

Risk-Based E-Business Testing

Part 2 Test Techniques and Tools

Paul Gerrard
Systeme Evolutif Ltd.
9 Cavendish Place
London W1M 9DL
Tel: +44 (0)20 7636 6060
Fax: +44 (0)20 7636 6072
paulg@evolutif.co.uk
www.evolutif.co.uk

Abstract

This paper describes twenty techniques for testing E-Business applications. These techniques were introduced in the companion paper: "Risk-Based E-Business Testing, Part 1, Risks and Test Strategy", reference 1. The techniques are described under five categories: Static Testing, Test Browsing, Functional testing, Non-Functional Testing and Large Scale Integration testing. A chapter on Post-Deployment Monitoring is appended.

Static tests aim to identify statically detectable faults in web pages. Tools that can check the syntax of the HTML and other embedded code in web pages are now widely available. Internet browsers and versions implement different subsets and supersets of standard HTML but tools can identify anomalies in the HTML as well as make recommendations for correct usage. Between the leading browsers, there are differences in the appearance of web pages containing identical HTML that has been validated. These differences are often cosmetic, but can affect more than the appearance as objects may be pushed off screen because of different object sizing.

Test Browsing aims to ensure that web page navigation and integrated pages operate correctly. Link checking verifies that buttons or text links to other pages work. Object load and timing checks aim to ensure that objects can be loaded and are displayed with an acceptable delay. Transaction verification aims to ensure that server-based components are called correctly, with all parameters passed.

Functional testing covers what might be the 'traditional' test approaches to component, integration and system testing. Browser page testing covers the functionality that executes solely on the browser. CGI component testing covers all server-based components. Transaction testing verifies that the browser/user interface, web server and back end server-based components integrate as a whole. Application system testing covers the functional requirements for the system as a whole. Internationalisation covers the requirements for multilingualism, multicurrency and localisation.

Finally, it is becoming more important that web sites are monitored in production as performance, security and reliability may all falter but users may never report problems. Post-Deployment Monitoring may involve reuse of existing automated tests or one of the increasing number of remote web site testing service providers.

CONTENTS

1	Introduction	1
	1.1 Overview	1
	1.2 Scope	1
2	Static Testing	3
3	Test Browsing	9
	3.1 Test browsing	9
	3.2 Automated browsing	9
	3.3 What is an acceptable download speed?	10
	3.4 Transaction verification.....	11
4	Functional Testing	13
	4.1 Browser page testing.....	13
	4.2 CGI component testing.....	13
	4.3 Transaction Testing	16
	4.4 Application System Testing.....	17
	4.5 Internationalisation	19
5	Non-Functional Testing	20
	5.1 Configuration Testing	20
	5.2 Usability Testing	21
	5.3 Performance Testing	22
	5.4 Availability Testing	26
	5.5 Security Testing.....	27
	5.6 Well documented holes have counter-measures, but are YOU up to date?	28
6	Large Scale Integration	30
	6.1 End-to-End Functionality	31
7	Post-Deployment Monitoring.....	32
8	References	35

1 INTRODUCTION

1.1 Overview

This paper describes twenty techniques for testing E-Business applications. These techniques were introduced in the companion paper: “Risk-Based E-Business Testing, Part 1, Risks and Test Strategy”, reference 1. The techniques are described under five categories: Static Testing, Test Browsing, Functional testing, Non-Functional testing and Large Scale Integration. A chapter on Post-Deployment Monitoring is appended.

The Test process Framework Table from part 1 is reproduced overleaf.

1.2 Scope

The paper does not dwell on the types of testing that have direct counterparts in client/server or host-based systems.

This paper only covers systems that have been implemented in the most common technologies, namely:

- HTML static pages
- Embedded VBScript and JavaScript
- Server-based CGI programs, scripts or Active Server Pages.

It does not cover the testing of the following specific technologies:

- Java Applets.
- Java Servlets.
- ActiveX components.
- COM/DCOM server-based objects.

The paper does not cover the testing of large applications being implemented in, say, Java applets. In these cases, the applets themselves may be large applications. These should be component tested in the same way that non web GUI applications are. We suggest that Transaction testing and Application System Testing are as appropriate for Java applets as any other application implemented using Web technology.

Many of the test types described in the following section justify a paper of their own. Performance, security and usability testing are each significant topics requiring thorough treatment.

Most sections provide references for further reading.

One of the challenges of E-business testing is: where does developer testing stop and higher level testing begin? It is probably appropriate that the test techniques described in the early sections are done by developers rather than system or acceptance testers. Allocation of responsibility for more and less technical techniques is a major consideration for your test strategy.

Table 1 E-Business Test Process Framework

Test Type	Test Priorities					Test Types Mapped to Usual Test Stages				
	Smoke	Usability	Performance	Functionality	Static/ Dynamic	Desktop Development	Infrastructure Testing	System Testing	Integration Testing	Post- Deployment Monitoring
Static Testing										
Content Checking		Y			S	A/M				
HTML testing	Y				S	A/M				
Browser syntax compatibility	Y				S	A				
Visual browser validation		Y			D	M		M		M
Test Browsing										
Link checking	Y				D			A		A
Object load and timing		Y	Y		D			A		A
Transaction verification	Y				S	A/M		A/M		
Functional Testing										
Browser page testing	Y				D	A/M				
CGI component testing	Y				D		A/M			
Transaction Testing				Y	D			A/M		
Application System Testing				Y	D			A/M		
Internationalisation		Y			D	A/M		A/M		
Non-Functional Testing										
Configuration testing	Y				D	M		A/M	M	
Performance			Y		D		A	A		A
Soak Testing/reliability	Y				D	A	A	A	A	
Availability					D					A
Usability		Y			S/D			M		
Security				Y	D		A/M	A/M	A/M	A
Large Scale Integration										
External links/legacy system integration				Y	D		A/M		A/M	
End to end functionality	Y				D				A/M	A

2 STATIC TESTING

Static tests are those that do not involve executing software under test. Usually, static tests involve inspections and reviews of documentation, but can also cover static analysis of code using automated tools. In the case of web sites, we are most interested in automated methods for inspecting and validating HTML for syntax faults but also for compatibility with the various browsers. Although tools can validate the underlying HTML, we recommend that you also perform a visual check to ensure (mainly cosmetic) differences between browsers do not cause problems for users.

Tools that perform automated HTML validation and compatibility often perform other functions such as link, spelling and accessibility checks. These tools are now widely available, some are free, others can be obtained at a small cost. Some tool vendors bundle this functionality with their proprietary test tool suites.

There are four main areas of concern, that static testing aims to address.

Content checking

The content of Web pages, the accuracy, completeness, consistency, spelling as well as accessibility are no longer 'cosmetic'. It could be said that cosmetic does not equate to unimportant, as perhaps it might for internal IT systems. For many E-Business sites, content might be their 'raison d'être'. For example, if a Testing Services company spelt *kwality* wrong on their home page, this might not reflect well on the organisation. If content itself is the product for sale or is intended to be reused by customers, content faults might be deemed serious threats to success of the system and should be checked.

Most commercial web page development tools include spell checkers. Just like any documentation, however, pages should be independently reviewed to ensure they are consistent, grammatically correct and usable. This task might be included in a usability test.

HTML testing

Browsers download HTML pages from Web servers, interpret the HTML text and render the page within the browser window. In most ways, this is a similar process to software interpretation. However, unlike software interpreters, browsers are very forgiving of poor quality code. Invalid HTML syntax is usually ignored by the browser. In some ways, this is good. There are no unsightly and confusing syntax error messages displayed, ever. But if the programmer has got the HTML wrong, they may never achieve the layout effects or functionality they require. Web page developers and testers may not see problems, but users may experience obscure difficulties later.

Automated tools can validate HTML (or XML or WML) against rules laid down by established and emerging standards.

Over the page is the HTML Toolbox page of the NetMechanic web page checker (reference 4). This is typical of the web page test portals now available. As you can see, it can check the following:

- Links
- HTML syntax check (NetMechanic can also generate corrected HTML for you)
- HTML Browser compatibility
- HTML load times
- Spell check. (You can supply your own dictionary, rather than use the one supplied).

