Quality Attributes of Web Software Applications *

Jeff Offutt
Department of Information and Software Engineering
Software Engineering Research Lab
George Mason University
Fairfax, VA 22030-4444 USA
www.ise.gmu.edu/faculty/ofut/
ofut@ise.gmu.edu

January 21, 2002

To appear in IEEE Software special issue on Software Engineering for Internet Software, March/April 2002.

Abstract

In only four or five years, the world wide web has changed from a static collection of HTML web pages to a dynamic engine that powers e-commerce, collaborative work, and distribution of information and entertainment. These exciting changes have been fueled by many changes in software technology, the software development process, and how software is deployed. Although the word "heterogeneous" is commonly used to describe web software, we might easily forget to notice in how many ways it can be applied. In fact, the synonymous term "diverse" is more general and familiar, and may be more appropriate. Web software applications use diverse types of hardware, they include a diverse collection of types of implementation languages (including traditional programs, HTML, interpreted scripts, and databases), they are composed of software written in diverse languages, and they are built by collections of people with very diverse sets of skills.

Although these changes in **how** web applications are built are interesting and fascinating, one of the most unique aspects of web software applications is in terms of the **needs** they must satisfy. Web applications have very high requirements for a number of **quality attributes**. Some of these quality attributes have been important in other (mostly relatively small) segments of the industry, but some of them are relatively new. This paper discusses some of the unique technological aspects of building web software applications, the unique requirements of quality attributes, and how they can be achieved.

Keywords: web software engineering, quality attributes, e-commerce, web applications

1 Introduction

Only a few short years ago, the World Wide Web (WWW) was used to make information available to web surfers. These simple web sites were composed of mostly textual documents interconnected through hyper links. Since that time, we have gone through a major sea-change. Today, we use the WWW to run large-scale software applications for e-commerce, information distribution, entertainment, collaborative

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

working, surveys, and numerous other activities. The web applications run on hardware platforms that are distributed and heterogeneous computer systems. Although the term "heterogeneous" is often used to describe web applications, this word is traditionally associated with hardware components, thus this paper uses the more general and familiar synonymous term "diverse" to emphasize that building web applications requires diversity in many different ways.

Web applications are powered by software that 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. These are all aspects of the software that involve diversity. The software components are often distributed geographically both during development and deployment (diverse distribution), and communicate in a number of distinct and sometimes novel ways (diverse communication). Web applications are made up of diverse components including traditional and non-traditional software, interpreted scripting languages, plain HTML files, mixtures of HTML and programs, databases, graphical images, and complex user interfaces. As such, web applications are now built by large teams of people with very diverse talents, skills, knowledge, and backgrounds, including 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" [8], which is to say: an effective web site has to be carefully engineered by teams of people.

The tremendous growth in the use of web software applications makes this one of the largest and most important parts of the software industry. It is also now widely accepted that the safety and reliability of the United States' computer software is at risk. A recent study from the National Research Council found that the current base of science and technology is inadequate for building systems to control critical software infrastructure [11]. This same conclusion was reached by the President's commission on critical infrastructure protection in the PITAC report [9]. This inadequacy is particularly severe in the novel, high-risk area of web application software. A number of cutting-edge, diverse technologies are being used for web software and there is still little known about how to ensure quality attributes such as reliability of web software applications.

Web-based software systems are built by integrating a number of diverse components from a variety of sources. Some of these components are built as special-purpose applications, some are pre-existing, off-the-shelf software components built in-house, and some are purchased from third party vendors. In this kind of environment, systems designers choose from a potentially large number of components and need information about the suitability of the various components to make informed decisions about the various required quality attributes of the software.

Much of the new complexity found with web-based applications is a result of the way the different types

of software components are integrated together. Not only is the source unavailable for most of the components, the executables may be hosted on computers at remote, and even competing organizations. Thus, web software applications are composed of *very* loosely coupled components. To ensure high quality for these systems, we need novel techniques to integrate and evaluate the resulting connections of the various components.

A significant advantage of web-based software is that it allows data to be transferred among completely different types of software components that reside and execute on different computers. When other programming languages are used and the business application software gets more involved, the flows of data through the various pieces of the web software get more complicated. When combined with the abilities to keep data persistent through user sessions, persistent across sessions, and shared among sessions, the list of the unique abilities of web software begins to get very long.

