Web Software Applications Quality Attributes

Jeff Offutt
Information & Software Engineering
George Mason University
Fairfax, VA 22030 USA
http://www.ise.gmu.edu/~ofut/

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 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 (Web) was used to make information available to Web surfers. Simple web sites were composed of text documents interconnected through hyper links. Since then the use of the Web has changed dramatically. We use the Web to run large-scale software applications to support e-commerce, entertainment, information distribution, surveys, and collaboration. Web software applications run on distributed and heterogeneous computer hardware platforms. Although the term "heterogeneous" is often used to describe web applications, this word is traditionally associated with hardware components. This paper uses the synonymous term "diverse" to emphasize that building web applications requires many different kids of diversity. Diverse is both more general and more familiar.

Web applications are powered by distributed software that is implemented in many languages and styles, incorporates reuse and third-party components, is developed with cutting edge technologies, and interacts with users, databases, and other web sites. These are all aspects of the software that involve diversity. Web software applications use software components that are 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 constructed 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. Web applications are now built by large teams of people with very diverse talents, skills, knowledge, and backgrounds, including programmers, usability engineers, data communications and network experts data base administrators, information layout specialists, and graphics designers. 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 production of web software applications has been the fastest growing segments of the software industry for several years. Web software use a number of cutting-edge, diverse technologies and we know little about how to measure or ensure quality attributes such as reliability.

One of the main reasons web software applications are so complex is because of the way the software components are integrated together. The source is not available for most components, and 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 transfers data 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 Web Software Technology

The past five years has extraordinary growth in the commercial use of the Internet and the Web. The Web was originally used for communication (email, files, newsgroups, and chatrooms), then as a vehicle for distributing information, and finally as a full-fledged market medium for e-commerce. During this period, web sites have changed from primarily being mechanisms for displaying information for visitors, so called "soft brochures," to being interactive, highly functional systems that allow many types of businesses to interact with many types of users.

These changes have significant ramifications for 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.

2.1   Web Software Terms

In this paper, a *web page* refers to data that fits in one browser window. The user may need to scroll to see the entire page, and the page may contain elements from multiple files on the server, but it is all loaded into the window in one chunk. Web pages can be single files on the server, multiple files that are composed statically, or created by a program. A *static web page* has HTML that exists as a file on the web server. A *dynamic web page* is created by a program as needed. Different users may see different dynamic web pages, and the same user may see a different dynamic web page from the same URL at different times.

A *web site* is a larger entity, it is a number of web pages that are semantically related and physically linked. That is, they have some common meaning and have hyperlinks between them. For example, all the web pages for one conference are semantically related and constitute a web site. Most web sites have one or more special web pages, an *entry point*, which is the first web page to be viewed in a web site. The entry point is sometimes called a landmark or a portal.

2.2   Changes in Web Software Technology

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.
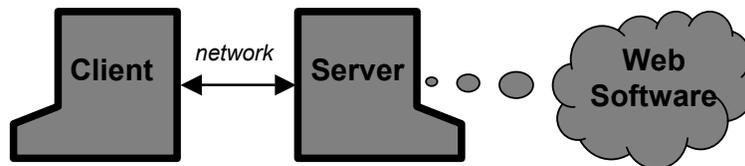

Figure 1: First Generation Web Sites

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

This situation has drastically changed, and the change is not only continuing, it is accelerating. The impacts these changes have on software engineering principles and processes is just now beginning to be understood. Web sites are now fully functional software systems. 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, and marketers. 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. 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).
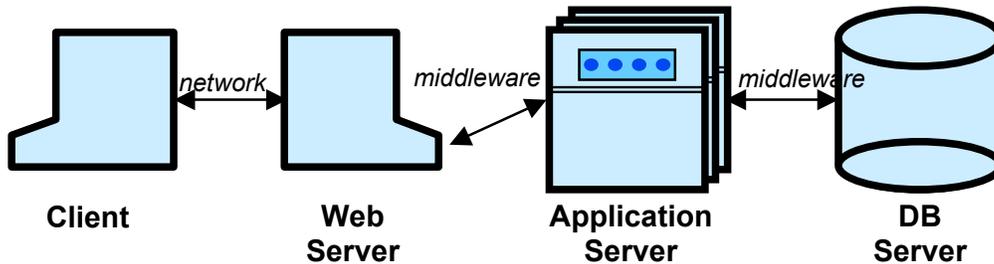

| Client | Web Server | Application Server | DB Server |

Figure 2: Modern Web Sites

Figure 2 illustrates the current model of web site software. Instead of a simple client-server model, the configuration 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 database.

The client-server interaction, as before, uses the Internet. 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.

New languages such as Java have 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. Separate the presentation (typically on the web server tier) from the business logic (on the application server tier) makes it easier to maintain and 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.

