▶▶ QUICK LOOK

- Why errors in Web-based applications are hard to reproduce
- Analyzing these errors in a way that saves testing time

Testing Web-based Applications

Analyzing and reproducing errors in a Web environment

by Hung Q. Nguyen

The testing of Web-based applications has much in common with the testing of desktop systems: You need to test the usual functionality, configuration, and compatibility, as well as performing all the standard test types. But Web application testing is more difficult because complexities are multiplied by all the distributed system components that interact with the application. When we see an error in a Web environment, it's often difficult to pinpoint where the error occurs, and, because the behavior we see or the error message we receive may be the result of errors happening on different parts of the Web system, the error may be difficult to reproduce. So how do we analyze errors within a Web-based system, and what considerations should be made for reproducing such errors?

When we have an understanding of the underlying technology, we are better able to maximize testing efficiencies—writing more reproducible bug reports and finding more errors in less time. This is easier said than done…*especially in Web environments.* Web environments are dense with error-prone technology variables. Here are five fundamental considerations of Web-application testing:

1. When we see an error on the client side, we are seeing the symptom of an error—not the error itself.
2. Errors may be environment-dependent and may not appear in different environments.
3. Errors may be in the code or in the configuration.
4. Errors may reside in any of several layers.
5. Examining the two classes of operating environments—*static* versus *dynamic*—demands different approaches.

Now let's take a look at each of these five considerations in more detail:

1 What are we really seeing–an error or a symptom?

Without diagnosing the environment, we can't be certain what causes a symptom to appear. If one of the environment-specific variables from either the client side or the server side is removed or altered, we might not be able to reproduce the problem.

Here is an example. I'm testing a Web-based defect tracking application, and going through the process of creating a new bug report. When I select the NEW button, I receive an error message:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

After spending some time investigating my browser environment, I discover that JavaScript is disabled in the browser preferences dialog box. Enabling JavaScript eliminates the error. (Whether or not this is a bug is not part of this discussion.) The idea here is that if I add additional information regarding the JavaScript setting to the bug report, I can save our team some time in analyzing this problem. Furthermore, "disabling JavaScript" is added to my test suite from this point on; it will be applied to all areas of the application so that all potentially related errors can be uncovered.

2

Is the error environment-dependent?

To reproduce an *environment-dependent* error we have to perfectly replicate both the exact sequence of activities *and* the environment conditions (operating system, browser version, add-on components, database server, Web server, third-party components, server/client resources, network bandwidth and traffic, etc.) in which the application operates. For example, when you try to log into your Web application while using a 28.8 kbps dial-up connection, you experience login failures due to timeout in the authentication process—but the same login steps will authenticate successfully if you are on a T-1 connection at 1.54 mbps. In this case, you have an environment-dependent error where the dependency is in the bandwidth.

Environment-*independent* errors, on the other hand, are relatively easier to reproduce—it's not necessary to replicate the operating environment. With environment-independent errors, all that need be replicated is the steps that reveal the error. For example, if the company name is misspelled on all of the product's online pages as **WebTessting.Con**, you will *always* see this error—independent of the hardware, software, and resource variables in your operating environment. More commonly, we refer to environment-independent errors as *functionality-specific* errors.

3

Is it a coding error or a configuration problem?

Errors (or the symptoms of supposed errors) may be resolved with code fixes (assuming the errors are in fact real) or system reconfiguration (client, server, or network). Don't jump too quickly to the conclusion that it's a *bug!*

To reproduce an environment-dependent error we have to perfectly replicate both the exact sequence of activities and the environment conditions.

Microsoft OLE DB Provider for ODBC Drivers error '80004005'

Here is an example illustrating the challenge of identifying possible configuration problems as opposed to actual software errors. It shows an error message caused by a "failed login" that has been generated by a Web application. By simply looking at this error message, it is impossible to determine whether this error is the result of a software bug, a server-side configuration issue, a compatibility issue, a browser configuration issue, or all of the above.

After further analyzing the failure, I discover several possible conditions that might generate this error message:

IIS (Web server) virtual directory has not been set up properly When the virtual directory is not properly configured, the requested files, scripts, or data will not be found. Typically, this is a server configuration issue. However, if the installation program failed to programmatically configure the Web server according to specification, then this is a *software error*. If a system administrator fails to properly configure the Web server according to specification, this then becomes a *user error*.

Application directory has not been configured properly to execute scripts A typical application-server directory contains scripts to be executed when they are called by a Web server on the behalf of a client. For security reasons, a Web server can be configured to allow or disallow scripts to be executed within certain directories. If your application-server directory is designed to contain scripts that will be executed—but the Web server is configured to disable script execution in that directory—the application will not work. Is this a *software error* or a *configuration problem?*

Default Web page has not been set up properly The issue is similar to the problem above.