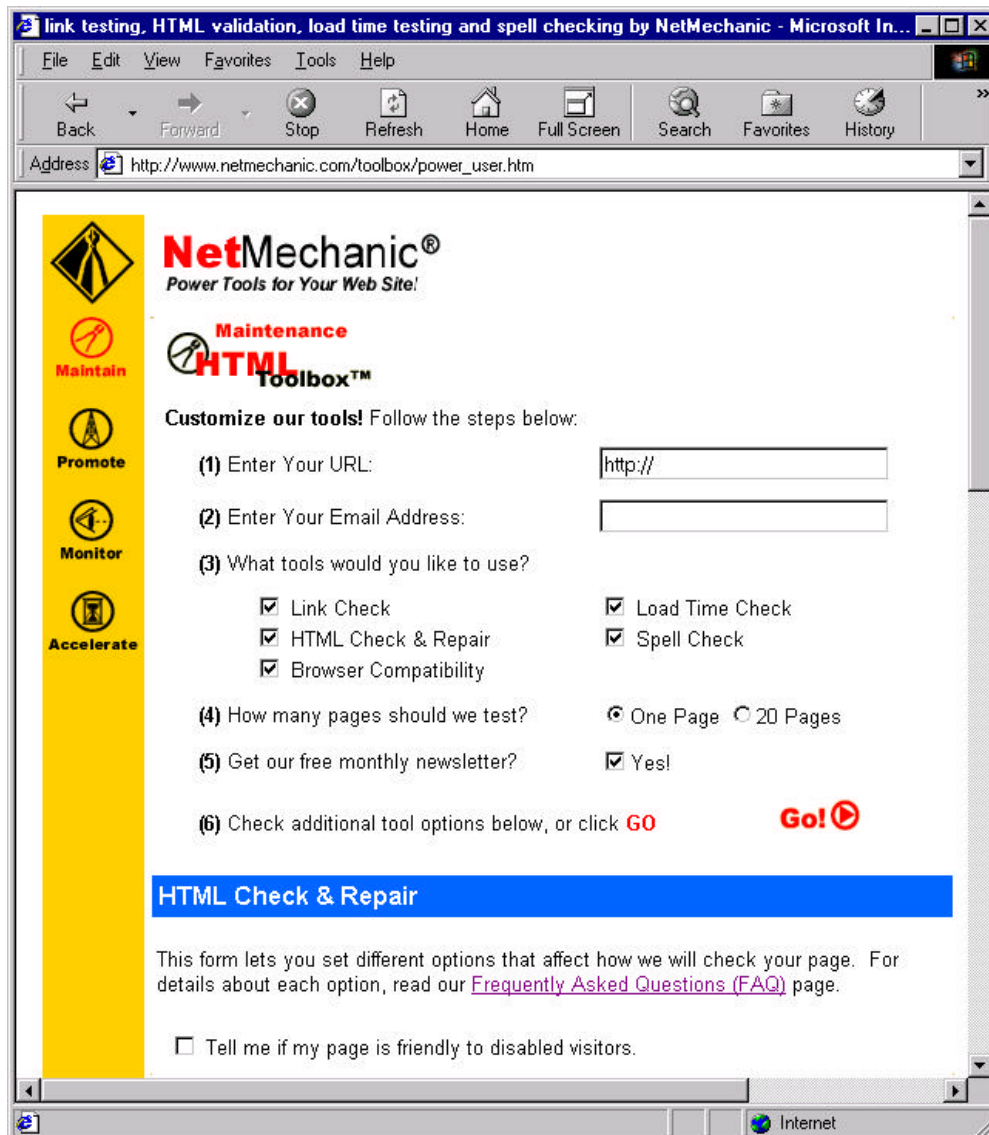


Figure 1 Typical Web page validation tool

In the report over the page, you can see that the HTML on lines 10 and 11 of the page have problems. The 'Click to Learn More' hyperlink brings up a small window that describes the fault that has been detected.

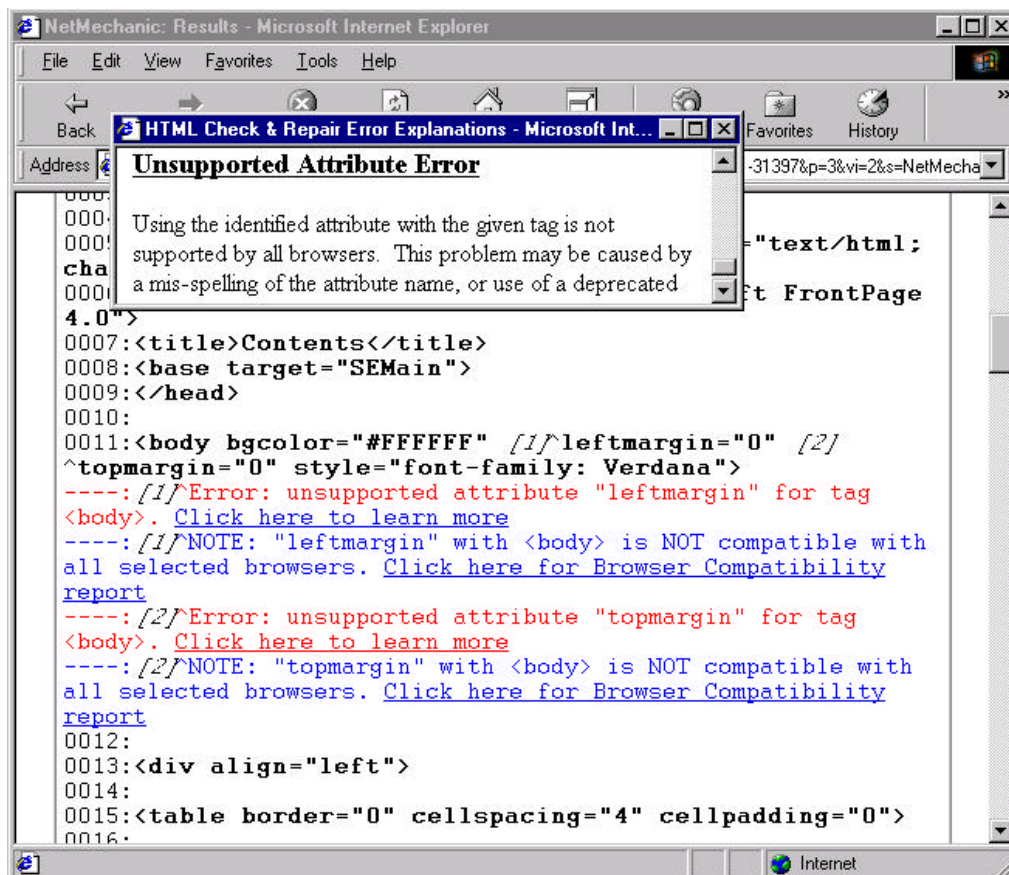


Figure 2 HTML validation report

Browser syntax compatibility

Although HTML is often promoted as a universal standard (Reference 27), browsers actually implement HTML differently. Some elements of HTML from previous standards are deprecated in HTML 4, but are still supported by these browsers as they are in common use. The event model (that defines behaviour for mouse clicks, button presses etc.) is far from complete in the current Standard, compared to what Visual Basic, for example, supports, so Microsoft, in particular, have implemented an enhanced event model for Internet Explorer.

Microsoft Internet Explorer (IE) and Netscape Navigator (Netscape) are clearly adopting different strategies:

- The IE strategy is to broadly support the HTML 4 standard, but Microsoft have introduced many additional elements that are supported by their browser and generated by their popular web page development tools (FrontPage and Visual Interdev).
- The Netscape strategy is to broadly support the HTML 4 standard, but Netscape have introduced a much smaller number of non-standard elements. It appears likely that Netscape will drop the deprecated features they have introduced.

A typical report on web page HTML compatibility appears in the figure below. You can see that the ALT attribute on an anchor tag (<A>) on line 139 of the web page is incorrect for both Explorer and Netscape. The LEFTMARGIN attribute of the BODY tag on line 10 is valid for all versions of Explorer and invalid for all versions of Netscape.

Tag	Attribute	Lines	Visitors Affected	Microsoft			Netscape		
				3	4	5	2	3	4
A	ALT	139	99.00 %	N	N	N	N	N	N
BODY	LEFTMARGIN	10	27.00 %	Y	Y	Y	N	N	N
BODY	TOPMARGIN	10	27.00 %	Y	Y	Y	N	N	N
FONT	FACE	14, 23, 25, 29, 33, 35, 37, 38, 39, 45, 46, 48, 51, 55, 56, 58, 63, 64, 66, 69, 72, 74, 75, 79, 81, 88, 88, 90, 91, 95, 95, 101, 101, 101, 104, 111, 112, 117, 131, 134, 139, 149, 156, 158	1.00 %	Y	Y	Y	N	Y	Y
IMG	</TD	149	99.00 %	N	N	N	N	N	N
TD	BGCOLOR	21, 54, 63, 101, 110, 134	1.00 %	Y	Y	Y	N	Y	Y

Figure 3 browser compatibility report

Setting aside the fact that the Microsoft strategy for browser domination appears to be working, in the interim, the existence of these two strategies causes difficulties. To support both browsers simultaneously, there are three approaches:

1. Adopt a reduced set of HTML elements within the HTML standard, that are supported by both browsers and restrict yourself to those.
2. Build parallel HTML implementations for each browser and determine which should be used at runtime. (Potentially complicated and more costly)
3. Adopt one browser as standard and ignore all others. (Only viable if you are building an intranet and have control over your users' technology platforms).

Page authoring products usually have configuration options to allow HTML that meets one of these following compatibility requirements:

- IE only
- Netscape only
- Both IE and Netscape
- Web TV
- An so on...

Once a strategy has been selected, you must agree with the developers whether (and how) they are adopting this strategy. Then you must use the available tools in a consistent way to report faults.

Visual browser validation

It should be obvious from the previous discussion that there will be significant differences in the appearance of web pages displayed by the different browsers, but also in their behaviour. Take for example the two identically sized screen dumps of a previous version of the Evolutif Web site. One is from IE4 and the other Netscape 4. (I must admit, I cannot recall which is which!).

Note that the last field displayed on the first page, displayed in Figure 4, is 'Phone' – and the field itself is almost completely cut off by the border of the page. In the second example, displayed in Figure 5, the last field is 'Fax'. In this case, an extra field is displayed by the browser.

Figure 4 Web page display, first example

Figure 5 Web page display, second example

Clearly, the differences here are cosmetic, but if your intention is to avoid scrolling of pages in a set resolution setting, then this problem might force a clickable button off the page and be less usable. Other, more serious problems have been reported, but are most commonly associated with differences in:

- The colours of standard objects e.g. horizontal lines, borders etc.