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.

3 Unique Aspects of Web Application Software

For reasons detailed below, web site software has to be better than most shrink-wrap or contract software. The combination of **higher quality requirements** and **unique** technologies make for a very

"interesting" situation. Of course, when academics use the word "interesting", we mean there are some fun challenges. But to software managers, the word "interesting" is usually scary.

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.

3.1   Changes In the Economics

Although researchers and practitioners have spent years developing processes and technologies to improve and measure software quality attributes, many software companies have had little motivation to improve the quality of their software. Software contractors are usually paid no matter how poor the quality of their delivered software is. Sometimes they can even get additional resources to correct problems of their own making. Commercial software vendors are driven almost entirely by time-to-market, which means they usually make more money by delivering 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.

However, the situation is quite different for web-based software. A recent survey of web software development managers and practitioners shows that web-based companies depend on customers using their sites, and most importantly, **returning** to their sites. Users will only return to sites that satisfy their needs. Thus, unlike many other applications web software applications only make money if their web sites satisfy customers' needs. Moreover, the time-to-market becomes much less important. Unlike many software vendors, if a new company releases competitive site that is of higher quality, customers will almost immediately start using the new site. Thus, instead of "sooner but worse", it is often advantageous to be "later-than and better". Although the use of "sticky web sites" has been discussed and ways to encourage users to come back have been developed [4], thus far the primary mechanism that has brought repeat users to web sites has been high quality. It seems unlikely that this will change anytime soon.

A *process-driver* can be considered to be a factor that has a strong influence on how software is developed. For example, 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: *efficiency*. Efficiency can be broken down into two separate elements, efficiency of process (*time-to-market*) and efficiency of execution (performance). This also mirrors traditional undergraduate CS objectives. But when I made the same survey of web software development managers and practitioners, they claimed 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 consistently given as:

1. Reliability

2. Usability

3. Security

An additional four important criteria are:

4. Availability

5. Scalability

6. Maintainability

7. Performance & Time-to-market

Of course, it would be foolish to claim that this is a complete list of important or even relevant quality

attributes. 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. If and when they do impact the software, they could be included as part of usability. Nevertheless, these quality attributes track closely with what is said in other books and articles [1,3,4,8]. 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: Highly reliable software has been necessary for applications that are safety critical, such as telecommunications, aerospace, and medical devices. Although many researchers are reluctant to admit it, the truth is that most software that is currently produced does **not** need to be highly reliable. The segments of the industry mentioned above are relatively small.

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. Web sites have vastly increased the customer base for software. For example, I have virtually no choice for which word processor to use, but there are dozens of online bookstores. Knowledgeable users will tolerate many more mistakes in the software than less knowledgeable users will. Many web sites also function in a very competitive market. More competition means that users will be able to make choices based on how well the software works. 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.

Perhaps more importantly, e-commerce web sites deal with very important items: Money and personal information such as credit cards, addresses, and buying habits. Web sites also offer transactions that are unrecoverable (buying merchandise) and have delayed results (shipped days or weeks later). These factors mean that customers expect web sites to work as well as grocery stores and catalogs -- because they replace those venues!

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! **Unfortunately, the new and diverse technologies mean that we know nothing about testing web site software.**

2. Usability: Traditional software applications have **users**, but web sites have **customers**. As said above, most web software applications have a broad customer base. These customers have grown to expect software to be as simple to learn as shopping 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.

Customers expect to be able to use web sites without training. Thus, the software must flow according to the users' expectations, offer only needed information, and when needed, and provide navigation controls that are clear and obvious. **Unfortunately, most software developers are not well educated in usability.**

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. **Security was once a math problem, then a networking problem, next a database problem, but now it is a software problem**.

4. Availability: In our grandfathers' time, shopkeepers in small towns would take their lunch breaks by simply putting 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 stores will be 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 Germany is evening in the U.S. and morning in Australia, and Christmas is just another day in China. Even a ten minute down-time can be damaging; customers will go to a different web site and may never return. **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".**

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 in the U.S. 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 commercial 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. The customers for both the corner store and the web site are limited by the size of the neighborhood, but **unlike** a small corner store, a commercial web site has a worldwide neighborhood and 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. It takes an engineer to develop web sites that can scale and still remain reliable, secure and available 24/7/365.

6. Maintainability: Web-based software applications have a higher frequency of new releases, or *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 modifications are usually collected over time and distributed to customers simultaneously. 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!

However, web-based software has a much faster update rate. 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.

