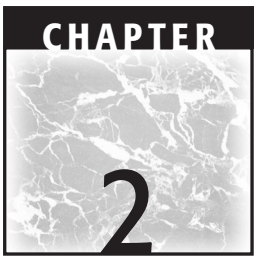

Web Testing versus Traditional Testing

Why Read This Chapter?

Web technologies require new testing and bug analysis methods. It is assumed that you have experience in testing applications in traditional environments; what you may lack, however, is the means to apply your experience to Web environments. To effectively make such a transition, you need to understand the technology and architecture differences between traditional testing and Web testing.

TOPICS COVERED IN THIS CHAPTER

- **Introduction**
- **The Application Model**
- **Hardware and Software Differences**
- **The Differences between Web and Traditional Client-Server Systems**
- **Web Systems**
- **Bug Inheritance**

(continued)

TOPICS COVERED IN THIS CHAPTER *(continued)*

- **Back-End Data Accessing**
- **Thin-Client versus Thick-Client Processing**
- **Interoperability Issues**
- **Testing Considerations**
- **Bibliography**

Introduction

This chapter presents the application model and shows how it applies to mainframes, PCs, and, ultimately, Web/client-server systems. It explores the technology differences between mainframes and Web/client-server systems, as well as the technology differences between PCs and Web/client-server systems. Testing methods that are suited to Web environments are also discussed.

Although many traditional software testing practices can be applied to the testing of Web-based applications, there are numerous technical issues that are specific to Web applications that need to be considered.

The Application Model

A computer system, which consists of hardware and software, can receive *inputs* from the user, then *stores* them somewhere, whether in volatile memory such as RAM (Random Access Memory), or in nonvolatile memory, such as hard disk memory. It can execute the instructions given by software by performing *computation* using the CPU (Central Processing Unit) computing power. Finally, it can process the *outputs* back to the user. Figure 2.1 illustrates how humans interact with computers. Through a user interface (UI), users interact with an application by offering input and receiving output in many different forms: query strings, database records, text forms, and so on. Applications take input, along with requested logic rules, and store them in memory, and then manipulate data through computing; they also perform file reading and writing (more input/output and data storing). Finally, output results are passed back to the user through the UI. Results may also be sent to other output devices, such as printers.

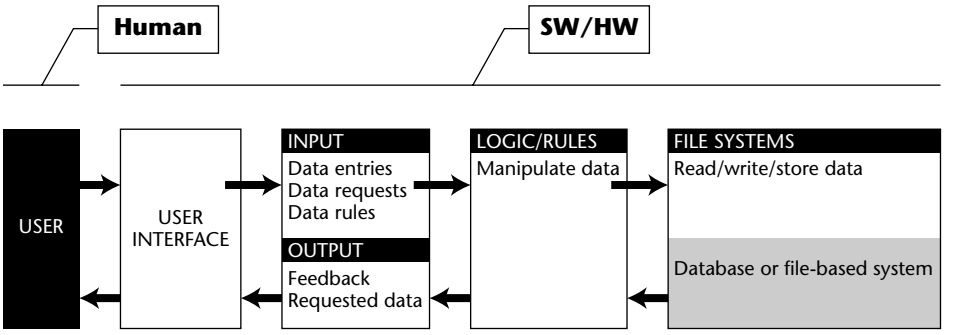

Figure 2.1 The application model.

In traditional mainframe systems, as illustrated in Figure 2.2, all of an application's processes, except for UI controls, occur on the mainframe computer. User interface controls take place on dumb terminals that simply echo text from the mainframe. Little computation or processing occurs on the terminals themselves. The network connects the dumb terminals to the mainframe. Dumb-terminal UIs are text-based or form-based (nongraphical). Users send data and commands to the system via keyboard inputs.

Desktop PC systems, as illustrated in Figure 2.3, consolidate all processes—from UI through rules to file systems—on a single physical box. No network is required for a desktop PC. Desktop PC applications can support either a text-based UI (command-line) or a Graphical User Interface (GUI). In addition to keyboard input events, GUI-based applications also support mouse input events such as click, double-click, mouse-over, drag-and-drop, and so on.

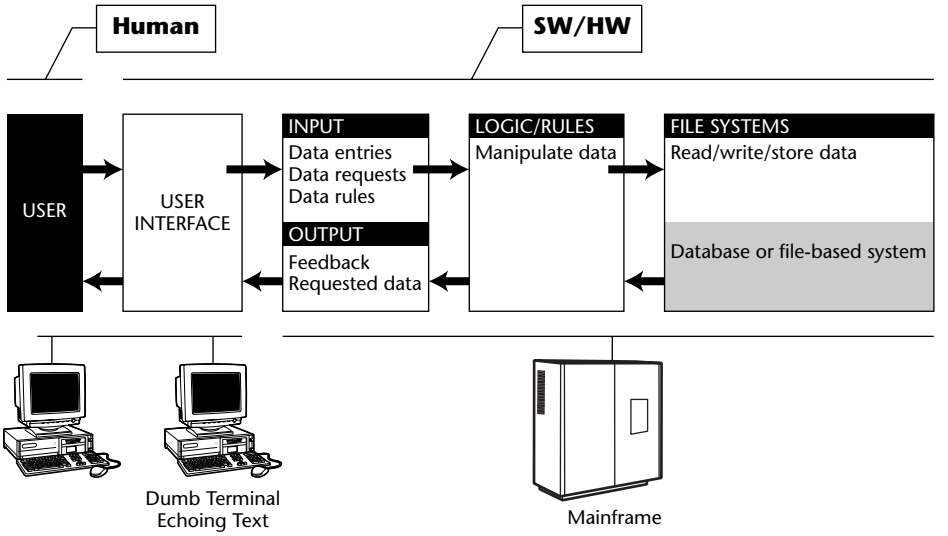

Figure 2.2 Mainframe systems.