SQL Server is not running The application server needs to connect to the backend database living on the SQL server in order to execute queries, store procedures, and access data. If the SQL server process itself is not running, then obviously the application will not work.

DLL/COM objects are missing or were unsuccessfully registered Perhaps the installation program failed to copy all the DLLs used by the application server during setup. If any DLL needed by the application server is missing, the application will not work.

Perhaps the installation program correctly copied all the needed modules, but failed to register one or more of them. For example, with OLE-based objects such as COM or DCOM, their class ID (CLSID) must be registered in the Registry Database before they can be used. If an application tries to access a COM object that was not registered successfully, the application will not work.

This problem is often caused by errors in the installation procedures. If, on the other hand, the components must be manually registered then this becomes a *configuration issue.*

Browser-side JavaScript setting has been disabled This is a browser-side configuration problem since the application requires the browser to have JavaScript enabled. Is this a software error, a configuration problem, or a technical support issue?

4 Which layer really causes the problem?

Errors in Web systems are often difficult to consistently reproduce because of the many variables introduced by the distributed nature of client/server architecture (i.e., server, client, and networking components). There are at least three usual suspects in a Web environment: The *client*, the *server*, and the *network*.

Both the client and the server carry configuration and compatibility issues that are similar to PC environments, where all components are in one box. Issues multiply within client/server systems, however, because there may be many clients and servers connected on a network. Typical client/server configuration and compatibility issues involve the hardware and operating system mix (UNIX-based boxes versus Windows-based boxes, for example) and the software mix on the server side (Web server packages, database server packages, firewalls, COM objects, CORBA objects, etc.). Issues may also involve the software mix on the client side (TCP/IP stacks, dialer software, helper components, browser brands, and browser versions). Additionally, browser settings, such as general settings, connection settings, security settings (including Ac-

EDIT PASS 05/21/99

Making Your Web Application Test Report More Reproducible

- Check if the client operating system, versions, and patches meet system requirements
- Check if the correct version of the browser is installed on the client machine
- Check if the browser is properly installed on the machine (for example, the JVM is also successfully installed)
- Check the browser settings
- Try the same set of steps with different browsers (e.g., Netscape Navigator versus Internet Explorer)
- Try the same set of steps with different supported versions of the same browsers (e.g., 3.1, 3.2, 4.2, 4.3, etc.)
- Check to ensure that all servers are running
- Check to ensure that all service-based components have been started
- Check to ensure that application access privileges are properly set up
- Check for missing components on the server (DLLs, scripts, etc.)
- Check for proper registration of components (COMs, Java, etc.)
- Check to ensure that DNS is properly configured
- Check if firewall configuration is causing packets to drop or blocking access
- Check if a slow connection is causing the application to time-out
- Check for potential race or time-related conditions
- Check for potential network inaccessibility issues on the client machines
- Check for potential network inaccessibility issues on the server machines
- Check if the server operating system version and patches meet system requirements
- Check if the proper versions of the server software such as Web server, SQL database, and other middle-ware packages are installed
- Check server configurations for proper settings