- The centering and scaling of objects in windows
- Display of scrolling marquees
- Table layouts - automatic sizing gives different results
- Table background and border colours differ

It would be possible for a cross-browser automated tool to be used to scan pages for examples of page objects being displayed differently, but we strongly recommend that you adopt a more pragmatic approach. Consider testing under one platform and verifying visually, that the appearance of pages on the other browsers are acceptable. We recommend that your developers (and testers) do not all use the same browser and version. Let them use their favourite browser or ask them to use a selection of browsers and versions, to avoid compatibility testing.

3 TEST BROWSING

Test browsing aims to address those faults that relate to the navigation through web pages, the availability of linked objects and the speed of download. It also covers the integration of web pages to server-based components, to ensure that the right component is called with the correct parameters.

3.1 Test browsing

If we were to focus on traversing links and opening pages, when we open a new page:

- Can the page be downloaded and displayed?
- Do all the objects on a page load?
- Do all the objects on a page load in an acceptable time?
- If the user turns images off, uses a non-graphical or no-frames browser, does it still work?
- Do all the text and graphical links work?

One of the most common problems with web sites relate to validity of on-site or off-site links. On-site links should load pages or objects from the same site of course, but many sites, especially search engines or directories have many, many links to off-site pages, normally other sites.

3.2 Automated browsing

There are many tools that can perform automated browsing tasks. Automated browsing covers the first three tasks, link checking, object loading and timing. Checking that pages work when users turn graphics off or use of no-frames browsers must be done manually.

Link checkers read a starting page and identify all the linked objects on that page. These can be any of the following:

- Linked pages (other pages to be navigated to by clicking on hyperlinks).
- Frame pages (where a page is partitioned into frames and each frame has its own HTML page to create the image displayed in the browser window).
- Images used for graphical appearance or as buttons to navigate (e.g. GIFs and JPEGs)
- Form handlers, where these are CGI scripts, Active Server Pages etc.
- ActiveX, Java applets and other objects that are downloaded and executed within the Browser.
- Other content files, such as video (AVI, MPEG), and audio (WAV, AU, MIDI, MPEG) files.
- Other Internet protocols such as email links, FTP, Newsgroups links.

Typically the tools that do link checking perform two functions. Firstly they identify all of the linked objects on a page and then attempt to download them. In doing so, the robots determine whether the object exists and can be loaded, and the size and download time for the object.

Typical reports list objects linked, validity and size. Some tools provide download time for each object-based on a selected network connection speeds. Nowadays, a typical modem connects at 33kbps. This is a reasonable assumption for most business-consumer applications. For business-business applications, end users might be connected at much higher speeds using their corporate LAN.

These tools work either as a portal-based services, where you request a Web robot or agent to visit your site on the web and traverse all of the links and objects and then produce a report. A typical report from the NetMechanic tool is displayed below:

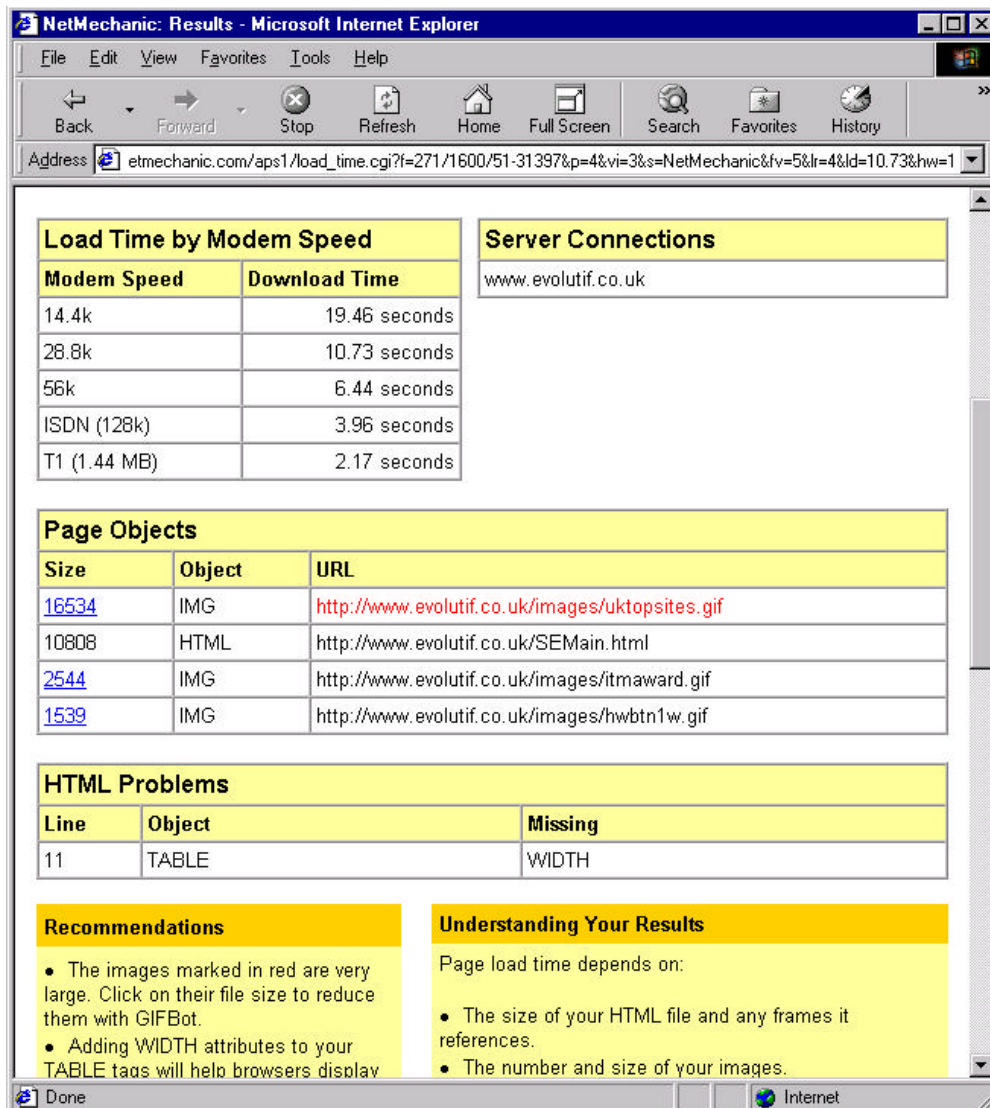


Figure 6 Report from load time checker

Some tools provide graphical output of the links for a site with the home page at the centre of such graphics. These diagrams normally resemble a collection of connected spidery shapes. This is less useful for large sites (although the diagrams can be pretty!)

The majority of Web page development tools e.g. Microsoft FrontPage offer link checking facilities. In effect, it is simply the object file size and connection speed that determine the download speed so these tools can usually report broken links and slow download speeds.

3.3 What is an acceptable download speed?

Firstly, you need to consider who your users are. In general, object sizes are not critical for users who connect using a 2 Mbps or T1 line (1.544Mbps) direct to the net or to our local intranet. You must bear in mind that home-based and small office-based users may be accessing the Web through 28k or 56k modems. You remember accessing the web using your 9600bps modem, don't you? In those days, you probably turned graphics off entirely. Nowadays, the standard modem speed is around 33k. this is what you should use for assessing the speed of downloads.

Until new technologies like ADSL are established, the size of pages will remain a big issue for consumer users.

The main concern with speed of page loading is page design, not the speed of our infrastructure and the net itself. Normally, if page graphics are not the main content we are offering on a web site, the images used to illustrate our site and provide graphical buttons for navigation and transactions are our main concern.

Normal recommendations for good (and fast) page design are:

- HTML IMG tags have WIDTH and HEIGHT attributes to define image size explicitly.
- The home page should be less than 30k in size total.
- All other pages less than 45k.
- Background and button/icon pictures should be less than 5k.
- If you must use an image for the page background, use the same image for all pages.
- Photos are JPEG format, computer generated images/schematics should be GIFs.
- All GIFs should be interlaced.
- Images should not be duplicated – use the same images repeatedly to avoid reloads.
- Limit table text size to 2k.
- Size tables explicitly rather than use browser autoformat.

3.4 Transaction verification

The most common mechanism used to implement functionality on web sites is the Common Gateway interface or CGI. When an HTML form is filled in and the form submitted using one of the form buttons, the form handler referenced by the form definition is invoked on the server. The form itself sends the data captured on visible fields and the preset information defined on invisible fields as parameters to the server-based form handler.

Server-based forms handlers are CGI programs. On Unix systems these are Perl scripts or executable programs written in C++ perhaps. In the Windows environment, these programs can be C++ programs, DLLs or Active Server Pages.

What happens is that the forms handler extracts the parameters supplied in the message sent by the browser and uses this data to perform the server-based transaction. Typically, forms handlers execute query or update transactions on a database and then generate a new page to be posted back to the client browser.

Transaction verification aims at ensuring that firstly, the correct forms handler is invoked and that the parameters passed to the forms handler are correct.

Part of the forms definition is the method by which the HTTP message is sent to the server. There are two methods used: GET and POST.

HTTP GET

When a CGI program is invoked by an HTTP GET message, the parameters are appended to the URL of the forms handler. For example, a form having two fields, reptype and sortorder, that generates a report would generate a URL such as:

```
http://www.mysite.com/cgi-bin/doreport.pl?reptype=ProdList&sortorder=ByPrice
```

In this case, the program `doreport.pl`, in directory `cgi-bin` on the web server would run with two parameters, `reptype` is "ProdList" and the `sortorder` is "ByPrice". The full URL above, including the parameter values is displayed in the URL window of the browser.

