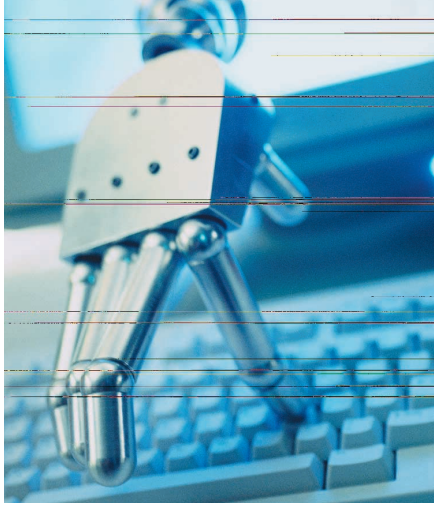


Using a simple test matrix can help you spot test areas that fit your system's needs.

Wing Lam

Testing E-Commerce Systems: A Practical Guide

As e-customers (whether business or consumer), we are unlikely to have confidence in a Web site that suffers frequent downtime, hangs in the middle of a transaction, or has a poor sense of usability. Testing, therefore, has a crucial role in the overall development process. Given unlimited time and resources, you could test a system to exhaustion. However, most projects operate within fixed budgets and time scales, so project managers need a systematic and cost-effective approach to testing that maximizes test confidence.

This article provides a quick and practical introduction to testing medium- to large-scale transactional e-commerce systems based on project experiences developing tailored solutions for B2C Web retailing and B2B procurement. Typical of most e-commerce systems, the application architecture includes front-end content delivery and management systems, and back-end transaction processing and legacy integration.

Aimed primarily at project and test managers, this article explains how to

- establish a systematic test process, and
- test e-commerce systems.

Inside

THE TEST PROCESS

You would normally expect to spend between 25 to 40 percent of total project effort on testing and validation activities. Most seasoned project managers would agree that test planning needs to be carried out early in the project lifecycle. This will ensure the time needed for test preparation (establishing a test environment, finding test personnel, writing test scripts, and so on) before any testing can actually start.

The different kinds of testing (unit, system, functional, black-box, and others) are well documented in the literature. However, less well articulated (despite the good general texts in software engineering) is the test process that ties together all testing and validation activities used to deliver a quality solution to the client. Table 1 presents a matrix model of the test process that my colleagues and I have come to adopt on several e-commerce projects. Several goals (listed in the table's first column) guide this model; one or more processes support the achievement of each goal.

We prefer the term "goals" to "phases" to indicate the fact that goals are concurrent. That is, you can be testing the system's current release, but at the same time, setting and agreeing to realistic expectations for the next release.

The model—or at least parts of it—should be familiar to those experienced in test management roles. Those who aren't or don't follow a systematic test process can use the matrix model as a template for planning testing and validation activities by allocating processes to a project plan. It is also useful as a tool for improving a test process by checking the model against the actual process.

TESTING AN E-COMMERCE SYSTEM

So how do you test an e-commerce system?

1520-9202/01/$10.00 © 2001 IEEE

What types of testing do you need to do? Here, I present a checklist of test areas that group together common types of testing normally applied to e-commerce systems. (Some of these tests also apply to other types of systems.)

You can cover certain test areas (such as browser compatibility) relatively quickly, whereas others (like those for transaction integrity) require a higher level of effort.

Browser compatibility

Customers will become frustrated if they can't get to your e-commerce site easily. Though relatively simple to do, it pays to spend enough time testing in this area.

Lack of support for early browsers. Not all browsers are equal. For example, some early versions of Internet Explorer and Netscape Navigator do not fully support

Table 1. Matrix model of the test process.