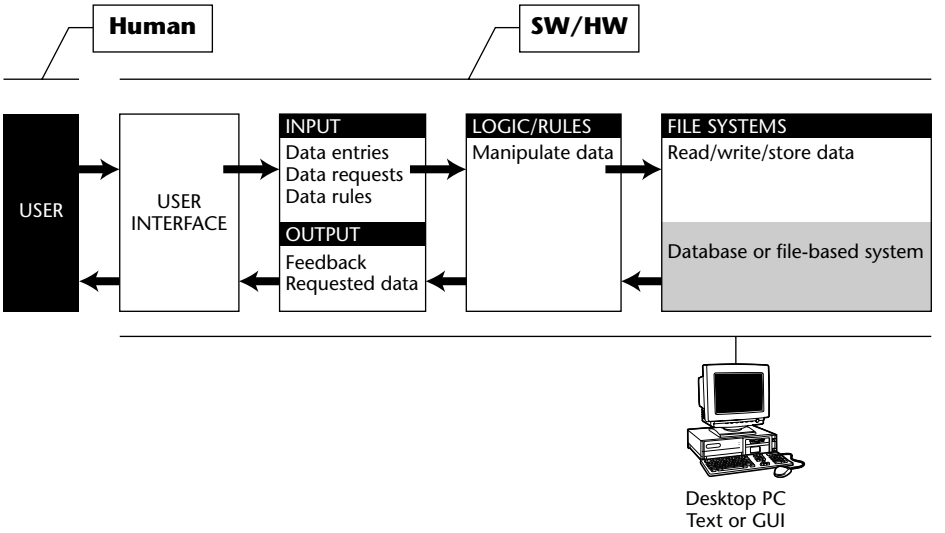

Figure 2.3 Desktop PC systems.

Client-server systems, upon which Web systems are built, require a network and at least two machines to operate: a *client* computer and a *server* computer, which serves requested data to the client computer. With the vast majority of Web applications, a Web browser serves as the UI container on the client computer.

The server receives input requests from the client and manipulates the data by applying the application's *business logic rules*. Business logic rules are the computations that an application is designed to carry out based on user input—for example, sales tax might be charged to any e-commerce customer who enters a California mailing address. Another example might be that customers over age 35 who respond to a certain online survey will be mailed a brochure automatically. This type of activity may require reading or writing to a database. Data is sent back to the client as output from the server. The results are then formatted and displayed in the client browser.

The client-server model, and consequently the Web application model, is not as neatly segmented as that of the mainframe and the desktop PC. In the client-server model, not only can either the client or the server handle some of the processing work, but server-side processes can be divided between multiple physical boxes or computers (application server, Web server, database server, etc.). Figure 2.4, one of many possible client-server models, depicts I/O and logic rules handled by an *application server* (the server in the center), while a *database server* (the server on the right) handles data storage. The dotted lines

in the illustration indicate processes that may take place on either the client-side or the server-side. See Chapter 5, "Web Application Components," for information regarding server types.

A Web system may comprise any number of physical server boxes, each handling one or more service types. Later in this chapter, Table 2.1 illustrates some of the possible three-box server configurations. Note that the example is relatively a basic system. A Web system may contain multiple Web servers, application servers, and multiple database servers (such as a *server farm*, a grouping of similar server types that share workload). Web systems may also include other server types, such as e-mail servers, chat servers, e-commerce servers, and user profile servers (see Chapter 5 for more information). Understanding how your Web application-under-test is structured is invaluable for bug analysis—trying to reproduce a bug or learning how you can find more bugs similar to the one that you are seeing.

Keep in mind that it is software, not hardware, that defines clients and servers. Simply put, clients are software programs that request services from other software programs on behalf of users. Servers are software programs that offer services. Additionally, *client-server* is also an overloaded term; it is only useful from the perspective of describing a system. A server may, and often does, become a client in the chain of requests. In addition, the server-side may include many applications and systems including mainframe systems.

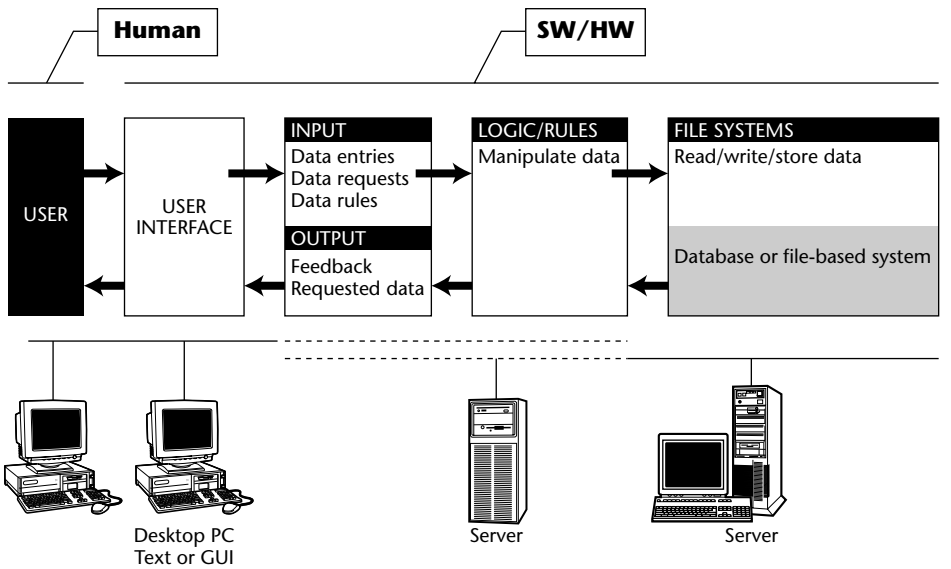

Figure 2.4 Client-server systems.

Hardware and Software Differences

Mainframe systems (Figure 2.5) are traditionally *controlled* environments, meaning that hardware and software are primarily supported (it does not necessarily mean that all the subcomponents are produced by the same company), end to end, by the same manufacturer. A mainframe with a single operating system, and applications sold and supported by the same manufacturer, can serve multiple terminals from a central location. Compatibility issues are more manageable compared to the PC and client-server systems.

A single desktop PC system consists of mixed hardware and software—multiple hardware components built and supported by different manufacturers, multiple operating systems, and nearly limitless combinations of software applications. Configuration and compatibility issues become difficult or almost impossible to manage in this environment.

A Web system consists of many clients, as well as server hosts (computers). The system's various flavors of hardware components and software applications begin to multiply. The server-side of Web systems may also support a mixture of software and hardware and, therefore, are more complex than mainframe systems, from the configuration and compatibility perspectives. See Figure 2.6 for an illustration of a client-server system running on a local area network (LAN).