HTTP POST

When a CGI program is invoked by an HTTP POST message, the parameters are incorporated into the body of the HTTP message (not in the URL, like GETs). The browser does not display the form parameters appended to the URL in the URL window. The forms handler obtains these values from either environment variables or the standard input to the program. In this case, in order to verify that the parameters are passed correctly, the programmer would need to incorporate a small amount of code to write out the parameters passed to the forms handler program. This code would be commented out when any anomalies are fixed and the code is working correctly.

4 FUNCTIONAL TESTING

Functional testing has been treated as a staged, bottom-up approach. Components that execute entirely within the browser are tested separately from the server-based components. Transaction testing addresses the integration of the browser pages, web server and other server-based components. Application System Testing addresses the traditional requirements for a complete functional system test. Internationalisation checks cover the particular requirements relating to address formats, multi-currency and tax arrangements.

4.1 Browser page testing

Browser page tests cover the objects that execute within the browser, but do not exercise the server-based components. These are typically:

- JavaScript/VBScript code within HTML pages that:
 - Implement special effects, such as changes to the appearance of buttons as the mouse pointer rolls over the button.
 - Implement field validation.
 - Open new windows, control the sizing and positioning of windows etc.
- (Usually Java) applets that implement screen functionality or graphical output.

These features of the web pages should be tested in-situ using the browser. These features are most easily tested against checklists of 'conventional' requirements or the specific requirements for particular pages.

NB: Some systems implement a large amount of functionality in Java applets; some entire applications are implemented in applets. In these cases, test them like any other GUI application.

For advice on testing GUI applications, see reference 8.

4.2 CGI component testing

CGI component testing covers the objects that execute on the server, but are initiated by forms-based user interactions on the browser. These are typically:

- Business logic to do database enquiry or updates.
- Product catalogue searches.
- Order processing.
- Credit checking and payment processing.
- Security checking and validation.

Usually, the server-based forms handlers are written at the same time as the forms that invoke them. However, we have found two problems with testing server-based code via the user interface:

- Firstly, in some projects, the user interface is actually the last thing to be written so server-based components cannot be tested completely until very late in the project.
- Secondly, it is common for server-based components to be programmed to deal with a very broad range of input parameters, invoking complex transactions that interface with legacy systems. Large numbers of tests may be difficult to implement manually so an automated alternative can be more effective and economic.

We advocate the use of dummy web pages or test drivers to test server-based components. In this way the manual effort can be leveraged dramatically. Consider Figure 7. On the left hand side is a window displaying a dummy page. Clicking on one of the links on the left hand window sends the HTTP message to the web server and displays the results in the page on the right (actually, the window name is “_test”). When a new link is clicked, the window on the right displays the results of the transaction initiated.

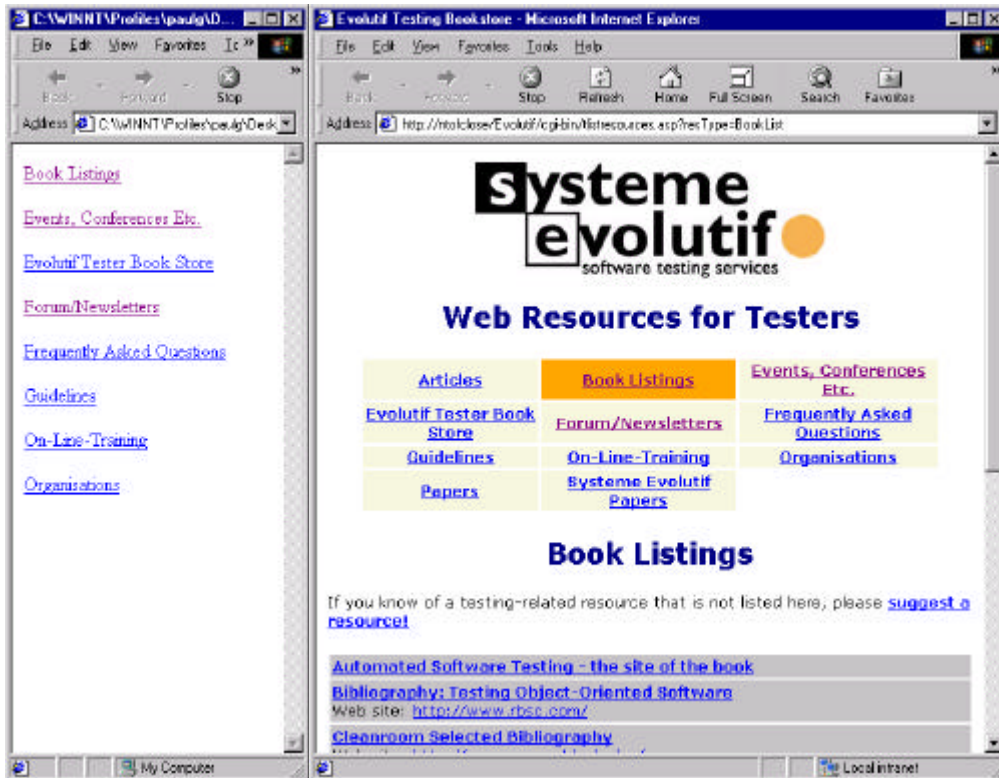


Figure 7 Using a dummy page to test server-based code

The HTML source code for the dummy page on the left is presented in Figure 8 below. The anchor tags (the <A> pairs) implement HTTP GETs of the URL identified in the HREF attribute. Note that the display of the full URL in the URL field of the browser window on the right.

```

<a href="http://ntofclose/Evolutif/cgi-bin/tlistresources.asp?resType=BookList"
target="_test">Book Listings</a><p>
<a href="http://ntofclose/Evolutif/cgi-bin/tlistresources.asp?resType=Event"
target="_test">Events, Conferences Etc.</a><p>
<a href="http://ntofclose/Evolutif/cgi-bin/tlistbooks.asp?resType=BookStore"
target="_test">Evolutif Tester Book Store</a><p>
<a href="http://ntofclose/Evolutif/cgi-bin/tlistresources.asp?resType=Forum"
target="_test">Forum/Newsletters</a><p>
<a href="http://ntofclose/Evolutif/cgi-bin/tlistresources.asp?resType=FAQ"
target="_test">Frequently Asked Questions</a><p>
<a href="http://ntofclose/Evolutif/cgi-bin/tlistresources.asp?resType=Guideline"
target="_test">Guidelines</a><p>
<a href="http://ntofclose/Evolutif/cgi-bin/tlistresources.asp?resType=OnLineTraining"
target="_test">On-Line-Training</a><p>
<a href="http://ntofclose/Evolutif/cgi-bin/tlistresources.asp?resType=Organisation"
target="_test">Organisations</a>

```

Figure 8 HTML source code in the test driver

If you look closely at the first line of HTML above, this generates the underlined link labelled “Book Listings” in the left hand window. When you click on this link, the browser sends an HTTP GET message, referencing the following URL:

<http://ntofclose/Evolutif/cgi-bin/tlistresources.asp?resType=BookList>

The resulting page generated by the ASP file is displayed in the window on the right. The internal name for this window is “_test” (see the “target=” HTML).

It is easy to click on each link in turn and verify that the correct page listing appears in the window on the right. Consider how you might generate this HTML from a simple database containing just two columns: URL and parameter resType. We have used various techniques including mail merge and writing Visual Basic routines within an Access database to generate simple test drivers using this approach.

Alternative techniques involve generation of dummy forms – many, many forms can be displayed serially in the same web page. Each form contains the fields that would have been supplied using the real web page in the application, but is generated by a database. With forms you can implement both HTTP GETs and POSTS, and in this way, simulate both types of transactions in a realistic, but simple way. There are some issues relating to the set up of cookies and potentially application and Session variables on the web server, but your developers should be able to create simple server-based scripts to ensure these are correctly set up. This is a more advanced topic, outside the scope of this paper.

Clearly, to implement such facilities you need some detailed knowledge of HTML and the way forms, HTTP GETs and POSTs work. However, if you can get the help of your developers or bring a developer into the test team, it should be a simple task to build some very effective test drivers for your server-based code.

In our own projects, Evolutif have built simple automated test drivers using Visual Basic and Access databases. In these cases, we used the Visual Basic INET object to simulate HTTP requests and retrieve web pages and execute server-based code. In this way, we have run functional tests that execute 40,000 transactions in a single test run.

Creating and running a manual test of 40,000 transactions is impractical. Creating and running such large tests using proprietary test execution tools that drive the browsers would also pose a significant challenge. There is a clear choice in this area:

Proprietary tools

Proprietary tools driving the user interface are expensive, easy to use, generate complex GUI scripts and force the tester to navigate slow, complex application scenarios.

Home grown tools/test drivers

Home-grown tools driving server-based code via the API can be cheap, easy to use, generate simple HTTP oriented scripts and eliminate slow navigation through complex application scenarios. They are less good at verifying actual results against expected results.

We strongly suggest that you investigate the use of test drivers for at least the higher volume test transactions as it may save you a lot of time. Further, by involving the developers, you may find that the burden of functionally testing server-based components may shift towards the developers (a good thing!). System testers can perhaps re-use the developers’ test drivers to explore more complex and higher volume scenarios.

4.3 Transaction Testing

Transaction testing aims to address the problem of integration of the complete end-to-end functionality to ensure that the entire transaction is processed correctly, from the user action on the browser interface through to back-end systems. Transaction testing is not the same as Application System Testing. Rather it focuses very much on selected test cases that exercise particular interfaces between components in the technical architecture and scenarios (in volumes) that System tests might not address.

Typical tests here would be selected transactions that invoke several components and interfaces that must collaborate to deliver a piece of functionality. Transaction Testing maps to traditional integration testing in the small, where a member of the programming team conducts tests to ensure the products of more than one programmer integrate, before submitting integrated components for system test.