2 Unique Aspects of Web Application Software

Software developers and managers working on web software have found that there are many new and unique challenges to building this type of software. Although it is obvious that we struggle to keep up with the technology, what may be less obvious is that it is hard to understand how to adapt processes and procedures to the new type of software. One of the major ways that software for the web is different from traditional software is in the basic economics behind the production of software, which has a strong impact on the process.

2.1 Changes In the Economics

We evaluate software by measuring the quality of attributes such as reliability, usability and maintainability. Although the field of software engineering has spent years developing processes and technologies to improve software quality attributes, most software companies have had little motivation to improve the quality of their software. Software contractors can be paid regardless of the quality of the delivered software, 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. Bug fixes are often delivered as new "releases" and can be sold to generate more revenue for the company. For most types of applications, there has traditionally been very little motivation for producing high quality software.

For web-based software, however, the situation is quite different. A recent survey of web software development managers and practitioners shows that companies that operate through the web depend on customers using their sites, and most importantly, **returning** to their sites. Users will only give return business to sites

that satisfy their needs. Thus, unlike many software contractors, web application developers will 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 that is of higher quality, customers will almost immediately shift their business to the new site. Thus, instead of "sooner but worse", it is often advantageous to be "later than and better". Although the idea of "sticky web sites" has been discussed and mechanisms to encourage users to come back have been developed [4], thus far the only mechanism that has brought repeat users to web sites has been high quality. It seems likely that this will continue to be true for the foreseeable future.

In software development, a *process-driver* is a factor that has a strong influence on the process used to develop the software. Thus, if software is required to have very high reliability, the development process must be adapted to ensure that the software works well. When I have surveyed the important quality process-drivers for traditional software, developers always give a single answer that stands alone 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. Instead, three of the most important quality criteria for success of web applications (and thus, the underlying software), were given as:

1. Reliability
2. Usability
3. Security

An additional four important criteria are:

4. Availability
5. Scalability
6. Maintainability
7. Time-to-market

Of course, it would be foolish to claim that this is a complete list of important or even relevant quality attributes. Certainly speed of execution is also important, but this is influenced by the network more than the software. Also, quality attributes such as customer service, quality of products, price, and delivery are also important, however these are not related to the software and thus out of scope of this paper. Nevertheless, these quality attributes track closely with what is said in other books and articles [8, 2, 1, 5, 3]. Thus, there is wide agreement that satisfying quality attributes is essential to web software and these seven provide a solid basis for discussion.

1. Reliability: This is a very familiar quality attribute that is associated with an extensive research literature and a collection of commercial tools for testing, ensuring, assuring, and measuring the reliability of software. The necessity of highly reliable software is also familiar in software applications that are

safety critical, such as telecommunications, aerospace, and medical devices. Although many of us who are researchers in these areas are reluctant to admit it, the truth is that most of the software that is currently produced does **not** need to be highly reliable. The segments of the industry mentioned above are relatively small. As an educator, I have been teaching software testing in various forms for 15 years, yet have always felt like I was selling something that nobody wants.

Web software, however, is **critical** to the commercial success of many businesses and if the software does not work reliably, the businesses will not succeed. The user base for web software is very large and they come expecting the web applications to work as reliably if they are 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 simply have to point their browser to a different URL. Thus, if web software is unreliable, web sites that depend on the software will lose customers and the businesses may lose large sums of money. These factors combine to make reliability of web software crucial, and most importantly, companies can afford to spend resources to ensure high reliability. Indeed, they cannot afford **not** to!

2. Usability: As said above, most web software applications have a broad user base. These users have grown to expect software to be very simple to learn how to use, as simple as buying a product at a store. Although a lot of knowledge is available for how to develop usable software and web sites (an example is Nielsen's text [6]), many web sites still do not meet the usability requirements that most of us expect. This, coupled with the fact that customers exhibit little site loyalty, means that web sites that are not usable will not be used: customers will quickly switch to more usable web sites as soon as they are put online.