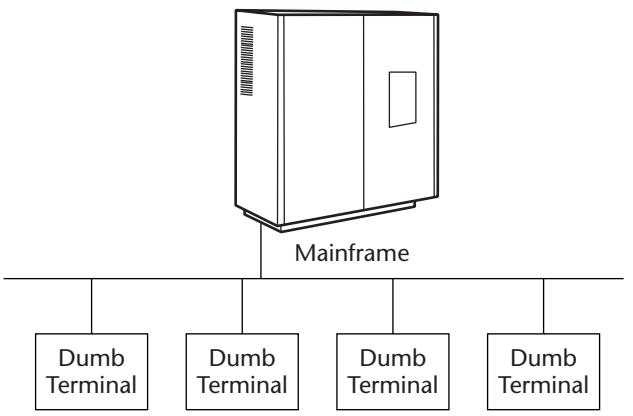

Figure 2.5 Controlled hardware and software environment.

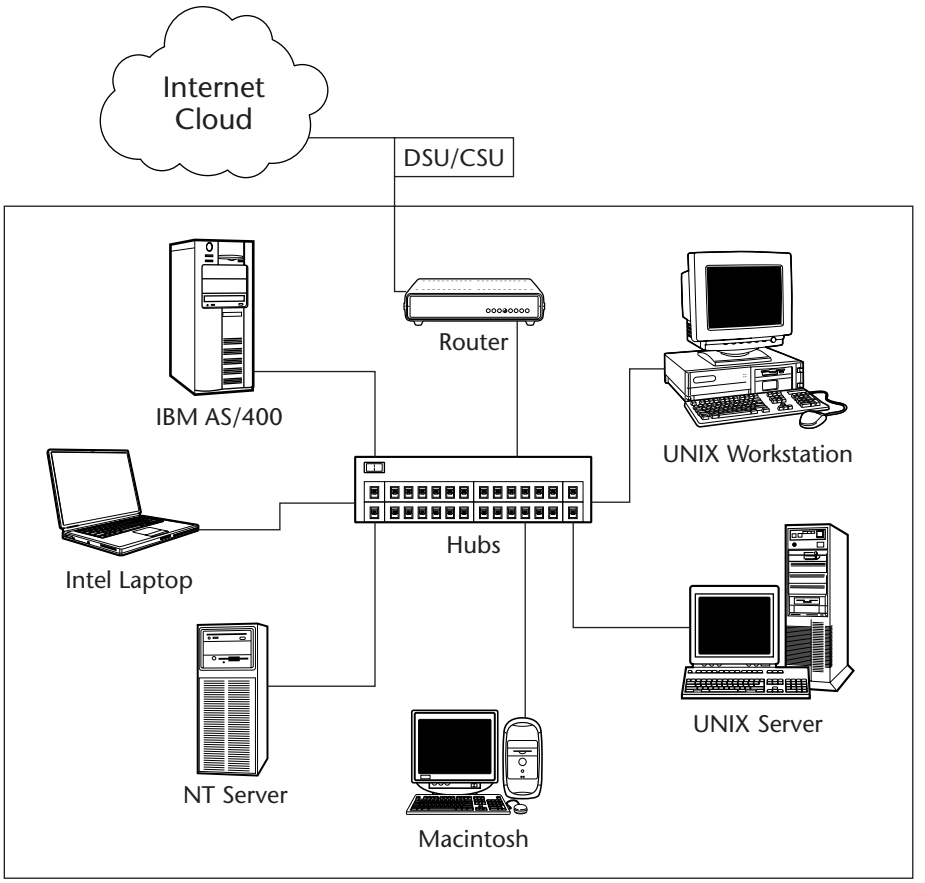

Figure 2.6 A client-server system on a LAN.

The GUI of the PC makes multiple controls available on screen at any given time (e.g., menus, pull-down lists, help screens, pictures, and command buttons.). Consequently, event-driven browsers (in event-driven model, inputs are driven by events such as a mouse click or a keypress on the keyboard) are also produced, taking advantage of the event-handling feature offered by the operating system (OS). However, event-based GUI applications (data input coupled with events) are more difficult to test. For example, each event applied to a control in a GUI may affect the behavior of other controls. Also, special dependencies can exist between GUI screens; interdependencies and constraints must be identified and tested accordingly.

The Differences between Web and Traditional Client-Server Systems

The last two sections point out the application architecture and hardware and software differences among the mainframe, PC, and Web/client-server systems. We will begin this section by exploring additional differences between Web and traditional systems so that appropriate testing considerations can be formulated.

Client-Side Applications

As illustrated in Figure 2.7, most client-server systems are data-access-driven applications. A client typically enables users, through the UI, to send input data, receive output data, and interact with the back end (for example, sending a query command). Clients of traditional client-server systems are platform-specific. That is, for each supported client operating system (e.g., Windows 16- and 32-bit, Solaris, Linux, Macintosh, etc.), a client application will be developed and tested for that target operating system.

Most Web-based systems are also data-access-driven applications. The browser-based clients are designed to handle similar activities to those supported by a traditional client. The main difference is that the Web-based client is operating within the Web browser's environment. Web browsers consist of operating system-specific client software running on a client computer. It renders HyperText Markup Language (HTML), as well as active contents, to display Web page information. Several popular browsers also support active content such as client-side scripting, Java applet, ActiveX control, eXtensible Markup Language (XML), cascading style sheet (CSS), dynamic HTML (DHTML), security features, and other goodies. To do this, browser vendors must create rendering engines and interpreters to translate and format HTML contents. In making these software components, various browsers and their releases introduce incompatibility issues. See Chapter 10, "User Interface Tests," and Chapter 17, "Configuration and Compatibility Tests," for more information.

From the Web application producer's perspective, there is no need to develop operating-system-specific clients since the browser vendors have already done that (e.g., Netscape, Microsoft, AOL, etc.). In theory, if your HTML contents are designed to conform to HTML 4 standard, your client application should run properly in any browser that supports HTML 4 standard from any vendor. But in practice, we will find ourselves working laboriously to address vendor-specific incompatibility issues introduced by each browser and its various releases. At the writing of this book, the golden rule is: "Web browsers are not created equal."