A transaction that processes the payment for a book, bought online might involve the following:

- A button click on the user interface prompting validation of the fields on the screen.
- Dispatch of an HTTP message to the web server, with visible and hidden fields on the user screen being transmitted.
- Invocation of a CGI program on the web server that sends requests to various objects such as:
 - Credit checking object.
 - Order posting object.
 - Database update object.
 - Special offers and order confirmation email generator object.
 - Thank-you web page generator object.
- The response of the CGI program would be a 'confirmation' page with options to place more orders or log off.

The objects called by the CGI program may each have been tested in isolation. Often, they have been tested as they were coded with stubbed out interfaces and simple driver programs or dummy web pages to exercise them in artificial circumstances.

Transaction tests are designed to force the software to invoke the various components as a complete set and to investigate whether the direct and indirect interfaces work correctly. These test cases cover the following main concerns:

- Transfer of control between components.
- Transfer of data between components (in both directions)
- Consistency of use of data across components.

Typical test cases involve:

- Single shot transactions that pass zero, 1, many or a very large number of records.
- Reconciliation of pre-test database content, data entered through screens, system outputs and post-test database content.

Overall, Transaction tests aim to cover the complete end-to-end functionality within a system including the browser web pages, web server-based objects, other server-based objects, back-end databases and legacy systems.

Often, the interactions between components are complex or there is a significant amount of custom-built code in the server-based objects or there is custom-built middleware involved. In these circumstances, it is prudent to automate some of these transactions to ensure that these interfaces and custom code can support repeated use for an extended period. Repeated transactions driven by a test execution tool (or possible re-use of drivers mentioned above) can expose obscure memory leaks or synchronisation problems.

4.4 Application System Testing

Application System Testing aims to cover complete testing of the application with business oriented scenarios to ensure all features of the system meet their requirements. In this regard, AST is simply System Testing for E-Business systems.

Each feature of the system is identified in turn and test conditions selected to exercise the key functionality. Test scripts might be oriented towards running all of the tests for each feature repeatedly and working one's way through all features, but some percentage (the higher the better) of tests should follow business scenarios. There is very little more to say about this that is unique to E-Business systems. However, there are two particular points that are worth emphasising.

- Applications can be navigated using the browser using the navigation buttons (these are not under the control of the application).
- The Web is 'stateless'. There is no persistence between transactions unless developers implement certain mechanisms.

Browsers allow the use of back, forward and refresh buttons. These buttons may cause server-based transactions to be repeated, perhaps many times over. How do these affect the use of your application? Is it sensible to rerun the same query twice? Possibly. Is it sensible to execute the same payment transaction twice? Certainly not! How will the application deal with these situations if a user, unwisely, chooses to do so?

The Web is 'stateless'. That is, each HTTP message received by a Web server is unique and unless certain mechanisms are adopted, applications cannot maintain the context of business transactions. What this means is that a web server cannot link a transaction to add products to a shopping cart to a later transaction that pays for the goods unless one of several mechanisms are used. How are the developers designing their applications to maintain the context of business transactions?

Cookies

Because there is no persistence between CGI transactions, the CGI mechanism allows the use of cookies. Cookies are small amounts of data that may be stored on the user's hard drive by the browser at the request of the Web site being accessed. When the user clicks a button and sends a new HTTP request to the web server to process a transaction, the cookie data is embedded in the HTTP message and the server can extract this cookie data for its own purposes.

Typically, cookies contain personalisation data such as the user's name, the date of their last visit and so on. Often, cookies contain a unique reference to identify a logged-in user session. All transactions posted to the web server during the session are labelled with this session identifier, so the server-based code can 'remember' who they are dealing with. Used sensibly, cookies are harmless enough.

However, the programmer has total control over cookies and could put more sensitive data such as passwords, credit card numbers and more personal information into them. Because of

this, browsers offer users the opportunity to be warned of cookies being sent and also to reject them. If a user rejects a cookie, will your application still work? Or will the transactions fall apart because there is no way to string them together?

There are also some limitations on the use of cookies and these are prime candidates for test cases. See reference 10 for a thorough description of the CGI and cookie mechanisms.

Hidden fields

The most common alternatives to cookies are hidden fields. When a user completes an HTML form displayed on a web page, the content of visible fields that the user can enter are appended to the HTTP message sent to the web server. However, there is a facility in HTML to include hidden fields within the form that have pre-set values. These values are also sent to the server at the same time, in the same message. Programmers use this facility to include context information, such as a transaction identifier in the forms that are returned to the browser. Because the hidden fields are invisible, the user never notices that data items like these accompany the data that they are entering. It is a simple mechanism, but very effective. There is of course a security hole here – you can view the HTML source code in the browser, and change it to suit yourself and potentially affect the behaviour of the system under test.

It is essential that you discuss with your developers how they get round the stateless nature of the web and how this might be subverted. This will lead you to a set of tests that could expose faults in the design of the application.

Cookie Testing

Cookies have an expiration date. That is, the programmer sets a date on which the cookie will be automatically deleted from disk by the browser. Ask the developers how they are using cookies and how they set expiration dates. If these dates could be exceeded, consider deleting them in between tests, or setting client system clocks to force them to expire. What happens in the application?

Cookies have maximum size of 4k each, there is a limit of 300 cookies on a client and 20 cookies for a single domain – are your developers exceeding these limits?

Loss of connection

The context of user transactions can be lost through other means. The first, most obvious one is a loss of connection between the client machine and the web. Users with dial-up modems are used to connections timing out or failing. How does your application cope with this? When you lose the connection and/or close your browser completely, cookies having no expiration date set will be lost. Can (should) the user retrace their steps in the application using the history facility in the browser?

Other situations to consider

What happens if a new user of the same client PC uses your application on the same domain? Will they pick up the cookies of the previous user? Will the new user's cookies overwrite the previous user's cookies?

Ask your developers if or how they can accommodate the above scenarios. If they have a means of recovery or avoidance of problems, you should test it. If not, you may have exposed a problem in their design.

4.5 Internationalisation

Is your site intended to be used by, say, English speaking users only? If not, what steps are being taken to make your site multilingual? In principle, localisation testing is all about verifying that all user messages, prompts and output is translated correctly and that the functionality delivered to the end user is identical.

In this case, there may be scope for automating a set of regression tests of the functionality and parameterising the scripts with alternate translations of system input and output. However, if the application has been developed to reuse tables of alternative messages, then it is probably easier to manually inspect the language-specific text tables rather than regression test functionality that does not change across languages.

In our experience, many sites adopt the rather heavy handed technique of hosting multiple copies of translated HTML pages on the same server, and you choose your own language-specific web site. These need more testing and configuration management is probably a bigger headache.

Functional differences between localised web sites should consider at least the following:

- Local variations in tax arrangements, e.g. Value-Added-Tax, purchase tax etc. (in the US, different states have different tax arrangements of course).
- Address formats: in the UK we have postcodes, in France there are departments, in the US there are Zip codes.
- Foreign character sets can be a potential problem. You should recognise that in foreign countries, users may select a different character set than English (British or US). Russian, Greek, Japanese character sets are dramatically different, aren't they? Will your validation or database settings allow for this?

There are many, many potential pitfalls in international Web sites. You must discuss with your developers and marketers how best to handle these problems.

5 NON-FUNCTIONAL TESTING

5.1 Configuration Testing

Configuration testing aims to demonstrate that your Web application will operate correctly on your nominated range of client hardware, operating system and browsers software combinations. On an Internet, you have no control over the end users' platform, so to avoid problems later, it is best to test a range of configurations to ensure that you can support at least the most common combinations.

The problem of course, is that there are virtually an unlimited numbers of configurations available:

O/S Platforms	DOS, Windows 3.1, 95, 98, NT 3.51, NT 4, Macintosh, Unix.
Network connections	Dial-up modems, direct Internet lines, ADSL, Wireless.
Commercial services	MSN, AOL, Freeserve and many others.
Browsers	Internet Explorer, Netscape, and many others, too many to mention...

The problem of test scope

- Which configurations will you support? (Not all will be worth considering).
- Which can you test economically?
- Which can you test automatically?

If you intend to perform configuration testing on more than a handful of platforms, the first task is to analyse your market. What kind of hardware and software are your users most likely to have? One might reasonably expect your analysis to result in a small subset of what is available:

- MSIE and Netscape on Windows 98 and Windows 2000 might cover the majority of business-to-business clients.

But... what about Unix and Macintosh users? Microsoft NT and windows 95 are likely to exist for some years yet, aren't they?

If you are to look seriously at configuration testing then you must also look at component commonality. Is there any reason to doubt that one browser version will work differently on two operating system versions? If there is, what is the potential impact? Are you using any of the O/S specific features?

There are two discussions to have:

- With the marketers: what are the most likely platform combinations to be used?
- With the developers: are there any platform specific features being used that make configuration testing essential?

In our experience, only the largest, most mission critical projects take configuration testing so seriously that they test more than a handful of configurations.

We haven't the time. Is there an Alternative?

We recommend that if you are concerned with only the two leading browsers, IE and Netscape then you consider the following alternative:

- Identify browser compatibility problems using the static analysis tools that are now widely available. To the degree that they identify HTML features that are non-standard and not compatible to one or more browser versions they are probably more effective than dynamic testing.
- Encourage your developers and testers to use a variety of browsers and versions. In this way, one would expect that in the day-to-day operation of the Web application, compatibility problems would be detected during development or other testing.

If you are only concerned about the two leading browsers, anything that gets past these two hurdles is likely to be so obscure, that few users would ever be affected and there's no guarantee that you would have found the problem anyway.