3. Security: We have all heard about the recent cases where web sites have been cracked into and private customer information distributed or held for ransom. This is only one example of the many potential security problems in web software applications. When the main use of the web was to distribute online brochures, the consequences of security breaches were relatively small. With the much broader uses today, however, company's web sites that are broken into face significant losses in revenue, large repair costs, legal consequences and can lose credibility with their customers. Thus, it is essential that web software applications handle customer data and other electronic information as securely as possible. One of the fastest growing research areas in computer science is that of software security, and web software developers are facing a huge shortfall both in terms of available knowledge and personnel who have the knowledge that is available.

4. Availability: In our grandfathers' 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 the lunch time, we understand when stores are closed after midnight, on holidays, and part of the weekends. But that only works for "brick-and-mortar" stores! When customers can visit our stores online, 2:00 AM in North America is the middle of the afternoon in Asia, and Thanksgiving holidays

fall on different days in different countries. On the web, customers not only expect availability "24/7", they expect the web site to be operational every day of the year – "24/7/365". Even a ten minute down-time can be damaging; I recently purchased $50 worth of books online, and switched companies because the first web side gave a "missing file" error message. Ten minutes later the first web site was operational again. Although this was only one sale, many customers would never come back.

Availability means more than just being up and running 24/7/365, availability also means that the web software must be available when accessed by diverse browsers. The seemingly never-ending browser wars of the past few years have meant that software vendors actively try to make sure that their software will **not** work under competitive browsers. By using features that are 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 means that the many users of Unix and Netscape 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 the part of the developers.

5. 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 and millions. This graphically illustrated problems with scalability – just like a small corner store, a commercial web site can be created by as few as three or four people. Unfortunately (or fortunately for the profit margin!), **unlike** a small corner store, a commercial web site can be visited by virtually an unlimited number of customers. Thus, web software applications must be prepared to grow quickly both in terms of number of users serviced and in terms of services offered.

The need for scalability has been a driver for much of the technology innovations of the past few years. The industry has developed new software languages, new design strategies, and new communication and data transfer protocols, in part to allow web sites to grow as needed. Scalability also directly influences other attributes. A truism that any programming teacher knows is that any design will work for small classroom exercises, but large software applications require discipline and creativity. As web sites grow, small weaknesses in the software that did not cause problems in operation can lead to failures (reliability problems), usability problems, and security breaches. Designing and building web software applications that can be easily scaled is currently one of the most interesting and important challenges in software design.

6. Maintainability: One novel aspect of web-based software systems is the frequency of new releases, or the *update rate*. Installing traditional software involves marketing, sales and shipping or even personal installation at customers' sites. Because this process is very expensive, large numbers of maintenance modi-

fications are usually collected over time and distributed to customers at the same time. If a software product is released today, the developers will start developing a list of necessary changes. If the first change is simple (say, changing the label on a button), the modification may be made immediately. But the delay in releases means that modification will not be available to the customers for months, if not years!

With web-based software, however, the update rate is much faster. Maintenance updates can be installed and immediately made available to customers through the web site. Thus, even small individual changes (such as changing the label on a button) can be installed immediately. One result of this is that instead of maintenance cycles of months or years, web sites can have maintenance cycles of days or even hours. Although previous software applications have had high maintenance requirements, and there has even been research into so called "on the fly" maintenance [12] for specialized applications, frequent and constant maintenance updates have never before been necessary for much the commercial software.

Another ramification of increased update rate has to do with compatibility. Users do not always upgrade their software, thus software vendors must ensure compatibility of new versions with old versions. Companies can control the distribution of web software to eliminate that sort of compatibility. Of course, this is replaced by browser compatibility; web applications must be able to run correctly on several web browsers, and multiple versions of each browser. One possibility of the lack of compatibility problem is that web software developers may not feel the same need to fix as many faults before release – they can always be fixed later and installed easily. However, I have seen no data to indicate this is happening.

7. Time-to-Market: Of course time-to-market has always been a key business driver, and is still important for web software. What is unusual is not that it is important, but that it shares the spotlight with other quality attributes. Indeed, being first to market is the most important goal for most of the software industry. The requirement for patience can impact the process and management of web software projects.