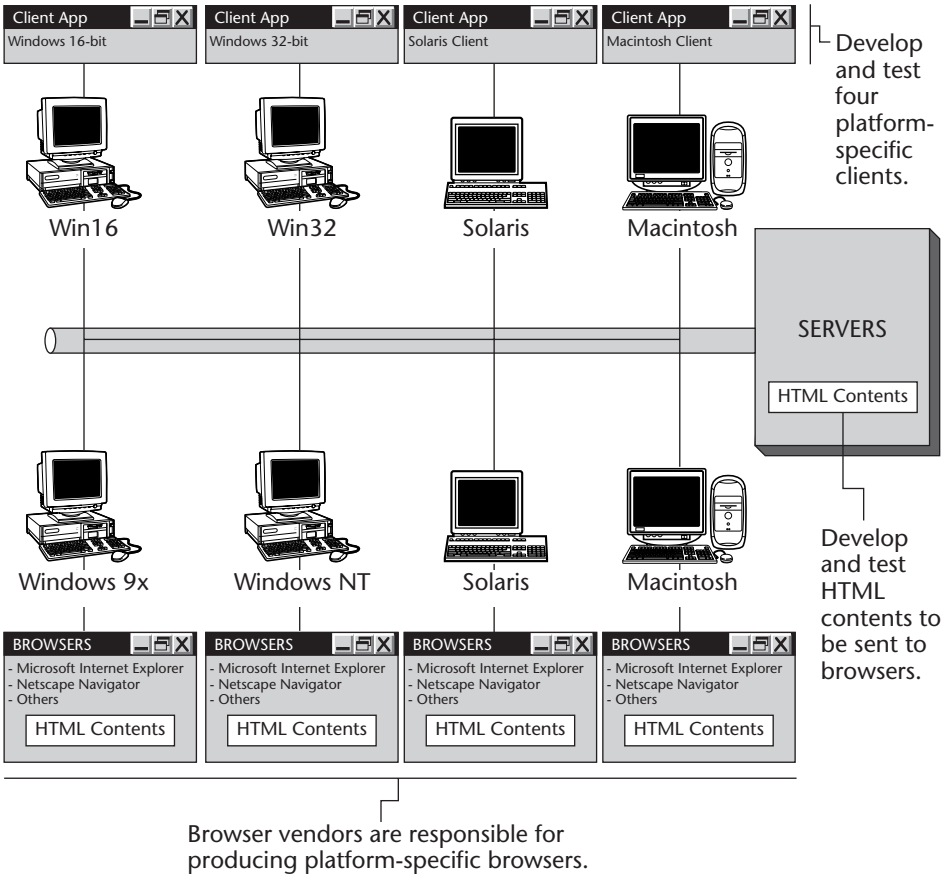

Figure 2.7 Client-server versus Web-based clients.

In addition to the desktop client computer and browser, there are new types of clients and browsers, which are a lot smaller than the desktop PC version. These clients are often battery-powered, rather than wall-electric-powered as is a desktop PC. These clients are mobile devices including PDAs (Personal Digital Assistants), smart phones, and handheld PCs. Since these devices represent another class of client computers, there are some differences in the mobile application model. For simplicity, in this chapter, we will only refer to the desktop PC in the client discussion. Read Chapter 6, "Mobile Web Application Platform," for discussions on the mobile client application model.

Event Handling

In the GUI and event-driven model, inputs, as the name implies, are driven by *events*. Events are actions taken by users, such as mouse movements and clicks,

or the input of data through a keyboard. Some objects (e.g., a push button) may receive mouse-over events whenever a mouse passes over them. A mouse single-click is an event. A mouse double-click is a different kind of event. A mouse click with a modifier key, such as Ctrl, is yet another type of event. Depending on the type of event applied to a particular UI object, certain procedures or functions in an application will be executed. In an event-driven environment, this is a type of procedure referred to as *event-handling code*.

Testing event-driven applications is more complicated because it's very labor-intensive to cover the testing of many combinations and sequences of events. Simply identifying all possible combinations of events can be a challenge because some actions trigger multiple events.

Browser-based applications introduce a different flavor of event-handling support. Because Web browsers were originally designed as a data presentation tool, there was no need for interactions other than single-clicking for navigation and data submission, and mouse-over ALT attribute for an alternate description of graphic. Therefore, standard HTML controls such as form-based control and hyperlinks are limited to single-click events. Although script-based events can be implemented to recognize other events such as double-clicking and drag-and-drop, it's not natural in the Web-based user interface to do so (not to mention that those other events also cause incompatibility problems among different browsers.

In Web-based applications, users may click links that generate simulated dialog boxes (the server sends back a page that includes tables, text fields, and other UI objects). Users may interact with browser-based UI objects in the process of generating input for the application. In turn, events are generated. Some of the event-handling code is in scripts that are embedded in the HTML page and executed on the client-side. Others are in UI components (such as Java applets and ActiveX controls) embedded in the HTML page and executed on the client-side. Still others are executed on the server-side. Understanding where (client- or server-side) each event is handled enables you to develop useful test cases as well as reproduce errors effectively.

Browser-based applications offer very limited keyboard event support. You can navigate within the page using Tab and Shift-Tab keys. You can activate a hyperlink to jump to another link or push a command button by pressing the Enter key while the hyperlink text, graphic, or a button is highlighted. Supports for keyboard shortcuts and access keys, such as Alt-[key] or Ctrl-[key], are not available for the Web applications running in the browser's environment, although they are available for the browser application itself. Another event-handling implication in browser-based applications is in the one-way request and submission model. The server generally does not receive commands or data until the user explicitly clicks a button, such as Submit to submit form data; or the user may request data from the server by clicking a link.

This is referred to as the *explicit submission model*. If the user simply closes down a browser but does not explicitly click on a button to save data or to log off, data will not be saved and the user is still considered logged on (on the server-side).

TEST CASE DEVELOPMENT TIPS

Based on the knowledge about the *explicit submission model,* try the following tests against your application under test:

TEST #1

- **Use your valid ID and password to log on to the system.**
- **After you are in, close your browser instead of logging off.**
- **Launch another instance of the browser and try to log in again. What happens? Does the system complain that you are already in? Does the system allow you to get in and treat it as if nothing has happened? Does the system allow you to get in as a different instance of the same user?**

TEST #2

Suppose that your product has a user-license restriction: That is, if you have five concurrent-user licenses, only five users can be logged on to the system concurrently. If the sixth user tries to log on, the system will block it and inform the sixth user that the application has run out of concurrent-user licenses.