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. **Although previous software applications have had high maintenance requirements, frequent and constant maintenance updates have never before been necessary for much the commercial software.**

7. <u>Performance & Time-to-Market</u>: If a web site is too slow, users will lose patience and go away. Nielsen [6] claims that users will lose concentration if they have to wait more than a few seconds and will leave your web site never to return after a delay of 30 seconds. Thus performance is important, but it is more important to note that performance is dominated by the internet traffic, often at the user's end. While software can impact the overall performance, it is really hard to write software that poorly.

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. Thus, the overall success for web site software depends on software engineering.

<u>Summary</u>: 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:

- Web software components are more loosely coupled than previous types of software applications.

- Web software components are created dynamically and even the flow of user choices through an application depends on the state of the system.

- 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.

- No type of application has ever had such compelling reasons to satisfy <u>all</u> of these quality attributes at the same time.

- We do not yet have the knowledge to satisfy or measure these criteria for the new technologies used in web software applications.

4 A Time of Transition

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 telecommunications and network routing, aerospace, and medical devices. Companies in these fields usually succeed 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 a practical investment for small companies. Finally, web applications are built with the latest cutting-edge software technology, not old, stable software technologies. Although, the use of new technology involves some risk, it allows us to achieve levels of scalability and maintainability that could not be achieved otherwise.

In 1995, web sites were 100% interface (that is, entirely HTML), in 1998 they were about 90% interface, but in 2002 web sites are less than 50% interface and the percentage is still shrinking. Thus it should be obvious that despite the current economic downturn, there is still a shortage of knowledgeable, skilled web programmers and software engineers.

Indeed, the ".com-.gone" crash of the last year has been painful for those who have lost jobs and took pay cuts. But a positive sign is that software engineering may finally be being validated: The companies that are still alive, doing well, growing, making money, and hiring are companies have "figured it out", and are applying sound software engineering principles and paying attention to quality attributes.

5 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

These issues present interesting research 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, and new research results are appearing every day.

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, J. Rashka, and D. McDiarmid. Quality Web Systems: Performance, Security, and Usability, Addison-Wesley, Boston, 2001.

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

[4] Daniel A. Menasce. 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," Web Engineering 2000, Lecture Notes in Computer Science 2016, Springer-Verlag, Berlin, 2001, pp. 3-13.

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

[7] J. Offutt. "Quality Attributes of Web Software Applications", IEEE Software: Special Issue on Software Engineering of Internet Software, 19(2):25-32, March/April 2002.

[8] T. A. Powell. Web Site Engineering: Beyond Web Page Design, Prentice Hall, Upper Saddle River, N.J., 2000.

[9] Scharl , Evolutionary Web Development, Springer-Verlag, Berlin, 2000.

6 *Author's Biography*

Jeff Offutt

Jeff Offutt is an associate professor in the Department of Information and Software Engineering, is a Research Scientist on a part-time basis with the National Institute of Standards and Technology's Information Technology Lab, Software Diagnostics and Conformance Testing Division, and holds a part-time visiting professor position at Skövde University in Sweden. His current research interests include program testing, object-oriented program analysis, module and integration testing, software architecture-based system testing, and formal methods. He has published over sixty refereed research papers and conferences and has received funding from various government agencies and companies. His current projects include NSF and NASA funded research to measure software maintenance of open-source software, analysis and testing of object-oriented software, testing of web applications, and deriving tests from formal specifications of safety critical software.

He teaches MS and PhD courses in Software Engineering and has developed new courses in a variety of Software Engineering subjects, including software testing, construction, design, user interface design, experimentation, and analysis.

Dr. Offutt received a BS degree with a double major in mathematics and data processing from Morehead State University, Morehead, Kentucky, in 1982, an MS degree in computer science from the Georgia Institute of Technology in 1985, and a PhD in computer science from the Georgia Institute of Technology in 1988. From 1988 to 1992, Offutt was an Assistant Professor in the department of Computer Science at Clemson University.

Offutt has served on numerous conference program committees, was program chair for ICECCS 2001, is on the editorial boards for the IEEE Transactions on Software Engineering, the Journal of Software Testing, Verification and Reliability, the Journal of Software and Systems Modeling, and the Software Quality Journal, is a regular reviewer for NSF and several major research journals, and has been invited to speak throughout the US, and Japan, China, and Sweden. He has been involved in a number of software proof-of-concept research systems, including Mothra, Godzilla, CBat, Mistix, Albert, CoupTest, and SpecTest, several of which have been used by many other software engineering researchers. Offutt previously worked on the Software Test and Evaluation Project, for Georgia Tech's Software Engineering Research Center, and helped design and implement the Mothra mutation testing system with Rich DeMillo. His doctoral research was a method for automatically generating test data to satisfy mutation analysis and included algorithms and an implementation of an automatic test data generator that is integrated with the Mothra system. He has also designed mutation operators for the Ada programming language.