5.2 Usability Testing

Usability and usability testing have never been as important as now. Typical Web site users have low boredom, frustration, inconvenience and insecurity thresholds. Home-based users may never have been trained in the use of a computer, let alone browsers and your web applications.

Regardless of sophistication, if the your web application doesn't allow the user to make enquiries and place orders easily, quickly and reliably, the site will fail. If the user cannot easily understand from your web site how to proceed, the user will leave your site and go elsewhere.

Think about this. In terms of difficulties for users, usability is probably the first problem that they experience, usability is probably a more important requirement than your site's functionality.

So the pressure to make the user experience as easy and fast as possible is considerable.

The problem with usability is the perceived difficulty in expressing requirements. However objective requirements can be defined:

- Messages to users will be in plain English.
- Commands, prompts, messages will have consistent meanings.
- Explanations are meaningful and relevant.
- The user must always know what state the system is in.
- Noticeable and informative feedback.
- The system will help (not hinder) the user:
 - don't need to confirm limited choice entries
 - system must provide defaults when applicable
 - must not prompt for data it does not need
 - must only display informational messages as requested by the user.

These requirements can be positively identified and measured.

Web conventions matter

Web page authoring guidelines provide checklists for Web conventions. These guidelines are generally accepted good practices. In principle, all web sites should consider committing to following them to ensure their web sites are accessible to all users. The best thing about such guidelines is that they can be used as objective requirements. When you test, failure to adhere to these guidelines can be raised as incidents.

Jakob Nielsen's book (reference 12) is a thorough treatment of good and bad practices in web design with lots of examples of both. (As a matter of interest, Vincent Flanders' www.websitesthatsuck.com page is both entertaining and educational.)

Heuristic evaluation

Heuristic evaluation is a sophisticated sounding name for a very straightforward technique. Reference 18 gives a very clear description of this popular approach. The process is very similar to a peer review:

- Select your evaluators to conduct private assessment.
- Compare the user interface to good practices (the heuristics).
- Raise observations of poor usability, frustrations etc.
- Evaluators may be asked to follow scripted scenarios e.g. make an enquiry, place an order etc.
- Convene a meeting to review the issues raised and discuss possible improvements. These meetings follow a 'focus group' approach. They are semi-structured discussions that are documented.

A thorough treatment of usability design and assessment approaches can be found in reference 20.

5.3 Performance Testing

Performance and associated issues such as resilience and reliability appear to dominate many people's thinking when it comes to non-functional testing. Certainly, everyone has used web sites that were slow to respond, and there have been many reports of sites that failed because large numbers of people visited sites simultaneously. In these cases, failures occur because applications are undersized, badly designed, unoptimised and inadequately tested.

In some respects, performance testing is very easy. All that is required is to simulate a number of users doing typical transactions and there are many tools available to do this. Simultaneously, some test transactions are used to measure response times, as a user would experience them. There are a large number of tools available to execute test transactions and take measurements. In practice, with a slow site, you can do this task manually, using with a stopwatch to take response times.

What's all the fuss about – it sounds simple.

In this section, I'm not going to provide a detailed description of how performance tests are planned, constructed, executed and analysed. I refer you to reference 7, which provides a broad description of client/server performance testing. In this section, I'll discuss some key points and some issues that distinguish Web performance testing from 'traditional' client/server.

Scope and Objectives

In principle, performance testing aims to demonstrate that:

- The system functions to specification with
- acceptable response times while
- processing the required transaction volumes on
- a production sized database.

Performance testing can be very expensive so we normally advocate setting other objectives to be achieved using an automated test framework and we normally include the following aspects, in defining test objectives:

- To assess system's capacity for growth – to determine by what margin the performance of a system meets requirements and speculate by how much loads can increase before response times become unacceptable.
- Stress tests to identify weak points – to subject the system to excessive or extreme loads to find the weakest components, so they can be tuned, upgraded and the system made more resilient.
- Soak, concurrency tests over extended periods to find obscure bugs – to exercise the system over 24 or 48 hour period to ensure that the system can be operated successfully for an extended period.
- Test bed to tune architectural components – we can reuse tests to progressively measure the effects of tuning or upgrade activities on system performance to ensure we have a fully optimised system.

Pre-requisites for performance testing

We normally specify the following pre-requisites before performance tests can be executed and the results produced deemed to be reliable. These are all explained in the paper referenced above, but there are particular difficulties in E-Business environments.

- Quantitative, relevant, measurable, realistic, achievable requirements.
- Stable software system.
- Actual or comparable production hardware.
- Controlled test environment.
- Tools (test data, test running, monitoring, analysis and reporting).
- Process.

Load requirements may be suspect

The first problem is that of workload requirements. With an internal system, there is usually a known, upper limit to the load it will experience. Intranets normally have a finite (and known) user base so it should be possible to predict a workload to simulate. With Internets, however, there is no reasonable limit to how many users *could* browse and load your website. The calculations are primarily based on the success of marketing campaigns, word of mouth recommendations and in many cases, luck.

In these circumstances, it is perhaps better to think of performance testing, less as a test with a defined target load, but as a measurement exercise to see how far the system can be loaded before selected response times become unacceptable.

In our experience of dealing with both established firms and dotcoms, the predicted growth of business processed on their web sites is grossly overestimated. For the purpose of acquiring venture capital and attention in the market, this is the game that is played. For performance testing, it can dramatically increase the cost of testing. Ambitious targets for on-line business volumes require expensive test software licenses, more test hardware, more network infrastructure and more time to plan, prepare and execute.

Decreased timescales may be a problem

The second obvious problem with E-Business performance testing is the time allowed to plan, prepare, execute and analyse performance tests. How long a period exists between system testing eliminating all but the trivial faults and delivery into production? We normally budget 6-8 weeks to prepare a test on a medium to high complexity environment. How much time will you have?

In E-Business projects, however, there can be some simplifying factors that make it easier to test Web applications:

- Firstly, web applications are relatively simple from the point of view of thin clients sending messages to servers – the scripts required to simulate user activity can be very simple so reasonably realistic tests can be constructed quickly.
- Because the HTTP calls to server-based objects and web pages are simple, they are much less affected by functional changes that correct faults during development and system testing. Consequently, work on creation of performance test scripts can often be started earlier than traditional client/server applications.

Tests using drivers only (and not browsers) may be adequate

Normally, we subject the system under test to load using drivers that simulate real users by sending messages across the network to servers, just like normal clients would. In the internet environment, the time taken by a browser to render a web page and present it to an end user may be a small time compared with the time taken for a system to receive the HTTP message, perform a transaction and dispatch a response to a client machine. It may be a reasonable compromise to ignore the time taken by browsers to render web pages and present them and just use the response times measured by the performance test drivers to reflect what a user would experience.

Scripting performance test tool drivers is relatively simple. Scripting GUI oriented transactions to drive the browser interface can be much more complicated and the synchronisation between test tool and browser is not always reliable.

If you do decide to ignore the time taken by browsers themselves, be sure to discuss this with your users, technical architect and developers to ensure they understand the compromise being made. If they deem this approach unacceptable, advise them of the delay that additional GUI scripting might introduce in the project.

Which performance test architecture?

There are three options for conducting performance testing. All require automated test tools. The pros and cons of each are summarised in Table 2 Three Performance Test architectures Compared. To implement a realistic performance test you need the following:

- Load generation tool.
- Load generation tool host machine(s).
- High capacity network connection for remote users.
- Test environment with fully configured system, production data volumes, security infrastructure implemented with production scale Internet connection.

Table 2 Three Performance Test architectures Compared

Consideration	Do It yourself	Outsourced	Performance Test Portal
Test tool license	<ul style="list-style-type: none"> ▪ You acquire the licenses for performance test tools 	<ul style="list-style-type: none"> ▪ Included in the price 	<ul style="list-style-type: none"> ▪ You rent a timeslot on their portal service
Test tool host	<ul style="list-style-type: none"> ▪ You provide 	<ul style="list-style-type: none"> ▪ Included in the price 	<ul style="list-style-type: none"> ▪ Included in the price
Internet connections	<ul style="list-style-type: none"> ▪ You must organise 	<ul style="list-style-type: none"> ▪ Theirs included. You liaise with your own ISP 	<ul style="list-style-type: none"> ▪ Theirs included. You liaise with your own ISP
Simplicity	<ul style="list-style-type: none"> ▪ Complicated, you do everything. 	<ul style="list-style-type: none"> ▪ Simplest solution – the services provider do it all. 	<ul style="list-style-type: none"> ▪ Simple infrastructure/tools solution, but you are responsible for building/running the test
Cost	<ul style="list-style-type: none"> ▪ Potentially very expensive 	<ul style="list-style-type: none"> ▪ Lower tool/infrastructure costs, but you pay for the services 	<ul style="list-style-type: none"> ▪ Low tool/infrastructure costs
Pros and cons	<ul style="list-style-type: none"> ▪ Could be very expensive. You acquire tools that may rarely be used in the future ▪ You need to organise, through perhaps two ISPs, large network connections (not an issue for intranets) ▪ You are in control. ▪ Complicated – you do everything. 	<ul style="list-style-type: none"> ▪ Potentially cheaper, if you would have hired consultants anyway ▪ Simpler, you need one arrangement with your own ISP only. ▪ You can specify the tests, the services company build/execute them, you manage the supplier. ▪ Simplest solution. 	<ul style="list-style-type: none"> ▪ Cheapest tool/infrastructure costs, but you still need to buy/acquire skills. ▪ Simplest infrastructure solution, but you must manage/perform the tests. ▪ You are in control. ▪ For majority of sites, a simple solution.