CREDIT CREDIT CREDIT

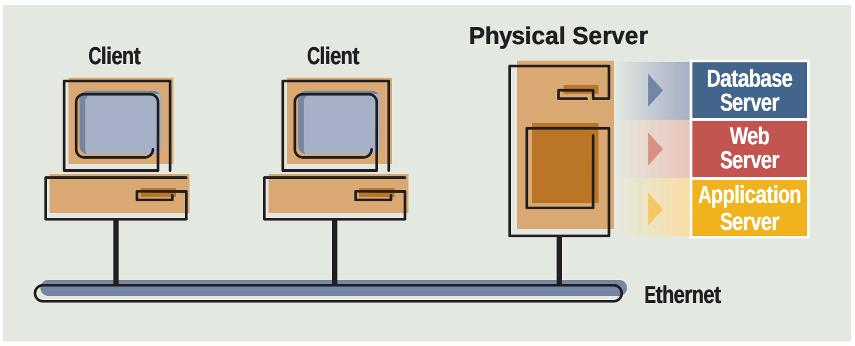


FIGURE 1 **Web server, application server, and database server in one box**

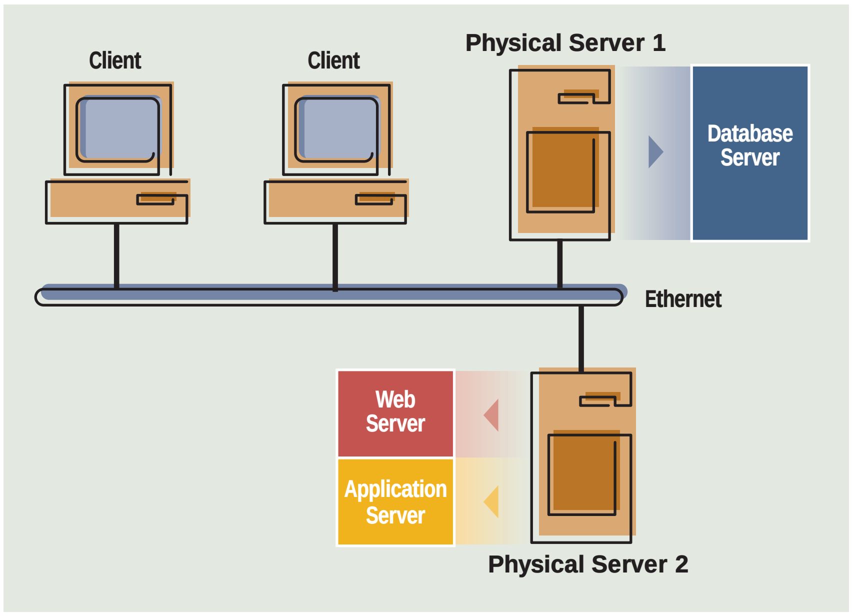


FIGURE 2 **Web server and application server in one box; database server in another box**

tiveX controls, plug-ins, Java, scripting, downloads, user authentication, etc.), content settings, program settings, and other advanced settings (including browsing options, multimedia options, Java VM options, printing options, and HTTP options) introduce a multitude of variables that should be tested and included in analyses.

The network offers another set of variables. The network affects the Web application in several ways, including timing-related issues (race conditions, performance, time-outs, etc.) due to bandwidth and latency, potential configuration and compatibility issues due to hardware devices such as gateways and routers, and side effects related to security implementations.

ANNIE BISSETT

5

Static and dynamic operating environments are different.

In general, there are two classes of operating environments—each with its own unique testing implications:

Static Environments (i.e., configuration and compatibility errors) in which incompatibility issues may exist regardless of variable conditions such as processing speed and available memory.

Dynamic Environments (i.e., resource and time-related errors) in which otherwise compatible components may exhibit errors due to memory-related errors and latency conditions. (We'll discuss dynamic environments in more detail later in this section.)

Static Operating Environment: Configuration and Compatibility Variables

Configuration and compatibility issues may occur at any point within a Web system: client, server, or network. *Configuration* issues involve various server software and hardware set-ups, browser settings, network connections, and TCP/IP stack set-ups. The browser setting/JavaScript example discussed earlier illustrated one type of configuration issue. A different type of configuration issue is shown in Figures 1 and 2, with two possible physical server configurations: *one-box* and *two-box* configurations.

Our sample application under test has some charting capabilities that enable a user to generate metrics reports, such as bar charts and line charts. When a user requests a metrics report, the application server pseudo code runs as follows:

1. Connect to the database server and run the query.
2. Write the query result to a file named **c:\temp\chart.val**
3. Execute the **Chart** JavaApplet. Read from **c:\temp\chart.val** and use the data to draw a graph.
4. Send the JavaApplet to the browser.

During testing for this application, I discovered that the charting feature worked on one of the above configurations, but not the other. After I investigated further, I learned that the problem only occurred in the two-box configuration. After examining the code, I realized that the problem is in steps 2 and 3. In step 2, the query result is written to **c:\temp\chart.val** of the database server local drive. In step 3, the **Chart** JavaApplet is running on the application server that is not in the same

I am not suggesting that we read the code every time we come across an error…I merely want to point out that it is essential to identify which server configurations are problematic, and include such information in bug reports. I would also run a cursory suite of test cases on all distributed configurations that are supported by the application server under test.

box with the database server. When it attempts to open the file **c:\temp\chart.val** on the application server local drive, the file is not there.

In this case, I am not suggesting that we read the code every time we come across an error; I leave the debugging work for the developers. I merely want to point out that it is essential to identify which server configurations are problematic, and include such information in bug reports. I would also run a cursory suite of test cases on all distributed configurations that are supported by the application server under test.

Compatibility issues are also important in static operating environments. As an example, in Figure 3 we see a compatibility difference between Netscape Navigator and Internet Explorer.

This is not to say that Internet Explorer is better than Netscape Navigator; it simply means that there are incompatibility issues between browsers—and that the code should not assume that relative paths work for all browsers. More importantly, it suggests that when you experience an error in one environment, the same error may *not* appear in a different environment if it's an environment-dependent error.