Goal	Processes
Set and agree on realistic expectations for the system	Prioritize business requirements; identify those that are essential and those that are optional. For a given delivery date, filter out, reprioritize, or renegotiate unrealistic system expectations. Specify requirements that are testable. Remove or firm up ambiguous requirements. Define acceptance criteria that are meaningful to the business. Define relevant industry standards or competitive benchmarks. Ensure that stakeholders agree to a capacity plan for the system with measurable capacity requirements.
Define a test strategy	Formulate and document a test strategy; relate it to the test quality plan. Identify high-risk zones early on; mark these for early testing and keep them away from the project's critical path. Identify all aspects of the system that require testing; in each case, indicate the level and form of testing. Identify relevant testing tools that can help automate or document elements of the test process.
Plan the testing	Estimate required resources, such as tools, people, hardware and software infrastructure, test location, and so on. Define roles and responsibilities; identify a suitably qualified test manager and test team. Draw up and agree to a detailed test plan with the client; define test milestones and test deliverables. Plan a pre-release simulation period where the system can be observed under simulated business usage.
Set up the test environment.	Set up the physical test environment—for example, the hardware and software infrastructure. Prepare test material such as detailed test plans, test scripts, and test data. Prepare test materials with reuse in mind. Set up the defect-tracking system; decide on the form of defect reporting and any test metrics to use. Define test standards and procedures; educate individuals on the use of standards and procedures. Acquire and prepare the necessary test tools; ensure that the test team receives relevant training.
Perform testing	Conduct appropriate quality reviews, such as code reviews and walk-throughs. Conduct testing, whether at the unit, integration, system, or use-acceptance level. Execute tests and collate test results; record results and incidents in the defect-tracking system. Analyze test results; monitor areas with a high incidence of defects and defect severity. Track and manage defect resolution. Where appropriate, repeat the test and reuse test materials to expedite retesting. Stage a pre-live simulation period; test the system under expected business usage and observe results. Manage the decision to go live based on testing results; consider if the system has achieved a release standard.
Monitor the deployed system	Monitor system performance using relevant monitoring tools; use performance reporting to guide performance-tuning activities. Capture any residual defects; assess their priority and, if serious, consider a rectification plan. Gather feedback and problem reports from users; analyze these to identify future improvements.
Manage successive releases	Capture and prioritize new business requirements; assess their impact on the system before committing to them. Fix outstanding and residual defects. Plan the system's next release; agree to a release plan with relevant stakeholders. Ensure a smooth transition from the current to the new release.

JavaScript. Decide on the lowest level of compatibility—for example, "the system should be compatible with IE version 3.0 and above and Netscape version 2.0 and above"—and test that the system does indeed work without problem on the early browser versions.

Browser-specific extensions. HTML is an evolving standard (HTML 4.0 is the latest specification) and should not be confused with HTML extensions specific to a particular browser implementation. For example, the table border color tag (<TABLE BORDER COLOR="$$$$$$">) is specific to Microsoft Internet Explorer; the Server script tag (<SERVER>) is an extension introduced with Navigator version 4.0.

Browser nuances. Even with the same release version, browsers behave differently on different platforms, and when used with different language options. Testing should cover at least the main platforms (Unix, Windows, Mac, and Linux) and the expected language options.

Page display

The display of pages in a browser forms the all-important interface between the customer and the business. Get this wrong in any way, and the job of building customer confidence and trust becomes much more difficult.

Incorrect display of pages. Displaying all pages correctly is not as easy as it sounds—most e-commerce systems generate Web pages dynamically (that is, they access a database and display the results in a browser) rather than display static HTML. Problems can occur—for example, when the browser retrieves an empty result set from the database or when a database connection is unavailable.

A page display might also involve loading resources (such as applets) beforehand, and you could encounter problems when such resources are unavailable. Test that your system handles such exceptions appropriately.

Runtime error messages. Users get frustrated when a browser throws up unexpected error messages. Such error messages are generally unfriendly and not meaningful to users; moreover, they leave a poor impression on the customer. Carry out tests to ensure that the application captures and handles all errors by, for example, generating an appropriate and user-friendly error page.

Poor page download times. US Web tracker Zona Research estimates that page load times of 8 seconds or more combined with ISP download times could cause up to 30 percent of customers to leave a site before buying anything. Pages that have a high graphical content and/or use applets, though aesthetically pleasing, are often download problematic. Test download time under realistic test conditions (that is, account for typical Internet traffic) rather than testing it locally.

Dead hyperlinks. Dead hyperlinks are a frequent cause of customer frustration; even the most popular portals suffer from hyperlinks that lead nowhere. Several automated tools now test hyperlinks.