5.4 Availability Testing

What is availability? Essentially, availability is measured in terms of uptime – the proportion of time that a Web-based service is available for use divided by the total time in the measurement period. In mainframe installations, uptime might be as high as 99.99% over a period of one year. That sounds incredibly high, doesn't it? 99.99% availability amounts to around 9 hours down time in the year, where a system runs 24 hours per day, 365 days per year. Assuming all systems have to be down for short periods to perform maintenance activities, this is really quite remarkable.

What might you expect of an E-commerce site? You would wish the shop to be open 24 hours per day, 365 days per year, wouldn't you? The objective of all Web sites is to be continuously available. Availability testing aims to evaluate a site's ability to stay up for a long time. Does that mean tests should span extremely long timescales? Not necessarily.

Where sites are required to be continuously available, they tend to be designs with reliable systems components, but also built-in recovery features that operate when failures occur.

These features introduce diverse routing for networks, multiple servers configured as clusters, middleware and distributed object technology that handles load balancing and rerouting of traffic in failure scenarios. Diversity and redundancy are also often the mechanisms used to allow backups, software and hardware upgrades and other maintenance activities to be performed.

Unfortunately, these configuration options are rarely well tested. Rather, they are trusted to work adequately and it is only when disaster strikes that their behaviour is understood.

Availability testing normally involves:

- Identification of the components that could fail and cause a loss of service.
- An analysis of the failure modes or scenarios that could occur where you need confidence that the recovery measure will work.
- An automated test that can be used to load the system and explore the behaviour of the system over an extended period.
- The same automated test that can be used to load the system under test and monitor the behaviour of the system under failure conditions.

Reliability (or soak) testing

These are automated tests run over an extended period, perhaps 24 or 48 hours to find (usually) obscure problems. Performance testing will flush out all of the obvious faults. The automated test does not necessarily have to be ramped up to extreme loads. (Your stress testing covers that). But you are particularly interested in the system's ability to withstand continuous running of as wide a variety of test transactions to find if there are any obscure memory leaks, locking or race conditions. As for performance testing, monitoring the resources being used by the system helps to identify problems that might take a long time to cause an actual failure. Typical symptoms are resources being used up over time and not being replaced and response times getting progressively worse over the duration of the test. You might not actually experience a failure, but given time in production, you might be able to predict what will happen.

Failover testing

Failover testing aims to explore the behaviour of the system under selected failure scenarios. In higher integrity environments, a safety analyst might conduct a Fault Tree Analysis (FTA) to identify the dependencies of a service on its underlying components. Fault tree diagrams are sometimes extremely complex with many Boolean symbols representing the dependencies on

sub-systems or their components and services on their subsystems. These diagrams are used to identify the unusual, but possible combinations of component failures that the system must withstand.

Whether you use a sophisticated technique like FTA or you are able to identify the main modes of failure easily, testing follows a fairly standard process. With the automated test running, a range of failure modes are explored. It is a bottom-up approach.

- Early tests focus on individual component failures (e.g. you offline a disk, power down a server, break a network connection etc.)
- Later tests simulate more complex (and unlikely) failure scenarios (e.g. losing the power to a whole building)

You need to execute these tests with an automated load running to explore system behaviour in production situations and gain confidence in the designed-in recovery measures. In particular, you want to know:

- How does the architecture behave in failure situations?
- Do load-balancing facilities work correctly?
- Do failover capabilities absorb the load when a component fails?
- Does automatic recovery operate (i.e. do restarted systems 'catch up'?)

5.5 Security Testing

When most people think of security, they have a vision of long haired college drop-outs working into the small hours at a furious pace trying to crack a remote system. Although this image works well for the movies, it isn't helpful to our understanding the security threats that face E-business systems. Security hackers often work in teams, they are very highly automated, often they adopt a scattergun approach in acquiring targets, and they are more sophisticated than the movies usually represent them.

The intruder's attack methodology

Attackers (as opposed to Ethical Hackers, who are usually consultants) have an attack methodology. They do not attack randomly and they are extremely careful to protect themselves. Although it's amusing to think of them as sloppy burglars, tripping up and leaving lots of clues, this is not typical. The brief description below give an insight to their approach.

Target acquisition and information gathering

In this phase the attacker scans the many open sources of information, large networks and individual computers. The equivalent of 'rattling the windows and doors' to find a vulnerable network or computer.

Initial access

In this phase, the attacker gains entry. Not the back door, not the front door, but through a pin sized crack.

Privilege escalation

They are in. Now how do they get root or administrator privilege?

Covering tracks and planting backdoors

Altered versions of UNIX login script, netstat, ps utilities let them return without leaving a trace. Network 'sniffers' let them see and attack all interfacing systems. Log cleaners remove all trace of their activity.

5.6 Well documented holes have counter-measures, but are YOU up to date?

All of the major products on the market, including browsers, client and server operating system software, networking software, web server and development products have security vulnerabilities.

If you care to read a book (e.g. reference 13) or scan the web sites dedicated to publicizing security vulnerabilities, you will be staggered to learn just how many there actually are. The nature of these vulnerabilities and the tools (freely available) that exist to exploit them should make any system administrator very concerned. Of course, all of the published vulnerabilities have countermeasures or patches provided by the software suppliers, but it is the system administrators' responsibility to stay up to date and apply the patches.

Web sites such as www.cert.org (reference 23), and www.ntbugtraq.com (reference 24) exist to publicise the known vulnerabilities, risks and the availability of patches and configuration advice in all products or for a single product (e.g. Windows NT).

Security testing is normally performed in two distinct ways: security audit and ethical hacking.

Security audit

Audits are normally done in two ways.

Manual security audits aim to ensure that all of the products installed on a site are secure when checked against the known vulnerabilities for those products. Essentially, this entails checking that all the security patches have been installed correctly and vulnerabilities that can be countered by configuration settings have been implemented.

Automated security scans performed by free and proprietary tools that perform ping sweeps, port scans, operating system detection. Some (free) tools do all three at once. Using these tools, it is possible to identify what services are running on potential targets and then, to focus attention on the most exposed systems. Based on the scan results, specific countermeasures for each vulnerability can be applied.

Ethical Hacking

Using the attacker methodology, ethical hackers proceed to attempt penetration attacks. Using the initial scan results, hackers proceed to enumerate resources, accounts, user groups and services running on the target machines using free (or their own) tools. Simple as it sounds, the next task is to obtain the administrator or root password. This is done manually, using an automated tool or by listening in on login exchanges on the network. In some environments (mainly Unix), using obscure buffer overflow techniques, it is possible to log on to remote systems as root and gain access directly. Once in, the hacker can do whatever they wish on the target system.

The alternative to attacking servers through vulnerabilities in the network infrastructure is to attack applications directly. Web applications rely on the web server dispatching HTML source code to client browsers and the source code is visible. It is a simple task to change this HTML code in a text editor, open it with the browser, and execute the transaction on the target system with changed parameters. If you see an HTML form with a hidden field called price with a value of \$1999 – it couldn't be easier to edit this HTML and change price to \$1.99 and then post the form to the application server. Do you think you could buy a widescreen TV online for two dollars? You might, and it might be worth a try.

Cookies are another favourite route to sensitive data. If the password for a service is generated by the system under attack and stored in a cookie, try and register several new

accounts. Then examine the cookie data and see if there's a pattern. If there is and you crack the password algorithm, you can generate all the passwords on the system – past and future, perhaps. All you need now is someone else's username and you can automate as many login attempts as you like. Then you use their account to buy a widescreen TV!

Security auditing and ethical hacking require in-depth technical knowledge of the systems under attack, the tools available to assist and the imagination and persistence to crack into systems. It is a highly specialised skill and although a good system administrator or system programmer probably has the technical know-how, you really need to hire specialists to undertake this kind of testing.

6 LARGE SCALE INTEGRATION

In section 4.3, Transaction Testing, I described the test activity associated with integration of browser pages, Web server-based objects, other server-based objects and back end legacy systems. In the context of E-business testing, Large Scale Integration (LSI) testing covers the legacy system and external system integration not already covered.

Why do we need LSI testing?

- It may not have been technically possible to perform legacy or external systems integration earlier in the project OR
- It may have been technically risky to perform LSI (there are no test systems, only production systems to test on).

An example where LSI was essential

An internet banking application had ten major interfaces to new and legacy systems. The development of the new system itself was outsourced. The supplier, working with a customer test team were responsible for all testing up to and including system test but no testing subsequently. Integration and System testing were performed on the supplier site with all the interfaces to external systems stubbed out. LSI testing was performed by the customer in their own test environment where interfaces to the bank's test legacy environment and other newly developed support systems could be installed. LSI required it's own separate stage because of commercial, technical and logistic constraints.

Integration knowledge

Integration testing, in principle, is a 'white box' test activity. That is, the tester needs to know something about the physical interfaces between systems. Only by knowing something about the internals, can the tester design tests that will exercise these interfaces adequately.

The tester needs to know:

- The details of the internals of the interface.
- The nature of the system-system dialogs.
- How to exercise the interface from the application user interface.
- How to create test data to exercise the interface.
- How to find evidence that the interface works.

One of the problems of LSI testing is that it can be difficult to find details of interfaces. It's not just the interface details that cause problems. It may be that there is no documentation available at all for the legacy systems. In our experience, the LSI test team often has to write the document describing a system's interfaces as well as the test plan to ensure they work.

