SQA India is a partnership firm based in India Offering a wide range of Software quality assurance and I.T. Enabled Services that match the high global standards in terms of efficiency and accuracy.

Welcome to the Software Quality Assurance (QA) and Testing Information Center. It offers articles, best practices, methodologies, FAQ, guides, books, directories, and tools that are related software QA and testing. Popular materials include job interview FAQ, test plan samples, Win Runner FAQ, and more.

What is Software Testing?

Software testing is more than just error detection;

Testing software is operating the software under controlled conditions, to
(1) *verify* that it behaves "as specified";
(2) to *detect* *errors*, and
(3) to *validate* that what has been specified is what the user actually wanted.

1. *Verification* is the checking or testing of items, including software, for conformance and consistency by evaluating the results against pre-specified requirements. [*V**erification: Are we building the system right?*].

2. *Error Detection*: Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should.

3. *Validation* looks at the system correctness – i.e. is the process of checking that what has been specified is what the user actually wanted. [*Validation: Are we building the right system?*]

In other words, validation checks to see if we are building what the customer wants/needs, and verification checks to see if we are building that system correctly. Both verification and validation are necessary, but different components of any testing activity.

The definition of testing according to the ANSI/IEEE 1059 standard is that testing is the process of analysing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item. Remember: The purpose of testing is verification, validation and error detection in order to find problems – and the purpose of finding those problems is to get them fixed.

Software Testing
Testing involves operation of a system or application under controlled conditions and evaluating the results. Every Test consists of 3 steps :

Planning	Inputs to be given, results to be obtained and the process to proceed is to planned
Execution	Preparing test environment, Completing the test, and determining test results
Evaluation	compare the actual test outcome with what the correct outcome should have been

Automated Testing

Automated testing is as simple as removing the "human factor" and letting the computer do the thinking. This can be done with integrated debug tests, to much more intricate processes. The idea of the these tests is to find bugs that are often very challenging or time intensive for human testers to find. This sort of testing can save many man hours and can be more "efficient" in some cases. But it will cost more to ask a developer to write more lines of code into the game (or an external tool) then it does to pay a tester and there is always the chance there is a bug in the bug testing

program. Reusability is another problem; you may not be able to transfer a testing program from one title (or platform) to another. And of course, there is always the "human factor" of testing that can never truly be replaced.

Other successful alternatives or variation: Nothing is infallible. Realistically, a moderate split of human and automated testing can rule out a wider range of possible bugs, rather than relying solely on one or the other. Giving the testere limited access to any automated tools can often help speed up the test cycle

SQA India could be a useful resource for Software Quality assurance. We follow Software Quality Assurance (SQA) methods that adhere to the quality assurance at every phase of SDLC (Software Development Life Cycle). A quality control checklist is developed for all phases of SDLC. We aim to provide quality and consistent results through automated process that have been tested over time. Purpose of the SQA Plan is to establish a uniform Web / Software Development Process, which is applicable throughout the software development life cycle.

Welcome to the Software Quality Assurance (QA) and Testing Information Center. It offers articles, best practices, methodologies, FAQ, guides, books, directories, and tools that are related software QA and testing. Popular materials include job interview FAQ, test plan samples, Win Runner FAQ, and more.

Our Software Testing Procedure

A good test effort is driven by questions such as:

- How could this software break?
- In what possible situations could this software fail to work predictably?

Software testing challenges the assumptions, risks, and uncertainty inherent in the work of other disciplines, and addresses those concerns using concrete demonstration and impartial evaluation.

Testing focuses primarily on evaluating or assessing product quality, which is realized through the following core practices:

- Find and document defects in software quality.
- Advise on the perceived software quality.
- Validate and prove the assumptions made in design and requirement specifications through concrete demonstration.
- Validate that the software product works as designed.
- Validate that the requirements are implemented appropriately

