future*, Technical Report, National Coordination Office for Computing, Information, and Communications, Washington, D.C., 1999; www.ccic.gov/ac/report.
4. D. A. Menascé, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*, Prentice Hall, Upper Saddle River, N.J., 2000.
5. E. Dustin, J. Rashka, and D. McDiarmid, *Quality Web Systems: Performance, Security, and Usability*, Addison-Wesley, Boston, 2001.
6. L. L. Constantine and L.A.D. Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design*, ACM Press, New York, 2000.
7. S. Murugesan and Y. Deshpande, "Web Engineering: A New Discipline for Development of Web-Based Systems," *Web Engineering 2000*, Lecture Notes in Computer Science 2016, Springer-Verlag, Berlin, 2001, pp. 3–13.
8. N. Kassem and the Enterprise Team, *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*, Sun Microsystems, Palo Alto, Calif., 2000.
9. J. Nielsen, *Designing Web Usability*, New Riders Publishing, Indianapolis, Ind., 2000.
10. M .E. Segal and O. Frieder, "On-the-Fly Program Modification: Systems for Dynamic Updating," *IEEE Software*, vol. 10, no. 2, Mar. 1993, pp. 53–65.
11. Wrox Multi Team, *Professional Java Server Programming, J2EE edition*, Wrox Press, Chicago, 2000.
12. A. Scharl, *Evolutionary Web Development*, Springer-Verlag, Berlin, 2000.

About the Author

Jeff Offutt is an Associate Professor of Information and Software Engineering at George Mason University. His current research interests include software testing, analysis and testing of Web applications, object-oriented program analysis, module and integration testing, formal methods, and software maintenance. He received a PhD in Computer Science from the Georgia Institute of Technology. He served as program chair for ICECCS 2001 and is on the editorial boards for *IEEE Transactions on Software Engineering*, *Journal of Software Testing, Verification and Reliability*, and *Journal of Software and Systems Modeling*. He is a member of the ACM and IEEE Computer Society. Contact him at the Dept. of Information and Software Engineering, Software Engineering Research Lab, George Mason University, Fairfax, VA 22030-4444; ofut@ise.gmu.edu; www.ise.gmu.edu/faculty/ofut.