- **Use your valid set of ID and password to log on to the system as six separate users. As the sixth user logs on, you may find that the system will detect the situation and block the sixth user from logging on.**
- **Close all five browsers that have already logged on instead of logging off.**
- **Launch another instance of the browser and try to log on again. Does the system still block this log on because it thinks that there are five users already on the system?**

TEST #3

- **Use your valid ID and password to log on to the system.**
- **Open an existing record as if you are going to update it.**
- **Close your browser instead of logging off.**
- **Launch another instance of the browser and try to log on again.**
- **Try to open the same record that you opened earlier in the previous browser. What happens? Is the record locked? Are you allowed to update the record anyway?**

By the way, there is no right or wrong answer for the three preceding tests because we don't know how your system is designed to handle user session, user authentication, and record locking. The main idea is to present you with some of the possible interesting scenarios that might affect your application under test due to the nature of the explicit submission model in Web-based applications.

Application Instance and Windows Handling

Standard event-based applications may support multiple instances, meaning that the same application can be loaded into memory many times as separate processes. Figure 2.8 shows two instances of Microsoft Word application. Similarly, multiple instances of a browser can run simultaneously. With multiple browser instances, users may be able to log in to the same Web-based application and access the same data table—on behalf of the same user or different users. Figure 2.9 illustrates two browser instances, each accessing the same application and data using the same or different user ID and password.

From the application's perspective, keeping track of multiple instances, the data, and the users who belong to each instance can be problematic. For example, a regular user has logged in using one instance of the browser. An Admin user has also logged in to the same system using another instance for the browser. It's common that the application server may mistakenly receive data from and send data to one user thinking that the data belongs to the other users. Test cases that uncover errors surrounding multiple-instance handling should be thoroughly designed and executed.

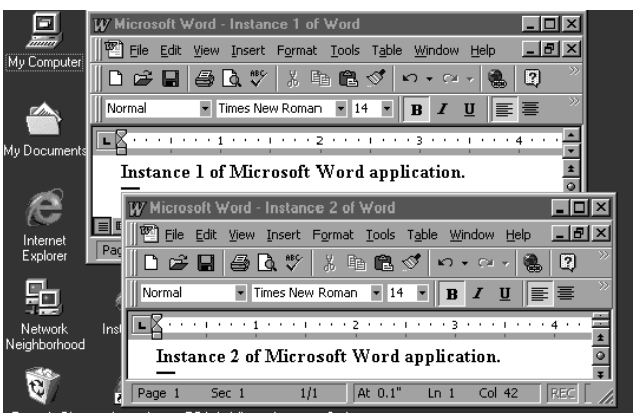

Figure 2.8 Multiple application instances.

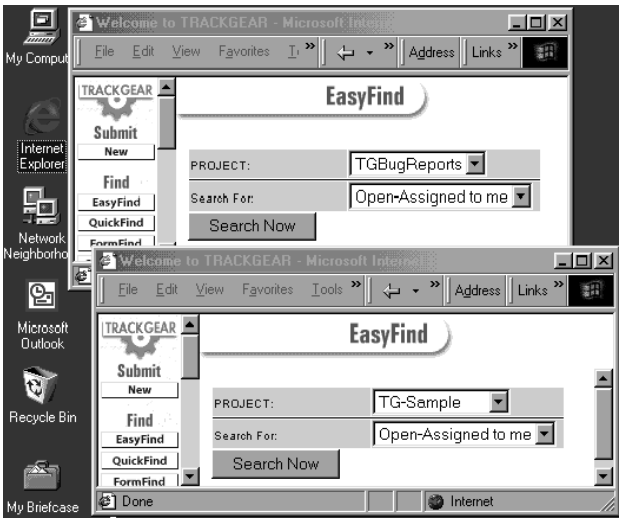

Figure 2.9 Multiple application windows.

Within the same instance of a standard event-based application, multiple windows may be opened simultaneously. Data altered in one of an application's windows may affect data in another of the application's windows. Such applications are referred to as *multiple document interface (MDI)* applications (Figure 2.10). Applications that allow only one active window at a time are known as *single document interface (SDI*) applications (Figure 2.11). SDI applications allow users to work with only one document at a time. Microsoft Word (Figure 2.10) is an example of an MDI application. Notepad (Figure 2.11) is an example of an SDI application.

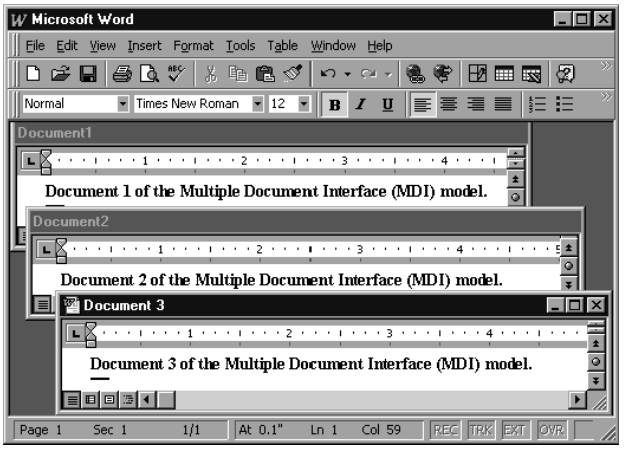

Figure 2.10 Multiple document interface (MDI) application.

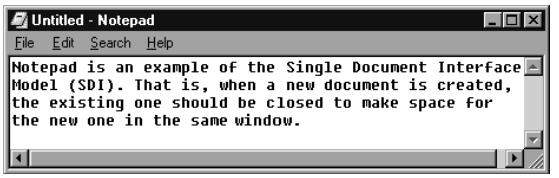

Figure 2.11 Single document interface (SDI) application.

Multiple document interface applications are more interesting to test because they might fail to keep track of events and data that belong to multiple windows. Test cases designed to uncover errors caused by the support of multiple windows should be considered.

Multiple document interface or multiple windows interface are only available for clients in a traditional client-server system. The Web browser interface is considered *flat* because it can only display one page at the time. There is no hierarchical structure for the Web pages, therefore, one can easily jump to several links and quickly lose track of the original position.

UI Controls

In essence, an HTML page that is displayed by a Web browser consists of text, hyperlinks, graphics, frames, tables, forms, and balloon help text (ALT tag). Basic browser-based applications do not support dialog boxes, toolbars, status bars, and other common UI controls. Extra effort can be made to take advantage of Java applets, ActiveX controls, scripts, CSS, and other helper applications to go beyond the basic functionality. However, there will be compatibility issues among different browsers.