Summary: These criteria have been discussed by software researchers, practitioners and educators for many years, but never in exactly this way. There are four major differences with web software applications:

1. These criteria have seldom been important to any but a small fraction of the software industry, whereas now they are crucially important to the bottom line of a large and fast growing part of the industry.
2. No type of application has ever had such compelling reasons to satisfy all of these quality attributes at the same time.
3. We do not yet have the knowledge to satisfy or measure these criteria for the new technologies used in web software applications.
4. Web software components are more loosely coupled than previous types of software applications.

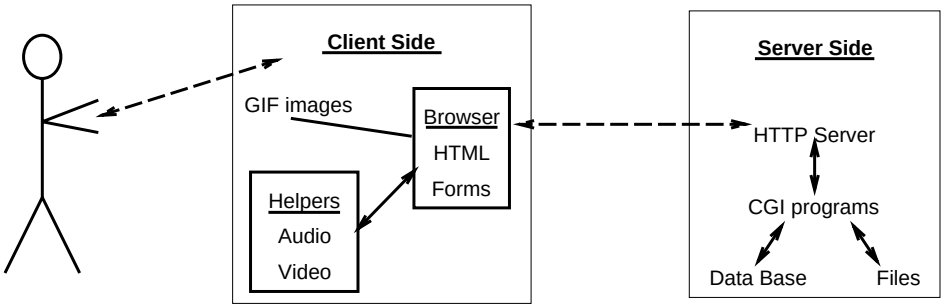

Figure 1: **First Generation Web Sites**

As said, there are other important quality attributes, including customer service, quality of products, price, and delivery. Except for where customer service is supported by usability, these attributes are not related to the software and thus out of scope of this paper.

2.2 Changes in Technology

The past five years has seen an explosive growth in the commercial use of the internet and the world wide web. In that time, we have moved from using the internet primarily for communication (email, files, newsgroups, and chatrooms), to using the world wide web as a vehicle for distributing information, and finally to a full-fledged market medium for e-commerce. At the same time, web sites have changed from primarily being mechanisms for displaying information for visitors to being interactive, highly functional systems that allow many types of businesses to interact with many types of users.

These changes are having 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." They were 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 a model for how the web was initially used. In many ways, this is a typical client-server configuration. The *client* is a web browser that people use to visit web sites. The web sites are on different computers, the *servers*, and the HTML files are sent to the client by a software package called a *web server*. 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 is a very simple model of operation that can support relatively small web sites. The software involved is by necessity small in scale, there is very little security, such web sites usually cannot support much traffic,

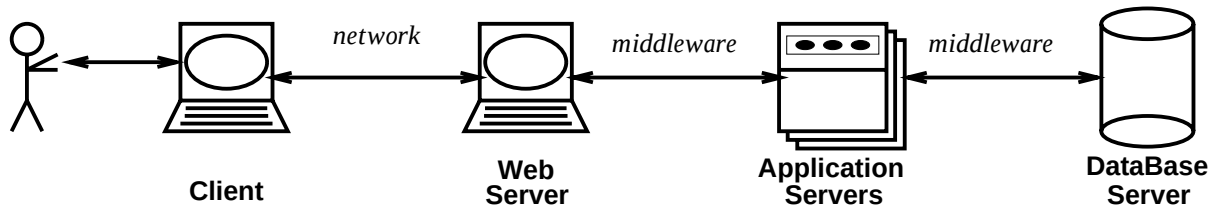

Figure 2: **Modern Web Sites**

and offer limited functionality. This was called a *2-tier system* because two separate computers were involved.

This situation has drastically changed. Moreover, most of the change has occurred in the past 24 months and is accelerating, and most software engineering researchers, educators and practitioners have not fully realized the extent of this change, particularly in terms of engineering principles and processes. Web sites are now fully functional software systems and provide business-to-customer e-commerce, business-to-business e-commerce, and a large variety of services to a large variety of users. Instead of referring to *visitors* to web sites, we refer to *users*, implying a large amount of interaction. Instead of *webmasters*, large web sites must employee *web managers* who lead diverse teams of IT professionals that include programmers, database administrators, network administrators, usability engineers, graphics designers, security experts, marketers, and others. This team uses a large variety of diverse technologies, including several varieties of Java (Java, Servlets, Enterprise JavaBeans, applets, and Java Server Pages), HTML, JavaScript, XML, UML, and many others. In addition, one of the biggest changes in technologies has been the growing use of software components and middleware from third-party vendors.