Dynamic Operating Environment: Things Don't Stay the Same

When the value of a specific environment attribute does not stay constant each time a test procedure is executed, it causes the operating environment to become *dynamic*. The attribute can be anything from resource-specific (available RAM, disk space, etc.) to timing-specific (network latency, the order of user transactions being submitted, etc.).

When a test case depends on the *exact* replication of both the *set of steps* and the *operating environment* but the operating environment cannot be replicated (due to its dynamic nature), the error becomes irreproducible or hard-to-reproduce.

By the way, this is the reason that memory-related errors are often hard to reproduce. When a memory-overwrite error exists in the code, for example, it will always cause a memory-overwritten problem. However, from a black-box testing perspective, we will never have a chance to see the symptom of this error until the specific overwritten byte(s) of code or data is executed or read. In this example, the set of steps represents the exact set of black-box activities. The memory-overwrite error represents the actual error in the code. The condition in which the overwritten byte is executed or read represents the dynamic operating environment or condition needed to reveal (reproduce) the error.

Here is a Web application example of a dynamic environment-related error in which we will examine a time-related error. The specification requires that:

- Project names within the system must be unique

The home directory path for the Web server on the host **myserver** is mapped to:
C:\INETPUB\WWWROOT\

When a page is requested from **http://myserver/** data will be pulled from:
C:\INETPUB\WWWROOT\

A filename (**mychart.jar**) is stored at **C:\INETPUB\WWWROOT\MYAPP\BIN**.

The application session path *(relative path)* is pointing to
C:\INETPUB\WWWROOT\MYAPP\BIN, and a file is requested from **.\LIB**.

If I use Internet Explorer version 3.x, the Web server looks for the file in **C:\INETPUB\WWWROOT\MYAPP\BIN\LIB** because the browser relies on the relative paths. This is the intended behavior and the file will be found; this tells me that my application will work as expected using Internet Explorer 3.x.

If instead I use Netscape Navigator version 3.x (a browser that doesn't like **.**), the Web server defaults to **C:\INETPUB\WWWROOT\LIB** and tries to look for **mychart.jar** from there instead. This is a problem for this particular application because the file (**mychart.jar**) will not be found there—so I know this feature will not work using Netscape 3.x.

When I brought up the Java Console, I saw the following, which confirmed my finding:
#Unable to load archive
http://myserver/lib/mychart.jar:java.io.IOException:<null>

FIGURE 3 **Compatibility issue between browsers**

- Error detection and handling for potential duplication be performed on the client-side using JavaScript
- Users will be able to add or delete project names by requesting the *Setting Up Projects* page
- When a user creates a new project name, a browser-side JavaScript checks the input name against the *select* list embedded in the HTML page (as illustrated in Figure 4)

Take a look at the time-related error illustrated in Figure 5. These *before* and *after* screenshots of the *Setting Up Projects* page illustrate that the application failed to detect the duplicate name "Doomed." Figure 4 walks you through the explanation of this time-related error that involves two users adding new project names to the same database.

As illustrated in Table 1, User A and User B create new projects simultaneously, but without knowledge of each other's actions. In step 3, User A adds a project named **Another**. Since that project name already exists, his browser's JavaScript displays a message prompting him for a different project name.

User B adds a project named **Doomed**. Her browser's JavaScript does not detect **Doomed** as a preexisting project name and so adds it to both the database and the returned list. The updated project name list is sent back to User B.

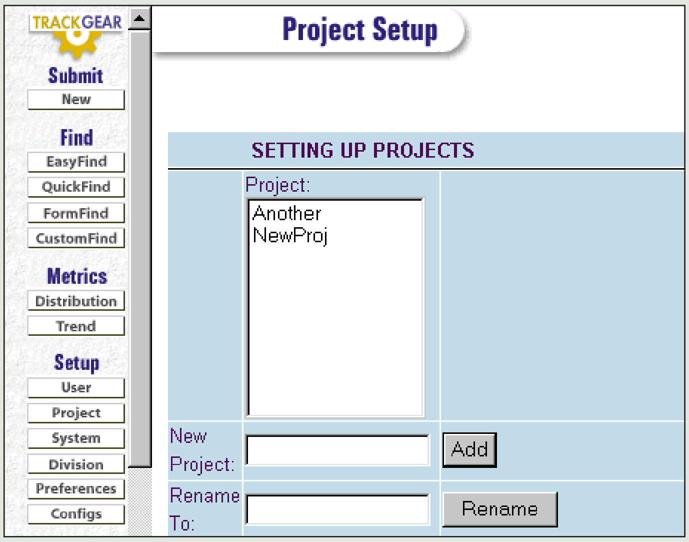
User A subsequently adds the same name, **Doomed**, to the project list. His browser's JavaScript does not detect the name on the HTML list, so it adds the name **Doomed** to the database again—as well as to the returned list. The updated project name list is sent back to User A with two **Doomed** entries included.