Testing Methods

1. **White Box**
   Also called 'Structural Testing / Glass Box Testing' is used for testing the code keeping the system specs in mind. Inner working is considered and thus Developers Test..
   - **Mutation Testing**
     Number of mutants of the same program created with minor changes and none of their result should coincide with that of the result of the original program given same test case.
     Basic Path Testing
     Testing is done based on Flow graph notation, uses Cyclometric complexity & Graph matrices.
     Control Structure Testing
     The Flow of control execution path is considered for testing. It does also checks :-
     Conditional Testing : Branch Testing, Domain Testing.
     Data Flow Testing.
     Loop testing :Simple, Nested, Conditional, Unstructured Loops.
2. **Black Box**
   Also called 'Functional Testing' as it concentrates on testing of the functionality rather than the internal

details of code.
Test cases are designed based on the task descriptions

- **Comparison Testing**
  Test cases results are compared with the results of the test Oracle.
  Graph Based Testing
  Cause and effect graphs are generated and cyclometric complexity considered in using the test cases.
  Boundary Value Testing
  Boundary values of the Equivalence classes are considered and tested as they generally fail in Equivalence class testing.
  Equivalence class Testing
  Test inputs are classified into Equivalence classes such that one input check validates all the input values in that class.

Gray Box Testing : Similar to Black box but the test cases, risk assessments, and test methods involved in gray box testing are developed based on the knowledge of the internal data and flow structures

Levels of Testing

1. **Unit Testing.**
   - Unit Testing is primarily carried out by the developers themselves.
   - Deals functional correctness and the completeness of individual program units.
   - White box testing methods are employed
2. **Integration Testing.**
   - Integration Testing: Deals with testing when several program units are integrated.
   - **Regression testing** : Change of behavior due to modification or addition is called 'Regression'. Used to bring changes from worst to least.
   - **Incremental Integration Testing** : Checks out for bugs which encounter when a module has been integrated to the existing.
   - **Smoke Testing** : It is the battery of test which checks the basic functionality of program. If fails then the program is not sent for further testing.
3. **System Testing.**
   - System Testing - Deals with testing the whole program system for its intended purpose.
   - **Recovery testing** : System is forced to fail and is checked out how well the system recovers the failure.
   - **Security Testing** : Checks the capability of system to defend itself from hostile attack on programs and data.
   - **Load & Stress Testing** : The system is tested for max load and extreme stress points are figured out.
   - **Performance Testing** : Used to determine the processing speed.
   - **Installation Testing** : Installation & uninstallation is checked out in the target platform.
4. **Acceptance Testing.**
   - UAT ensures that the project satisfies the customer requirements.
   - **Alpha Testing** : It is the test done by the client at the developer's site.
   - **Beta Testing** : This is the test done by the end-users at the client's site.
   - **Long Term Testing** : Checks out for faults occurrence in a long term usage of the product.
   - **Compatibility Testing** : Determines how well the product is substantial to product transition

What is Software Testing?

Software testing is more than just error detection;

Testing software is operating the software under controlled conditions, to (1) *verify* that it behaves "as specified"; (2) to *detect* *errors*, and (3) to *validate* that what has been specified is what the user actually wanted.

1. *Verification* is the checking or testing of items, including software, for conformance and consistency by evaluating the results against pre-specified requirements. [*V**erification: Are we building the system right?*]
2. *Error Detection*: Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should.
3. *Validation* looks at the system correctness – i.e. is the process of checking that what has been specified is what the user actually wanted. [*Validation: Are we building the right system?*]

In other words, validation checks to see if we are building what the customer wants/needs, and verification checks to see if we are building that system correctly. Both verification and validation are necessary, but different components of any testing activity.

The definition of testing according to the ANSI/IEEE 1059 standard is that testing is the process of analysing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item. Remember: The purpose of testing is verification, validation and error detection in order to find problems – and the purpose of finding those problems is to get them fixed.

Software Testing

Testing involves operation of a system or application under controlled conditions and evaluating the results.
Every Test consists of 3 steps :
Planning : Inputs to be given, results to be obtained and the process to proceed is to planned.
Execution : preparing test environment, Completing the test, and determining test results.
Evaluation : compare the actual test outcome with what the correct outcome should have been.