Categories of Test Tools

Configuration management tools **manage version and change control of system artifacts such as software components and test documentation.**

Database performance tools **measure database performance, such as the time to perform selected database queries. These tools facilitate database optimization.**

Debuggers **are typical programming tools that find and resolve software defects in the code. They are part of most modern application development environments.**

Defect management systems **record defects and track their status and resolution. Some include reporting tools to provide management information on defect spread and defect resolution rates.**

Network monitoring tools **watch the level of network traffic. They are useful for identifying network bottlenecks and testing the link between front- and back-end systems.**

Regression testing tools **store test cases and test data and can reapply the test cases after successive software changes.**

Site monitoring tools **monitor a site's performance, often from a user perspective. Use them to compile statistics such as end-to-end response time and throughput, and to periodically check a site's availability.**

Stress tools **help developers explore system behavior under high levels of operational usage and find a system's breakpoints.**

System resource monitors **are part of most OS server and Web server software; they monitor resources such as disk space, CPU usage, and memory.**

Test data generation tools **assist users in generating test data.**

Test result comparators **help compare the results of one set of testing to that of another set. Use them to check that code changes have not introduced adverse changes in system behavior.**

Transaction monitors **measure the performance of high-volume transaction processing systems.**

Web site security tools **help detect potential security problems. You can often set up security probing and monitoring tools to run on a scheduled basis.**

Tips for Better Site Navigation

- **Provide global navigation on each page so that the user has a feel for where a page sits within the overall site and what else the site has to offer.**
- **Provide hyperlinks to other pages on the site that have related content.**
- **Ensure navigation bars are consistent, both in terms of form and placement on the page.**
- **Use text navigation in preference to icons and pictures as this is less ambiguous.**
- **The *four-click rule* states that a user should be able to get from where he currently is in a site to any other point in the site in four clicks.**
- **Keep pages short to minimize the amount of scrolling a user has to perform. With long pages, publish a table of contents near the top to let users jump quickly to the appropriate point on the page.**
- **For large sites, provide a search box or search form to enable users to quickly search it.**
- **Provide a site map so users can get a bird's-eye view of the entire site.**
- **Provide a link to the home page on every page so that users can always return to a central reference point.**
- **Provide and highlight navigation links to the high-priority areas you want users to see.**

Plug-in dependency. Developers gear some sites toward browsers that have a particular plug-in—the Flash graphics plug-in or language plug-ins are some examples. However, it is unreasonable to turn away a potential e-customer simply because the site is unusable without the plug-in. Test that a site is functionally equivalent with and without the plug-in.

Aesthetics. Sites need to be aesthetically pleasing—those that aren't run the risk of conveying a poor image or, even worse, damaging an existing brand image. Where two competing sites are otherwise equal, a user's choice may rest on aesthetic appeal alone. Though somewhat subjective, aesthetic reviews against competitive sites are valuable and force you to look at what competitors are doing.

Font sizing. Most browsers allow users to change font sizes. For example, in MS Internet Explorer, users can vary font sizes from "smallest" to "largest." Designers often work with a particular font setting on their browser; consequently, site designs can often break-up in different font settings.

Session management

HTTP is a stateless protocol, and server-side programming tools that use Java or Active Server Pages make extensive use of session objects to capture state information. This is, for example, how many pages store items in a shopping cart. Our experience indicates that any system using session management should be tested for several characteristics.

Session expiration. Most applications and Web servers configure sessions so that they expire after a set time. Attempting to access a session object that has expired causes an error, and must be handled within the code; often it's not. Developers tend to overlook testing for session expiration, largely because under normal operational circumstances, session expiration is unlikely to occur.

Session storage. Consider if any issues relate to the storing of session objects (which often depend on the application software). For example, session objects may be stored as cookies, but what if users do not have cookies enabled in their browsers? Also, are there any limitations to the physical size of session objects?

Usability

Good site usability is crucial for attracting and retaining customers; indeed, one study indicated that 67 percent of customers did not complete online purchases because of poor usability.

Usability covers a broad area and is often hard to define—or never defined at all. However, this lack of definition conflicts with testing and the need to have testable requirements. (If you can't test requirements, how will you know that they have been met?)