The situation has changed for good reason; the old 2-tier model of web software applications did not support all of the high requirements we have on our quality attributes. The 2-tier model is 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 have access to all data files (security). As web sites grow, it becomes very hard to separate presentation from business logic in a 2-tier model, and the applications thus become cumbersome and hard to modify (scalability and maintainability). Previous generations of web software relied on CGI programs, usually written in Perl. Many developers have found that Perl works best for small applications, and large complex Perl programs can often be hard to understand or modify (maintainability) and hard to program correctly (reliability). Finally, having one web server imposes a bottleneck, if there is a problem on the server then users cannot access the web site (availability).

Figure 2 illustrates the current model of web site software. Instead of a simple client-server model, the configuration has expanded first to a 3-tier model, and now more generally to an "*N-tier*" model. A client is still a browser used by a person 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, the application server is actually a **collection** of application servers that operate in parallel. The application servers typically interact with one or more *database servers*, often running a commercial data base.

The client-server interaction, as before, uses the internet. The web and application servers often are connected by *middleware*, which are packages provided by software vendors to handle communication, data translation and process distribution. Likewise, the application-database servers often interact through middleware.

The advent of new languages for web software applications (such as Java) has solved a number of problems. Such software is easier to modify and program correctly. It also allows for more extensive reuse, which in turn can enhance maintainability, reliability, and scalability. The N-tier model allows for additional layers of security between potential crackers and the data and application business logic. Having the ability to separate presentation (typically on the web server tier) from the business logic (on the application server tier) makes it easier to maintain web software and to expand the software both in terms of customers that can be serviced and services that can be offered. The use of distributed computing, particularly for the application servers, allows the web application to tolerate failures, handle more customers, and allows developers to simplify the software design.

Newer models of design [7, 10] have extended these goals even further. Java Server Pages (JSPs) and Enterprise Java Beans (EJBs) are used to separate presentation from logic, which helps make software more maintainable. One method for further subdividing the work is to have a software "dispatcher" that accepts requests on the web server tier, and then forwards the request to an appropriate software component on the application server. These types of design strategies can 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.

It is obvious that the increased functionality of modern web sites is resulting in a need for increasingly complex software, system integration and design strategies, and development processes. This leads to two exciting conclusions. First, **one of the largest and fastest-growing segments of the software industry is finding itself more and more in need of the high-end software engineering practices and processes that researchers and educators have been developing and teaching for a number of years.** Second, **the new models for how web-based software is being produced and deployed means that**

many of the research solutions we have available now either will have to be adapted or totally replaced to fit the new situation. Thus, we need significant research progress, significant education, and significant training in a diverse set of software engineering areas.

These facts, coupled with the speedy proliferation of web sites and use of the web by industry and government, mean that the software that drives them is a critical resource to the nation's infrastructure that is at great risk. This is a significant risk to both industry and government, but also an opportunity for software engineering researchers and educators.

3 Planning for the Future

There is one more issue that is relevant: the lack of engineers who are 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 [11]. Much of web software is built from existing systems, and the analysis needed to compose and integrate these components into systems is very complicated. On the other hand, the combination of an acute shortage of skilled IT engineers and the large number of IT jobs means companies will often need to resort to hiring engineers who have less skills and education than desired. As a small example, U.S. universities graduate approximately 25,000 bachelors in Computer Science every year. Yet industry recently estimated that it needs more than 200,000 IT professionals [11]. Even with the current economic downturn[1], the current university output is not enough. If universities could double the production of Computer Scientists, we still could not even put a dent in the need. This need must be met by a combination of three changes: (1) re-training experienced engineers to work with the new technology, (2) increased knowledge and technology to increase efficiency, partly by reducing the number of engineers that are needed, and (3) finding ways to allow people with less education and skills to contribute.