This result fails to meet the product's specification. Unless this situation happens to be a well-designed test case, accidentally discovering this error and attempting to reproduce it is not a simple task. In this example, the actual error is in the failure of the application to check for server-side duplicate names (in addition to client-side checking). The steps include User A's

```
...
<td width="80" bgcolor=#00CCCC>&nbsp;&nbsp;</td>
<td width="80" bgcolor=#00CCCC align="left" height="9">
<font size=1 face="Arial" color="#400040">
Project:<br></font>
<select name="namelist" size="9" OnChange="ListSelected()">
        <option value="Another">Another</option>
        <option value="NewProj">NewProj</option>
</select></td>
<td width="100" bgcolor=#00CCCC>&nbsp;&nbsp;</td>
...
```

FIGURE 4 **Browser-side JavaScript checks the input name against the *values* in the *namelist***

BEFORE



Project Setup

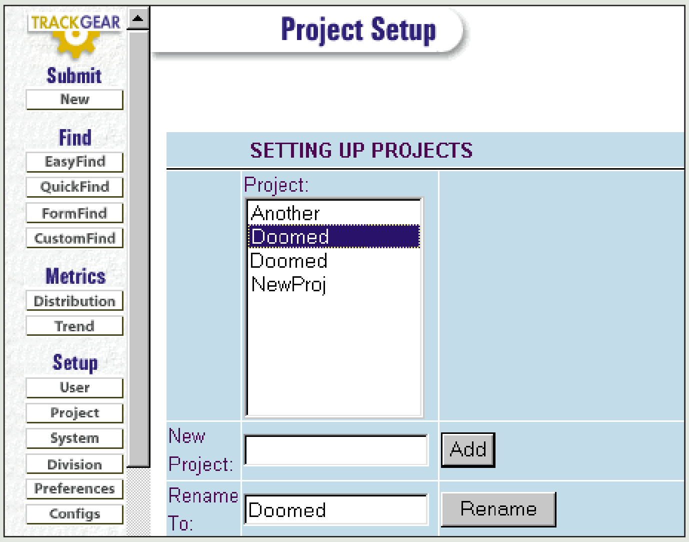
SETTING UP PROJECTS

Project:
Another
NewProj

New Project:

Rename To:

AFTER



Project Setup

SETTING UP PROJECTS

Project:
Another
Doomed
Doomed
NewProj

New Project:

Rename To: Doomed

FIGURE 5 **TOP: Before the project name "Doomed" has been entered by the user; BOTTOM: After the application failed to detect the duplicate project named "Doomed"**

activities. The dynamic operating environment is created by User B's activities—which are hidden or unknown to User A.

In Conclusion

To be effective in analyzing and reproducing errors in a Web environment, you need to have a command over the operating environment. You also need to understand how environment-specific variables may affect your ability to replicate errors. With the application of some of the skills covered in this article, I hope that your Web testing experience will be less frustrating and more enjoyable.

Remember that nothing will replace your testing skills—your ability to come up with good test cases, ask relevant *what-if* questions, keep careful notes, and methodically investigate hard-to-reproduce errors. It is these skills that will assist you in finding not only the errors that you are investigating, but also the undiscovered errors that are related to them. STQE

Hung Q. Nguyen (hungn@logigear.com) *is the president and CEO of LogiGear Corporation, a full-service consulting firm offering outsourced testing, QA training, and TRACKGEAR™ (a Web-based defect tracking solution). He is co-author of* Testing Computer Software *and author of the soon-to-be-published* Testing Web Applications.

STEP	REQUEST	HTML LIST BEFORE	HTML LIST AFTER	DATABASE BEFORE	DATABASE AFTER
1	USER A gets the Setting Up Project Page		Another NewProj	Another NewProj	Another NewProj
2	USER B gets the Setting Up Project Page		Another NewProj	Another NewProj	Another NewProj
3	USER A adds a new project named "Another"	Another NewProj	Another NewProj	Another NewProj	Another NewProj
4	USER B adds a new project named "Doomed"	Another NewProj	Another NewProj Doomed	Another NewProj	Another NewProj Doomed
5	USER A adds a new project named "Doomed"	Another NewProj	Another NewProj **Doomed** **Doomed**	Another NewProj Doomed	Another NewProj **Doomed** **Doomed**

TABLE 1 **User A and User B activities**