As a rule of thumb, usability is the ease with which a customer can perform a set of tasks, typically determined by the time taken for successful completion. You can test usability through usability studies, in which you give a group of individuals selected tasks to carry out, observe their performance, and collect feedback. Alternatively, you can obtain this information in joint facilitation sessions in which Web site designers sit down with users immediately after a usability session and obtain direct verbal feedback on the site's usability.

Nonintuitive design. You can label aspects of a Web site as nonintuitive if, on average, it takes a new visitor a disproportionate amount of time to perform set tasks. When designers become closely involved in a site's design, it becomes harder for them to objectively assess the site's intuitiveness.

Poor site navigation. Site navigation is crucial for attracting customers and retaining them. Sophisticated Web sites, such as for travel booking, need to pay particular attention to navigation issues. (See the "Tips for Better Site Navigation" sidebar.)

Catalog navigation. Large product catalogs are central to many e-commerce systems, B2B procurement systems in

particular. Customers should be able to quickly browse and search through catalogs. Developers can define tests to measure the effectiveness of product navigation mechanisms. For example, you could test that a search on particular keywords brings up the correct products.

Lack of help. It's not always obvious what to do on a Web site. For example, users may need to fill out forms in a particular sequence because they have associated validation rules that raise errors when some information is not correctly filled in. Guiding a customer through interactions is a key factor for first-time visitors.

"Test" content for offensive, misleading, or litigious statements; infringements; and personalization errors.

Content analysis

So far, I've been concerned with the delivery and presentation of content. However, the actual content provided by a site also needs to be "tested."

Offensive, misleading, and litigious content. An e-commerce site's contents must be sound. In the UK, legislation such as the Trade Marks Action (1994), Control of Misleading Advertisements Regulations (1988), and Obscene Publications Act (1976) applies to all published material, including Web content. (See http://www.marketinglaw.co.uk for current UK- and EU-specific legislation.) Products or services that are incorrectly described or misleading (check those product descriptions!) may violate the Trade Description Act.

Infringements. Test your Web site content for infringements. Graphics and images used in the design should be royalty free (if not wholly owned); also check for copyright infringement. In some circumstances, the courts will consider use of trademarked material without the owner's written consent as infringement. Courts would almost certainly view discrediting or denigrating a competitor's brand as an infringement.

Personalization. Increasingly, e-commerce systems are incorporating customer relationship management functionality that provides personalization for individual customers. In many cases, a personalization profile determines what content the site offers to these customers. However, situations can occur where the site offers incorrect or inappropriate content to customers because of errors in the personalization functionality.

Availability

Unavailability equals lost revenue; it also harms a business's reputation and can encourage customers to take their business to competitors. Businesses need to offer 24/7 availability to their customers, with availability levels of 99 percent or higher being the desired norm.

Unacceptable levels of unavailability. Several factors can influence availability, such as hardware reliability, software reliability, the effectiveness of load balancing, and the database's ability to handle concurrent users. Before going live, predicted business usage patterns should indicate maximum stress levels. You should test system availability against the maximum stress levels plus a safety margin for a defined period of time.

Denial of service. Yahoo, Amazon.com, and Buy.com have all recently suffered denial-of-service attacks. Such attacks (also known as saturation attacks) involve bombarding the Web server with bogus requests, making it inaccessible to genuine users. Organizations are not defenseless against such attacks; Amazon.com for example, uses security software to filter out bogus requests. Test your e-commerce systems for vulnerability to denial-of-service attacks.

Backup and recovery

You can't guarantee that any component of your e-commerce system won't fail, whether hardware or software, so it's sensible to test that you can quickly recover from a failure when it does happen.

Failure/fall-over recovery. Design systems so that one component's failure doesn't necessarily bring down other components. In MS Internet Information Server (version 4.0), for example, Active Server Page scripts can run from the Web server as separate processes.

Test both the speed and ease at which systems can recover from component failure; for example, how do you restart the service? Is it a case of pressing the restart button, or should the system support team look at other factors first? Is it a simple administrative task, or one that requires specialist knowledge? For non-fatal failures, you might be able to reload the system's affected components rather than taking the system down completely. For example, in a Java implementation, you can reload Java servlets dynamically.