Web Systems

The complexities of the PC model are multiplied exponentially in Web systems (Figure 2.12). In addition to the testing challenges that are presented by multiple client PCs and mobile devices, the server-side of Web systems involves hardware of varying types and a software mix of OSs, service processes, server packages, and databases.

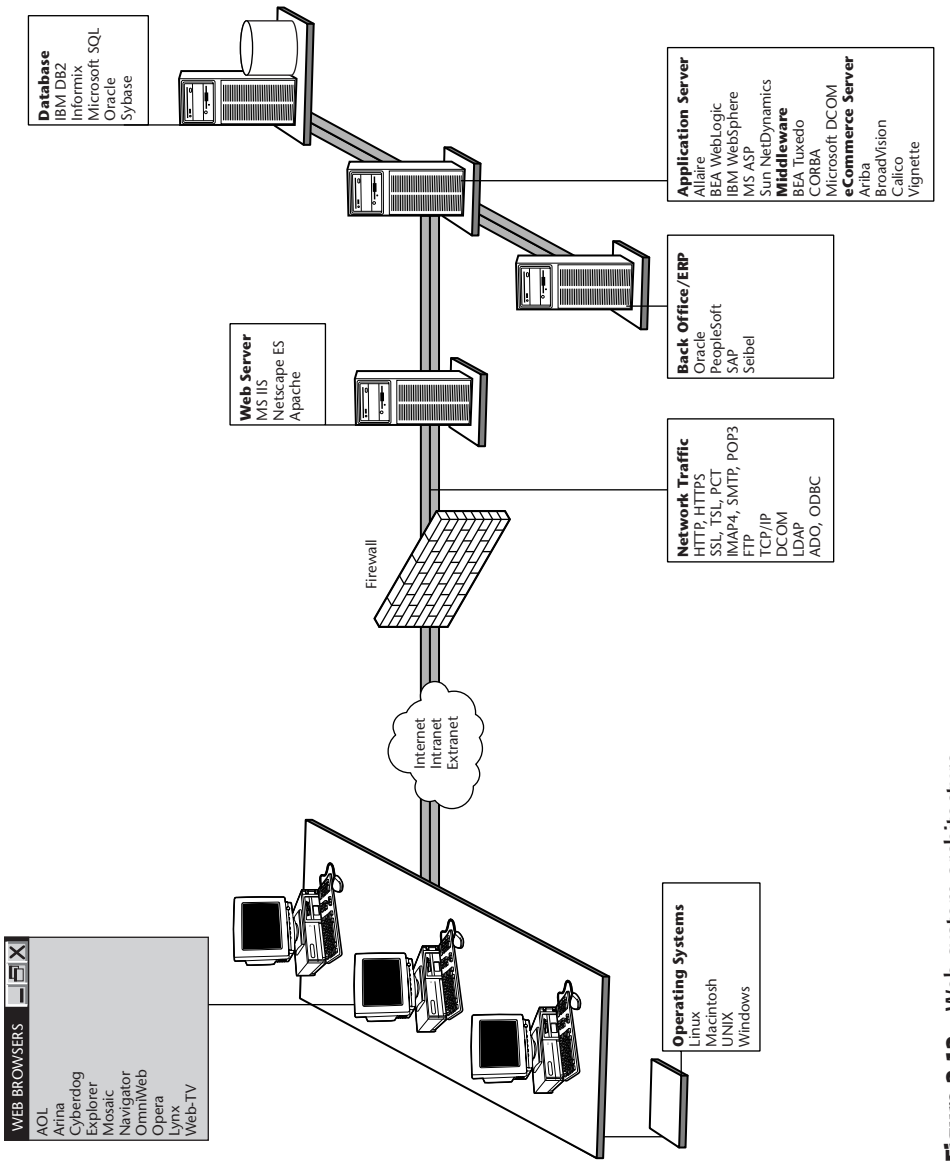

Figure 2.12 Web system architecture.

Hardware Mix

With Web systems and their mixture of many brands of hardware to support, the environment can become very difficult to control. Web systems have the capacity to use machines of different platforms, such as UNIX, Linux, Windows, and Macintosh boxes. A Web system might include a UNIX server that is used in conjunction with other servers that are Linux, Windows-based, or Macintosh-based. Web systems may also include mixtures of models from the same platform (on both the client- and server-sides). Such hardware mixtures present testing challenges because different computers in the same system may employ different OSs, CPU speeds, buses, I/O interfaces, and more. Each combination has the potential to cause problems.

Software Mix

At the highest level, as illustrated in Figure 2.12, Web systems may consist of various OSs, Web servers, application servers, middleware, e-commerce servers, database servers, major enterprise resource planning (ERP) suites, firewalls, and browsers. Application development teams often have little control over the kind of environment into which their applications are installed. In producing software for mainframe systems, development was tailored to one specific system. Today, for Web systems, software is often designed to run on a wide range of hardware and OS combinations, and risks of software incompatibility are always present. An example is that different applications may not share the same version of a database server. On the Microsoft platform, a missing or incompatible DLL (dynamic link library) is another example. (Dynamic link libraries are software components that can exist on both the client- and server-sides whose functions can be called by multiple programs on demand.)

Another problem inherent in the simultaneous use of software from multiple vendors is that when each application undergoes a periodic upgrade (client- or server-side), there is a chance that the upgrades will not be compatible with preexisting software.

A Web system software mix may include any combination of the following:

- Multiple operating systems
- Multiple software packages
- Multiple software components
- Multiple server types, brands, and models
- Multiple browser brands and versions

Server-Based Applications

Server-based applications are different from client applications in two ways. First, server-based applications are programs that don't have a UI with which the end users of the system interact. End users only interact with the client-side application. In turn, the client interacts with server-based applications to access functionality and data via communication protocols, application programming interface (API), and other interfacing standards. Second, server-based applications run unattended; that is, when a server-based application is started, it's intended to stay up, waiting to provide services to client applications whether there is any client out there requesting services. In contrast, to use a client application, an end user must explicitly launch the client application and interact with it via a UI. Therefore, to black-box testers, server-based applications are black boxes.