We'll assume interface documentation exists. Typically, to define the integration test plan the tester follows the same general process:

- For each interface, identify the dialogs between systems and which business or system event triggers them to work.
- Derive test cases for success and failure to negotiate each step in the dialog.
- Derive test cases from the interface data validation/use descriptions to ensure valid data is transmitted, invalid data is rejected and that the storage and use of data in each interfacing system reconciles.
- Define your test environment/infrastructure needs early so you get them in good time.

Conducting integration tests

Integration tests tend to be either very simple and easily automated or complicated and have to be manually executed. In general, early tests focus on the correctness of the interface calls. Later tests (usually automated) focus on memory leaks, loss of synchronization between systems and failure and recovery of client, server or network.

6.1 End-to-End Functionality

How does end to end testing differ from Large Scale Integration testing? Broadly, End-to-End tests are usually associated with user acceptance. Assuming that the technical testers have proven the interfaces between the new system and other systems work, the imperative for a business wishing to deploy the new system is to ensure the system supports the intended business activity. For example, if a system supports on-line purchase and delivery of books:

- Can the user search for a book, add it to a shopping basket and place an order?
- Can the customer credit card be validated, payment authorised and processed successfully?
- Does the legacy order processing system receive the on-line order accurately?
- Is the book in stock, located in the warehouse, packed, labelled and dispatched correctly?
- Are “order confirmation”, “progress notification” and “thank-you for ordering” emails sent at the right time and reaching the customer reliably?
- And so on...

Ultimately, the sponsors of the system want to know whether the new system meets the cardinal business objectives of the project. To demonstrate this, testers must trace paths through the business process and develop scenarios that will exercise the system and provide evidence that the system supports the business process.

Compared with system testing, it may take a relatively small number of test cases to give confidence that the system works correctly. The difficulty in staging such tests at all, is they require the entire technical architecture.

7 POST-DEPLOYMENT MONITORING

Imagine that you were investigating an E-commerce web site and thinking about using it for the first time and it failed. Would you report a problem to the company running the site? It's not likely that you would. Once again, the general attitude to web sites is just like retail stores. If a store is closed, service is slow or difficult to use, you just walk on by and visit the next store in the street. It's the same with Web sites. Search engines and portals offer you a multitude of websites matching your search criteria so it is easy to click on a different link. With some E-Business, for example electronic banking, this is less likely. After all, the bank has YOUR money and you want it, so you would complain pretty quickly.

Unlike users of our internal systems, we cannot rely on users of our web site to report problems.

With this background, it is sensible to monitor your site in production. Is this testing? It's not the traditional role of testers to test a site in production, but monitoring a live site involves exactly the same activity as testing a development site.

There are four main areas where post-deployment monitoring is appropriate:

Availability

- is our site available - NOW?
- You might check this every 15 minutes, every single day.

Link checking

- Does your site rely on many links to external sites?
- Have target sites and resources been moved?
- You might check this once per day.

Object load times

- Are our pages (and images) getting bigger over time?
- Is the database becoming huge?
- Are key transactions getting slower?
- You might check this once a week.

Security

- Are our security policies still sound?
- Are we vulnerable to a new form of attack?
- Most likely to be done by internal or external consultants using scanning tools.
- You might check this every time a new vulnerability is announced.

The report below was produced for the Evolutif web site by the NetMechanic tool. This is typical of the reports that these test tools produce.

Report For: www.evolutif.co.uk
Date: Thu 28 Sep
Time: 6:00 - 13:00 USA Eastern Time
Server Type: Microsoft-IIS/4.0

Overall Rating: Fair

Performance Summary			
Event	Your Server's Average	NetMechanic Average	Percentile
Host Ping	*	277.26 millisecc	th
DNS Look Up	0.18 sec	0.13 sec	9 th
Connect Time	0.09 sec	1.11 sec	62 th
Download Time (10k file)	0.50 sec	1.10 sec	37 th
Timeouts	0	-	

Host Ping: Indicates the network speed between your server and ours.
DNS Look Up: Indicates the time required to look up your server's address in the DNS database.
Connect Time: The time required to open a network connection to your server.
Download Time (10k file): The time which would be required to download a 10k file from your server.
Number of Timeouts: The number of times your server failed to respond within 30 seconds.

Warning areas are highlighted in dark yellow; problem areas are highlighted in red.

What can be checked?

- Test sites can 'Ping' your site to see if the server hardware is responding.
- HTTP requests can tell whether the Web service is running.
- HTTP GETs of selected web pages can tell whether pages are downloadable and the speed can be monitored over time.
- Calls to CGI programs can execute selected test transactions to ensure that connections to back end systems are also operating properly.
- Link checkers scan all pages for links, follow them until they result in off site pages, they are checked and the test stops.

Security scanners can evaluate exposure to product vulnerabilities.

There are three options for post-deployment monitoring of web sites:

Do it yourself

In principle, you could reuse the automated tests that you prepared during the development phase and execute these on a remote machine on a different site to where your systems operate. You would also want to base this test machine on a site connected through a different ISP to ensure ISP problems don't affect the test machine as well.

Remote monitoring services

There are now many companies offering continuous site monitoring. These are often the same companies that have remote HTML validation and speed-checking portals. In these cases, prices can be as low as a few dollars per month to monitor a site or specific pages. The companies may have remote agents-based in many places to provide comparative timings. The more sophisticated services are more expensive, of course. Most offering include regular (e.g. 15 minute) checks and alerts in the form of email, fax and pager messages. Weekly and monthly trend reports may also be offered as part of the service.

Consultants

Consultants are most often used for security audits. The general recommendation is that you should do a security audit every three months, although it might be most wise to do spot checks when every new vulnerability (on products you use) is publicised to ensure you are not exposed. Be aware, the hackers monitor the same security alert email lists as you do. When a new alert is published, they have a window of opportunity to exploit it before the patch is installed in all exposed sites.

8 REFERENCES

E-Business Testing

1. <http://www.evolutif.co.uk/> E-Business Risk-Based Testing, Part 1: E-business Risk and Test Strategy. Paul Gerrard, June 2000.

Tools and tool Listings

2. www.softwareqatest.com/qaweb1.html Comprehensive Web (and other) test tools listing.
3. <http://Validator.w3.org> Link checker.
4. <http://www.netmechanic.com> Link checker and other site validation:
5. <http://www.htmlvalidator.com/> CSE HTML Validator it user configurable HTML, XHTML, and WML syntax checker available for Windows 95/98/2000/NT. With a built in browser facility (needs IE4 or later), you can browse pages, see the page source code and validate it at the same time.
6. <http://www.cast.org/bobby/> Bobby is a free application that will analyse web pages for their accessibility to people with disabilities. It will also find HTML compatibility problems that prevent pages from displaying correctly on different web browsers. Available as a Web-based services and downloadable tool.

Testing Papers

7. <http://www.evolutif.co.uk/> Client/Server Performance Testing, Paul Gerrard and Andy O'Brien 1995. This paper describes the end-to-end process of planning. Preparing, executing and analysing performance tests in a client/server environment. Although it doesn't mention the Internet, the principles appear to be unchanged since the paper was written in 1995.
8. <http://www.evolutif.co.uk/> Testing GUI Applications, Paul Gerrard, 1997.

Books

9. How the Internet Works, Preston Galla, ISBN 0-7897-2132-5. Best selling introductory text.
10. Webmaster in a nutshell, (five books on CD-Rom), O'Reilly
 - HTML: the definitive guide
 - JavaScript: the definitive guide
 - CGI programming for the world wide web
 - Programming Perl
 - Webmaster in a nutshell
11. Understanding Electronic Commerce, David Kosiur, Microsoft Press
12. Designing Web Usability, Jacob Nielsen, ISBN 1-56205-810-X. A very readable, well illustrated and up to date book on web usability.
13. "Hacking Exposed: Network Security Secrets and Solutions" - McClure, Scambray and Kurtz ISBN 0 07 212127 0 <http://www.hackingexposed.com>

Web Resources

14. E-commerce Awareness Forum newsletter, www.mbs-program.com
15. Web Accessibility Initiative:
<http://www.w3.org/WAI/www.webreference.com/authoring/design/tutorials.html>

16. General Web usability. This is Jakob Nielsen's web site focusing entirely on usability issues: <http://www.useit.com/>
17. Response times: the three important limits:
<http://www.useit.com/papers/responsetime.html>
18. How to conduct a heuristic evaluation:
http://www.useit.com/papers/heuristic/heuristic_evaluation.html
19. Software Usability Measurement Inventory [SUMI] & Website Analysis & Measurement Inventory [WAMMI]: www.ucc.ie/hfrg/ & www.acm.org/~perlman
20. Software for Use, Constantine and Lockwood, ISBN 0-201-92478-1. Comprehensive and up to date treatment of usage-centred design with information on all main methods of usability test and evaluation. See accompanying web site:
<http://www.foruse.com>.

Security

21. Internet Security Policy: A Technical Guide, NIST,
<http://csrc.nist.gov/isptg/html/ISPTG-1.html>
22. eCommerce Trust Study, Cheskin research, <http://www.studioarchetype.com/cheskin/>
23. <http://www.cert.org> - generic security bug watch run by the Software Engineering Institute – register and get regular advisories on new vulnerabilities in the major OS, Web and browser technologies.
24. <http://www.ntbugtraq.com> - same as the CERT site above, but for Microsoft NT only.
26. Firewalls Frequently Asked Questions – excellent starting point for this topic,
<http://www.faqs.org/faqs/firewalls-faq/>

Other References

27. <http://www.w3.org/TR/html4/> HTML 4.01 Specification, W3C Recommendation 24 December 1999
28. World Wide Web consortium: <http://www.w3.org>.