Automated Testing

Automated testing is as simple as removing the "human factor" and letting the computer do the thinking. This can be done with integrated debug tests, to much more intricate processes. The idea of the these tests is to find bugs that are often very challenging or time intensive for human testers to find. This sort of testing can save many man hours and can be more "efficient" in some cases. But it will cost more to ask a developer to write more lines of code into the game (or an external tool) then it does to pay a tester and there is always the chance there is a bug in the bug testing program. Reusability is another problem; you may not be able to transfer a testing program from one title (or platform) to another. And of course, there is always the "human factor" of testing that can never truly be replaced.

Other successful alternatives or variation: Nothing is infallible. Realistically, a moderate split of human and automated testing can rule out a wider range of possible bugs, rather than relying solely on one or the other. Giving the testere limited access to any automated tools can often help speed up the test cycle.

Release Acceptance Test

The release acceptance test (RAT), also referred to as a build acceptance or smoke test, is run on each development release to check that each build is stable enough for further testing. Typically, this test suite consists of entrance and exit test cases plus test cases that check mainstream functions of the program with mainstream data. Copies of the RAT can be distributed to developers so that they can run the tests before submitting builds to the testing group. If a build does not pass a RAT test, it is reasonable to do the following:

- Suspend testing on the new build and resume testing on the prior build until another build is received.
- Report the failing criteria to the development team.
- Request a new build.

Functional Acceptance Simple Test

The functional acceptance simple test(FAST) is run on each development release to check that key features of the program are appropriately accessible and functioning properly on the at least one test configuration (preferable the minimum or common configuration).This test suite consists of

simple test cases that check the lowest level of functionality for each command- to ensure that task-oriented functional tests(TOFTs) cna be performed on the program. The objective is to decompose the functionality of a program down to the command level and then apply test cases to check that each command works as intended. No attention is paid to the combination of these basic commands, the context of the feature that is formed by these combined commands, or the end result of the overall feature. For example, FAST for a File/Save As menu command checks that the Save As dialog box displays. However, it does not validate that the overall file-saving feature works nor does it validate the integrity of save files.

Deployment Acceptance Test

The configuration on which the Web system will be deployed will often be much different from develop-and-test configurations. Testing efforts must consider this in the preparation and writing of test cases for installation time acceptance tests. This type of test usually includes the full installation of the applications to the targeted environments or configurations

Structural System Testing Techniques
Stress Testing:

Usage	" To determine if the system can function when subject to large volumes. " It includes testing of input transactions Internal tables Disk Space Out put Communication Computer capacity Interaction with people.
Objectives	" To simulate production environment " Normal or above normal volumes of transactions can be processed through the transaction within expected time frame. " Application system would be able to process larger volume of data. " System capacity should have sufficient resources to meet expected turnaround time.
How to Use	" It should simulate as closely as possible to production environment. " Online system should be stress tested with users entering test data with normal or above normal pace. " Batch system should be tested with huge volumes/ numbers of batches " The test conditions should have error conditions. " Transactions used in stress testing are obtained from following 3 sources : Test data generators Test transactions created by test group Transactions which were previously used in production. " In stress testing the system should run as it would in the production environment.
When to use	" When there is uncertainty that system will work with huge volumes of data and without generating any faults. " Attempt is made to break system with huge amount of data. " Most commonly used technique to test for online transaction systems as other techniques are not effective.
Examples	" Sufficient disk space allocated " Communication lines are adequate
Disadvantage	" Amount of time taken to prepare for testing " Amount of resources utilized during test execution.

Execution Testing:

Usage	" To determine whether the system achieves the desired level of proficiency in the production status. " Used to verify - Response time Turn around time Design performance. " Test execution can be done using the simulated system and actual system. " The system either can be tested as a whole or in parts.
Objectives	" To determine whether the system can meet the specific performance criteria " Verify whether system make optimum use of hardware and software. " Determining response time to online use requests " Determining transaction processing turnaround time.

How to Use	" Can be performed in any phase of SDLC " Used to evaluate single aspect of system " Executed in following manner - Using h/w and s/w monitor Simulation of functioning using simulation model Creating quick or dirty programs to evaluate approximate performance of completed system.
When to use	" Should be used early in SDLC " Should be performed when it is known that the results can be used to make changes to the system structure.
Examples	" Transaction turnaround time adequacy " Optimum use of h/w and s/w.