You may ask: "So it is with desktop applications. What's the big deal?" Here is an example: When a failure is caused by an error in a client-side or desktop application, the users or testers can provide essential information that helps reproduce or analyze the failure because they are right in front of the application. Server-based applications or systems are often isolated from the end users. When a server-based application fails, as testers or users from the client-side, we often don't know when it failed, what happened before it failed, who was or how many users were on the system at the time it failed, and so on. This makes reproducing bugs even more challenging for us. In testing Web systems, we need a better way to track what goes on with applications on the server-side.

One of the techniques used to enhance our failure reproducibility capability is *event logging*. With event logging, server-based applications can record activities to a file that might not be normally seen by an end user. When an application uses event logging, the recorded information that is saved can be read in a reliable way. Operating systems often include logging utilities. For example, Microsoft Windows 2000 includes the Event Viewer, which enables users to monitor events logged in the Application (the most interesting for testing), Security, and System logs. The Application log allows you to track events generated by a specific application. For example, you might want the log file to read and write errors generated by your application. The Application log will allow you to do so. You can create and include additional logging capabilities to your application under test to facilitate the defect analysis and debugging process, should your developers and your test teams find value in them. (Refer to the "Server-Side Testing Tips" section of Chapter 12, "Server-Side Testing," for more information on using log files.) Have discussions with your developers to determine how event logging can be incorporated into or created to support the testing process.

Distributed Server Configurations

Server software can be distributed among any number of physical server boxes, which further complicates testing. Table 2.1 illustrates several possible server configurations that a Web application may support. You should identify the configurations that the application under test claims to support. Matrices of all possible combinations should be developed. Especially for commercial-off-the-shelf server applications, testing should be executed on each configuration to ensure that application features are intact. Realistically, this might not be possible due to resource and time constraints. For more information on testing strategies, see Chapter 17.

Table 2.1 Distributed Server Configurations

	BOX 1	BOX 2	BOX 3
One-box model	NT-based Web server		
	NT-based application server		
	NT-based database server		
Two-box model	NT-based Web server	NT-based database server	
	NT-based application server		
Three-box model	NT-based Web server	NT-based Web server	UNIX-based database server
	NT-based application server	NT-based application server	
One-box model	UNIX-based Web server		
	UNIX-based application server		
	UNIX-based database server		
Two-box model	UNIX-based Web server	UNIX-based database server	
	UNIX-based application server		

(continued)

Table 2.1 *(continued)*

	BOX 1	BOX 2	BOX 3
Three-box model	NT-based Web server	NT-based Web server	NT-based database server
	NT-based application server	NT-based application server	

The Network

The network is the glue that holds Web systems together. It connects clients to servers and servers to servers. This variable introduces new testing issues, including reliability, accessibility, performance, security, configuration, and compatibility. As illustrated in Figure 2.12, the network traffic may consist of several protocols supported by the TCP/IP network. It's also possible to have several networks using different net OSs connecting to each other by gateways. Testing issues related to the network can be a challenge or beyond the reach of black-box testing. However, understanding the testing-related issues surrounding the network enables us to better define testing problems and ask for appropriate help. (See Chapter 4, "Networking Basics," for more information.) In addition, with the proliferation of mobile devices, wireless networks are becoming more popular. It is useful to also have a good understanding of how a wireless network may affect your Web applications, especially mobile Web applications. (See Chapter 6, "Mobile Web Application Platform," for an overview of wireless network.)

Bug Inheritance

It is common for Web applications to rely on preexisting objects or components. Therefore, the newly created systems inherit not just the features but also the bugs that existed in the original objects.

One of the important benefits of both *object-oriented programming* (OOP) and *component-based programming* is reusability. Rather than writing the code from scratch, a developer can take advantage of preexisting features created by other developers (with use of the application programming interface or the API and proper permission) by incorporating those features into his or her own application. In effect, code is recycled, eliminating the need to rewrite existing code. This model helps accelerate development time, reduces the amount of code that needs to be written, and maintains consistency between applications.

The potential problem with this shared model is that bugs are passed along with components. Web applications, due to their component-based architecture, are particularly vulnerable to the sharing of bugs.

At the lowest level, the problem has two major impacts on testing. First, existing objects or components must be tested thoroughly before other applications or objects can use their functionality. Second, regression testing must be executed comprehensively (see "Regression Testing" in Chapter 3, "Software Testing Basics," for more information). Even a small change in a parent object can alter the functionality of an application or object that uses it.

This problem is not new. Object-oriented programming and component-based software have long been used in PCs. With the Web system architecture, however, the problem is multiplied due to the fact that components are shared across servers on a network. The problem is exacerbated by the demand that software be developed in an increasingly shorter time.

At the higher level, bugs in server packages, such as Web servers and database servers, and bugs in Web browsers themselves, will also have an effect on the software under test (see Chapter 5 "Web Application Component," for more information). This problem has a greater impact on security risks. Chapter 18, "Web Security Testing," contains an example of a Buffer Overflow error.

Back-End Data Accessing

Data in a Web system is often distributed; that is, it resides on one or more (server) computers rather than the client computer. There are several methods of storing data on a back-end server. For example, data can be stored in flat files, in a nonrelational database, in a relational database, or in an object-oriented database. In a typical Web application system, it's common that a relational database is employed so that data accessing and manipulation can be more efficient compared to a flat-file database.

In a flat-file system, when a query is initiated, the results of that query are dumped into files on a storage device. An application then opens, reads, and manipulates data from these files and generates reports on behalf of the user. To get to the data, the applications need to know exactly where files are located and what their names are. Access security is usually imposed at the application level and file level.

In contrast, a database, such as a relational database, stores data in tables of records. Through the database engine, applications access data by getting a set of records without knowing where the physical data files are located or what they are named. Data in relational databases are accessed via database names (not to be mistaken with file names) and table names. Relational database files can be stored on multiple servers. Web systems using a relational database can impose security at the application server level, the database server level, as

well as at table and user-based privilege level. All of this means that testing back-end data accessing by itself is a big challenge to testers, especially black-box testers because they do not see the activities on the back end. It makes reproducing errors more difficult and capturing test coverage more complicated. See Chapter 14, "Database Tests," for more information on testing back-end database accessing.