Backup failure. How quick and easy it is to perform and restore backups? If database records become corrupted, for example, how quickly can you backup the data? How quickly can new hardware, software, and networking components replace failed components?

Fault tolerance. In fault-tolerant architectures, redundant hardware and software components take over when components fail. For example, if one Web server goes down, a second Web server can service requests, and the problem remains transparent to the user. Test fault tolerance by switching off individual components or simulating their failure. My colleagues and I know of one incident where the system used clustering technology to provide fault tolerance. However, testing found that the clustering technology was ineffective because it was incorrectly configured.

Transactions

Transaction processing is a central element of most e-commerce systems. With distributed systems, transaction processing can span several individual systems, leading to more complex testing.

Transaction integrity. Does the system correctly perform all aspects of transaction processing? For example, does the system call the appropriate database triggers and procedures? Does it commit and roll back transactions correctly? Have system developers set the appropriate isolation levels? Devise a set of inputs that will attempt to create "nonstandard" transactions—for example, transactions with a zero quantity or value—as well as standard ones.

Functional testing typically consumes between 25 and 50 percent of the total testing effort.

Throughput. The most popular Internet sites easily exceed a million hits per day. In a transactional e-commerce system, you must test the transaction engine's throughput; that is, its ability to manage, say, 1,000 transactions per minute. Most modern testing and monitoring tools have facilities for throughput testing.

Auditing. An audit trail records all transactions made by the system. Tests should verify that the audits satisfy all legal obligations, and that the auditing software captures all the required transaction details, such as transaction time and date, originator and recipient, values, and so on.

Shopping, order processing, and purchasing

My experience suggests that functional testing typically consumes between 25 and 50 percent of the total testing effort. In most e-commerce systems, shopping, order processing, and purchasing form the core functionality. Other functional areas can include customer profiling, discount and offer management, automatic marketing, and inventory management. Although most developers largely consider functional testing a manual process, tools (such as Mercury Winrunner) can often help automate aspects of functional testing by automatically capturing and replaying user interactions.

Shopping-cart functionality. Many e-commerce sites use shopping carts (or some variant along the same theme) as a mechanism for the customer to compile orders. Basic shopping carts provide "list and total-up" functionality, whereas more sophisticated ones can provide extra functionality, such as tax and shipping calculation, loyalty discounts, and "save and recall" features for orders. Testing should cover all aspects of shopping-cart functionality based on test suites of shopping items.

Order processing. Order processing can involve the automated creation or update of transactions, or the delivery of information to back-end systems.

Payment processing. Approaches to payment processing vary from the simple capture of credit card details for offline manual processing to real-time card processing. Irrespective of the chosen approach, you must test several basic actions, such as determining whether the system carries out the necessary validation and debits the correct amount from the credit card. The log must also correctly record each transaction.

Order tracking. An increasingly common feature in B2B systems is the ability to track orders. One point of testing is the accuracy of the order-tracking functionality. You need to enter test orders into the system, and compare the automated system's status report to the actual order status.

Internationalization

E-commerce systems, especially B2C systems, are often required to operate across geographic boundaries. Although English is the dominant language, non-English sites are increasingly common and gaining a head start in cornering foreign markets. A system must operate in selected target countries as well as it does in its home country.

Language support. Does the site offer the option to view non-English pages? If the choice of language is based on browser preferences, does it work on all the desired browsers? Many older browsers do not support language customization.

Language display. Test that words are correctly displayed and that sentences are grammatically correct. Use a native speaker to verify that this is the case.

Cultural sensitivity. Does the site respect cultural differences between the home and foreign country? Check that a site does not include content that is likely to offend the culture or religious affiliation of the country concerned.

Regional accounting. Developing e-commerce systems where trade takes place across accounting regions raises issues about how to charge customers. For example, tests should ensure that the system correctly calculates taxes and exchange rates for the regions concerned.

Operational business procedures

E-commerce business procedures often involve both manual and automated procedures. A telephone help desk, for example, can handle product returns or order inquires, but relies on information from the automated processes. Don't forget that you must test e-business procedures, not just the e-commerce system per se.