We are already seeing some progress in all three of these directions. For the first, training classes and university courses in web software engineering technologies are increasing and experiencing very high enrollment. As an example, the web software engineering courses at George Mason University are over-subscribed every semester, with a number of the students being non-degree professionals who are seeking to improve their marketability. For the second, we are continually seeing new textbooks, tools, languages, and standards that make web software engineering knowledge more accessible, easier to learn, use, and deploy. As an example, a number of innovations in XML 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

[1]Most economists and business leaders currently believe that the many layoffs and bankruptcies in the e-commerce sector in 2001 is a result of temporary problems, and will be relatively short-lived, with significant growth in the near future. I am optimistically accepting this prognosis; if it is wrong then this paper will be irrelevant anyway.

be created with less effort For the third, automated technologies have recently allowed non-programmers to contribute more to web software development. For example, 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 web pages that can even be enhanced with dynamic HTML and JavaScripts while requiring very little knowledge of HTML and programming.

Achieving the high requirements for quality attributes is quite a difficult challenge. Some of these requirements, for example the need for reliability, have been achieved by other segments of the software industry, particularly telcommunications and network routing, aerospace, and medical devices. High reliability has usually been achieved 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 the web software that is needed. Throwing extra resources in the form of time, developers and testing is not practical for small companies that cannot afford such a large investment. Finally, it should be obvious that old, stable software technologies cannot be used. Instead, the web relies on the latest cutting-edge software technology. Although, the use of new technology involves some risk, but it allows us to achieve levels of scalability and maintainability that could not be achieved otherwise.

4 Conclusions

To summarize the unique aspects of web software, modern web site applications require:

- software that is more complex than before
- software that is of high quality
- software that can be updated quickly and reliably

From the viewpoint of research, these issues present challenging problems. We do not currently have the knowledge to create software for the web that is both of sufficient **complexity** and of sufficient **quality**. On the other hand, the field is making extraordinary progress. The technological innovations that have been developed in just the past three years are stunning both in breadth and in depth. Research in this area is also exploding. New conferences are appearing almost monthly, and traditional conferences are featuring more and more tracks and papers on web software issues. From my own experience, it seems that every new PhD student wants to write a thesis on some aspect of web software engineering. We are also seeing more and more textbooks and classes that teach web software application material. As one small example, George Mason University began offering a graduate course in web software engineering in Fall 2000 and the class was immediately the most popular class, with a large waiting list.

Software engineering for the world wide web is indeed different, but we can adapt much of what we already know to understand these differences. The progress we are making is extraordinary, and is allowing much of the research of the last 20 years to be fruitfully used. Indeed, this is an exciting time to be a software engineer!

References

[1] Larry L. Constantine and Lucy A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design*. ACM Press, 2000.

[2] Elfriede Dustin, Jeff Rashka, and Douglas McDiarmid. *Quality Web Systems: Performance, Security, and Usability*. Addison-Wesley, 2001.

[3] Nicholas Kassem and the Enterprise Team. *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*. Sun Microsystems, 2000.

[4] Daniel A. Menascé. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall, 2000.

[5] S. Murugesan and Y. Deshpande. Web engineering: A new discipline for development of web-based systems. In *WebEngineering 2000*, pages 3–13, Berlin Heidelberg, Germany, 2001. Springer-Verlag Lecture Notes in Computer Science 2016, S. Murugesan and Y. Deshpande (Eds.).

[6] Jakob Nielsen. *Designing Web Usability*. New Riders Publishing, 2000.

[7] Patzer. *Professional Java Server Programming*. Wrox Press, J2EE edition, 2000.

[8] T. A. Powell. *Web Site Engineering: Beyond Web Page Design*. Prentice Hall, 2000.

[9] President's Information Technology Advisory Committee Report To The President. Information technology research: Investing in our future. Technical report, National Coordination Office Computing, Information, and Commun ications, February 1999. http://www.ccic.gov/ac/report/.

[10] Arno Scharl. *Evolutionary Web Development*. Springer, 2000.

[11] Fred B. Schneider. *Trust in Cyberspace*. National Academy Press, 1999.

[12] M. E. Segal and O Frieder. On-the-fly program modification: Systems for dynamic updating. *IEEE Software*, 10(2):53–65, March 1993.