Thin-Client versus Thick-Client Processing

Thin-client versus thick-client processing is concerned with where applications and components reside and execute. Components may reside on a client machine and on one or more server machines. The two possibilities are:

Thin client. With thin-client systems, the client PC or mobile device does very little processing. Business logic rules are executed on the server side. Some simple HTML Web-based applications and handheld devices utilize this model. This approach centralizes processing on the server and eliminates most client-side incompatibility concerns. (See Table 2.2.)

Thick client. The client machine runs the UI portion of the application as well as the execution of some or all of the business logic. In this case, the browser not only has to format the HTML page, but it also has to execute other components such as Java applet and ActiveX. The server machine houses the database that processes data requests from the client. Processing is shared between client and server. (See Table 2.3.)

Table 2.2 Thin Client

DESKTOP PC THIN CLIENT	SERVER
UI	Application rules
	Database

Table 2.3 Thick Client

DESKTOP PC THICK CLIENT	SERVER
UI	Database
Application rules	

The client doing much of a system's work (e.g., executing business logic rules, DHTML, Java applets, ActiveX controls, or style sheets on the client-side) is referred to as *thick-client processing*. Thick-client processing relieves processing strain on the server and takes full advantage of the client's processor. With thick-client processing, there are likely to be more incompatibility problems on the client-side.

Thin-client versus thick-client application testing issues revolve around the compromises among feature, compatibility, and performance issues. For more information regarding thin-client versus thick-client application, please see Chapter 5.

Interoperability Issues

Interoperability is the ability of a system or components within a system to interact and work seamlessly with other systems or other components. This is normally achieved by adhering to certain APIs, communication protocol standards, or to interface-converting technology such as Common Object Request Broker Architecture (CORBA) or Distributed Common Object Model (DCOM). There are many hardware and software interoperability dependencies associated with Web systems, so it is essential that our test-planning process include study of the system architectural design.

Interoperability issues—it is possible that information will be lost or misinterpreted in communication between components. Figure 2.13 shows a simplified Web system that includes three box servers and a client machine. In this example, the client requests all database records from the server-side. The application server in turn queries the database server. Now, if the database server fails to execute the query, what will happen? Will the database server tell the application server that the query has failed? If the application server gets no response from the database server, will it resend the query? Possibly, the application server will receive an error message that it does not understand. Consequently, what message will be passed back to the client? Will the application server simply notify the client that the request must be resent? Or will it neglect to inform the client of anything at all? All of these scenarios need to be investigated in the study of the system architectural design.

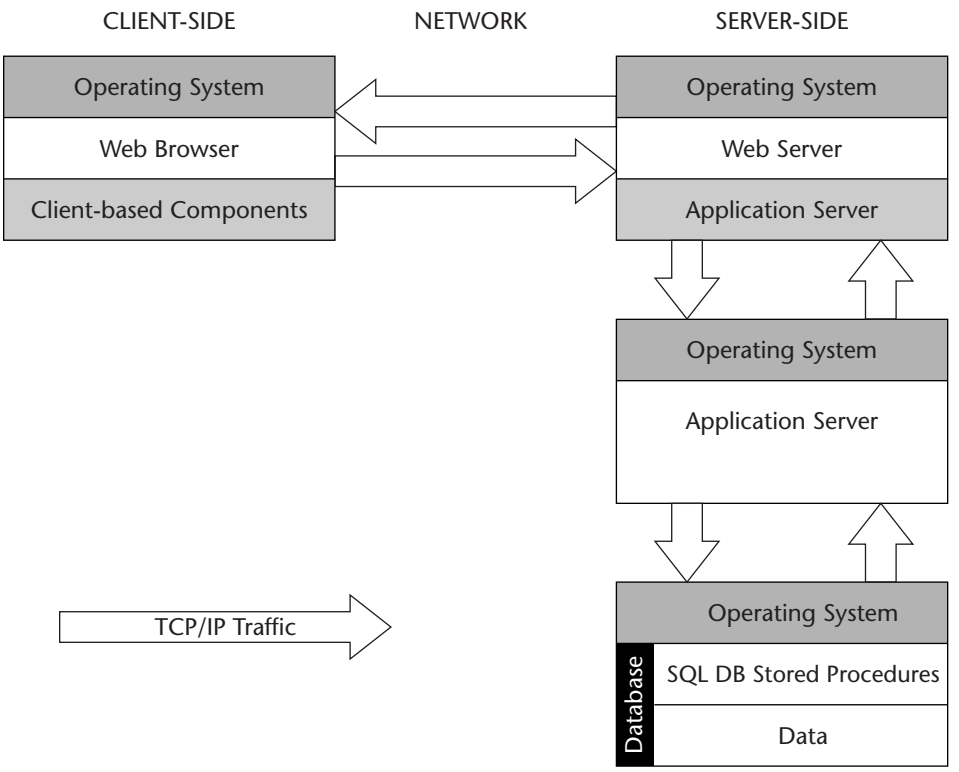

Figure 2.13 Interoperability.

Testing Considerations

The key areas of testing for Web applications beyond traditional testing include:

- Web UI implementation
- System integration
- Server and client installation
- Web-based help
- Configuration and compatibility
- Database
- Security
- Performance, load, and stress

For definitions for these tests, see Chapter 3, "Software Testing Basics." In addition, see Chapters 9 through 19 for in-depth discussions on these tests.

Now that we have established the fact that testing Web applications is complex, our objective for the rest of the book is to offer you quick access to information that can help you meet the testing challenges. The materials are built upon some of the testing knowledge that you already have. We hope that you will find the information useful.

Bibliography

Kaner, Cem, Jack Falk, Hung Q. Nguyen *Testing Computer Software,* 2nd edition. New York: John Wiley & Sons, Inc., 1999.

LogiGear Corporation. *QA Training Handbook: Testing Web Applications.* Foster City, CA: LogiGear Corporation, 2002.

———*QA Training Handbook: Testing Windows Desktop and Server-Based Applications.* Foster City, CA: LogiGear Corporation, 2002.

Microsoft Corporation. *Microsoft SQL Server 2000 Resource Kit.* Redmond, WA: Microsoft Press, 2001.

Orfali, Robert, Dan Harkey, Jeri Edwards, *Client/Server Survival Guide, Third Edition.* New York: John Wiley & Sons, 1999.

Reilly, Douglas J. *Designing Microsoft ASP.NET Applications*. Redmond, WA: Microsoft Press, 2001.

———*Inside Server-Based Applications.* Redmond, WA: Microsoft Press, 2000.