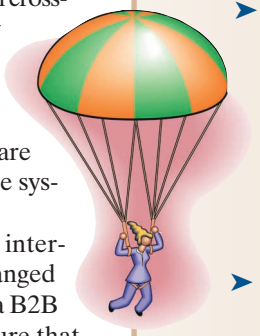
As a first step, identify the various business scenarios that can take place. For example, a customer can request a product return (over the phone, or via e-mail or other communications means) or cancel an order. Payments can be delayed or canceled.

Next, try and estimate the size and scale of each scenario: How many returns are you likely to get during a peak

period? Simulate the peak situation and see how well the e-business procedure copes. In particular, observe where the main bottlenecks are and where you can make the most value-added improvements. Before going live, ensure that you know at least the rough capacity of operational business procedures.

Systems integration

Developers must often integrate e-commerce systems with existing back-office or legacy systems, such as accounting and stock-control systems. Or, you could need to integrate e-commerce systems with third-party applications, such as e-mail. In some cases, such as in cross-platform integration, you can employ specialized middleware technology such as CORBA (the Common Object Request Broker Architecture) or messaging (implemented in software such as IBM MQ Series) to integrate systems.



Data interface format. The data interface defines the format of data exchanged by front- and back-end systems. In a B2B scenario, for example, we must ensure that an order between a supplier and a consumer conforms to an agreed order format. Such interfaces can vary from simple to highly complex, and tests should ensure that the system generates and transmits data in the correct format. Tools such as XML (Extensible Mark-Up Language) alleviate data interface problems by providing document type definitions (DTDs).

Interface frequency and activation. The processing between front- and back-end systems may be time dependent. For example, a back-end system could be necessary to process a data transmission from the front-end system immediately or within a defined period. Tests should ascertain whether a system actually observes timeliness constraints and whether it activates data transmissions at the correct time.

Updates. One system must often update information in another system. Verify that batch programs and remote procedures perform the necessary update operations without side effects.

Interface volume capacity. Do the interfaces handle the volume of data envisaged? Bear in mind that legacy back-end systems may not have been designed for the volume of data typical today.

Common Testing Pitfalls

- ***Poor estimation.*** **Developers underestimate the effort and resources required for testing. Consequently, they miss deadlines or deliver a partially tested system to the client.**
- ***Untestable requirements.*** **Describing requirements ambiguously renders them impossible or difficult to test.**
- ***Insufficient test coverage.*** **An insufficient number of test cases cannot test the full functionality of the system.**
- ***Inadequate test data.*** **The test data fails to cover the range of all possible data values—that is, it omits boundary values.**
- ***False assumptions.*** **Developers sometimes make claims about the system based on assumptions about the underlying hardware or software. Watch out for statements that begin "the system should" rather than "the system [actually] does."**
- ***Testing too late.*** **Testing too late during the development process leaves little time to maneuver when tests find major defects.**
- ***"Stress-easy" testing.*** **When testing does not place the system under sufficiently high levels of stress, it fails to investigate system breakpoints, which therefore remain unknown.**
- ***Environmental mismatch.*** **Testing the system in an environment that is not the same as the environment on which it will be installed doesn't tell you about how it will work in the real world. Such mismatched tests make little or no attempt to replicate the mix of peripherals, hardware, and applications present in the installation environment.**
- ***Ignoring exceptions.*** **Developers sometimes erroneously slant testing toward normal or regular cases. Such testing often ignores system exceptions, leading to a system that works most of the time, but not all the time.**
- ***Configuration mismanagement.*** **In some cases, a software release contains components that have not been tested at all or were not tested**
  wit

Integrated performance. The performance of two systems integrated together can be a key issue. For example, you may need to integrate an e-commerce system with the payment systems of a financial processor to process credit cards online.

In another scenario, employing middleware technology such as CORBA can incur unacceptable overhead by using object brokers. Perform tests under simulated business usage to ensure acceptable integration performance.

Performance

The performance that matters the most is the end performance delivered to the customer, and Internet traffic makes this performance variable. Testing's goal is to ascertain a system's performance degradation profile (how fast and how much it degrades) and system breakpoints, which are the thresholds over which a system component, hardware or software, will crash or stop functioning.

Performance bottlenecks. An e-commerce system is only as strong as its weakest link. For example, performance suffers when you pair a Web server that can handle thousands of multiple users with a database that can handle only 20 concurrent connections. Effective testing involves identifying and stressing the weakest links to avoid overspending on the strong links.

Load handling. A large UK catalog firm's supply chain became unable to cope with order processing in time for Christmas 1999. This situation forced the business to turn away customers in mid-December. Determining the rate of performance degradation (the degradation profile) and a system's breakpoints is essential to achieving specific quality of service (QoS) requirements. For example, you can specify a QoS requirement such as, "Performance will drop no more than 10 percent when the number of concurrent users reaches 10,000."

You can use tools such as Apache JMeter (http://java.apache.org) and WebLoad (http://www.radview.com) to test server performance both on static and dynamic resources (such as CGI scripts, servlets, or Perl scripts) under different load types. You can ramp up the traffic on a Web site from 100 to 1,000 simultaneous users, depending on your estimates for normal and peak business usage.

Scalability analysis. E-commerce systems that do not scale well are limited from day one. Scalability analysis involves "testing" a system's architectural design and its capability to meet future (rather than current) performance requirements. Basic areas of scalability are hardware, application software, databases, and middleware.

Horizontal scalability involves adding new components, such as servers. Vertical scalability concerns how existing components can be grown, for example, by adding more disks to a disk array. Scalability analysis seeks to identify where—and how easily—you can scale the architecture.

Login and security

Security (or a lack of it) is a major barrier to e-commerce, particularly in Europe. With the rise in credit card fraud and high-profile hacker attacks, customers increasingly avoid e-commerce sites and systems they perceive as insecure.

Login/registration capability. Several potential problems can arise from failed login functionality. The most typical of such problems is a user providing a valid login and password but not being able to log in. Other problems can be more complex, such as when the system sets personalization settings incorrectly. Where appropriate, tests should also verify that the system correctly sets privilege levels.

Penetration and access control. Security personnel define penetration as unauthorized access to restricted areas and information. External penetration refers to attacks from the outside (over the Internet); internal penetration refers to attacks from within.

A large recruitment agency recently damaged its image when hackers gained unauthorized access (over the Internet and without password) to its resume database. This database contained sensitive information on workers' personal details and salary.

We also know of a case where an intruder gained access to unauthorized areas by simply typing in a URL directly rather than going through the intended route via the home page. You should test security measures against unauthorized access to determine the actual level of protection.

Insecure information transmission. Transfer of information occurs from browser to server, server to browser, server to back-end system and vice versa. For some, Secure Sockets Layer (SSL) has become the de facto standard for secure online communications. Where information is sensitive, you must test against stipulated security requirements. What level of encryption is required, and does the encryption system meet these requirements?

Web attacks. How prone is the system to Web attacks such as spoofing, snooping, and sniffing? Where Web attacks are a high priority (such as for financial systems), consider a security evaluation from an independent security organization. Some security tools, for example, check for sniffers and highlight insecure network configurations.

Computer viruses. How resilient is the system against computer viruses? Test that any virus protection software loads at the appropriate time, and that the necessary procedures (manual or automated) to update virus files are in place.

Digital signatures. The use of digital signatures is on the rise, particularly in secure document exchange applications where authentication and nonrepudiation are key. Tests should cover the means by which digital signatures are generated and issued. They should also check whether certificate files are loaded and installed correctly, and that the application handles invalid signatures. Don't assume that it will work; test that it does!

Testing's goal is to determine a system's degradation profile and breakpoints.

The production of high-quality e-commerce systems relies on comprehensive and well-managed testing. I strongly suggest that project and test managers treat the ideas presented here not as a definitive guide, but as a framework for fleshing out, rethinking, or evaluating their own test models and strategies. We have a simple rule of thumb about testing: If in doubt, you need to do more testing. ■

Wing Lam *is a consultant with Accenture; he has been involved in the design and delivery of large-scale e-commerce systems. Contact him at wlam_uk@yahoo.com.*