Task-Oriented Functional Test

The task-oriented functional test (TOFT) consists of positive test cases that are designed to verify program features by checking the task that each feature performs against specifications, user guides, requirements, and design documents. Usually, features are organized into list or test matrix format. Each feature is tested for:

- The validity of the task it performs with supported data conditions under supported operating conditions.
- The integrity od the task's end result
- The feature's integrity when used in conjunction with related features

Forced-Error Test

The forced-error test (FET) consists of negative test cases that are designed to force a program into error conditions. A list of all error messages thatthe program issues should be generated. The list is used as a baseline for developing test cases. An attempt is made to generate each error message in the list. Obviously, test to validate error-handling schemes cannot be performed until all the handling and error message have been coded. However, FETs should be thought through as early as possible. Sometimes, the error messages are not available. The error cases can still be considered by walking through the program and deciding how the program might fail in a given user interface such as a dialog or in the course of executing a given task or printing a given report. Test cases should be created for each condition to determine what error message is generated.

Real-world User-level Test

These tests simulate the actions customers may take with a program. Real-World user-level testing often detects errors that are otherwise missed by formal test types.

Exploratory Test

Exploratory Tests do not involve a test plan, checklist, or assigned tasks. The strategy here is to use past testing experience to make educated guesses about places and functionality that may be problematic. Testing is then focused on those areas. Exploratory testing can be scheduled. It can also be reserved for unforeseen downtime that presents itself during the testing process.

Compatibility and Configuration Testing

Compatibility and configuration testng is performanced to check that an application functions properly across various hardware and software environments. Often, the stragegy is to run the functional acceptance simple tests or a subset of the task-oriented functional tests on a range of software and hardware configurations. Sometimes, another strategy is to create a specific test that takes into account the error risks associated with configuration differences. For example, you might design an extensive series of tests to check for browser compatibility issues. Software compatibility configurations include variances in OS versions, input/output (I/O) devices, extension, network software, concurrent applications, online services and firewalls. Hardwere configurations include variances in manufacturers, CPU types, RAM, graphic display cards, video capture cards, sound cards, monitors, network cards, and connection types(e.g. T1, DSL, modem, etc..).

Documentation

Testing of reference guides and user guises check that all features are reasonably documented. Every page of documentation should be keystroke-tested for the following errors:

- Accuracy of every statement of fact
- Accuracy of every screen shot, figure and illustation
- Accuracy of placement of figures and illustation
- Accuracy of every tutorial, tip, and instruction
- Accuracy of marketing collateral (claims, system requirements,and screen shots)
- Accuracy of downloadable documentation(PDFs, htm, or test files)

Online Help Test

Online help tests check the accuracy of help contents, correctness of features in the help system, and functionality of the help system.

Install/uninstall Test

Web system often require both client-side and server-side installs. Testing of the installer checks that installed features function properly--including icons, support documentation , the README file, and registry keys. The test verifies that the correct directories are created and that the correct system files are copied to the appropriate directories. The test also confirms that various error conditions are detected and handled gracefully.

Testing of the uninstaller checks that the installed directories and files are appropriately removed, that configuration and system-related filea are also appropriately removed or modified, and that the operating environment is recovered in its original state.

User Interface Tests

Easy-of-use UI testing evaluates how intuitive a system is. Issues pertaining to navigation, usablility, commands, and accessibility are considered. User interface functionality testing examines how well a UI operates to specifications.

AREAS COVERED IN UI TESTING

- Usability
- Look and feel
- Navigation controls/navigation bar
- Instructional and technical information style
- Images
- Tables
- Navigation branching
- Accessibility

External Beta Testing

External beta testing offers developers their first glimpse at how users may actually interact with a program. Copies of the program or a test URL, sometimes accompanied with letter of instruction, are sent out to a group of volunteers who try out the program and respond to questions in the letter. Beta testing is black-box, real-world testing. Beta testing can be difficult to manage, and the feedback that it generates normally comes too late in the development process to contribute to improved usability and functionality. External beta-tester feedback may be reflected in a README file or deferred to future releases.

Security Tests

Security measures protect Web systems from both internal and external threats. E-commerce concerns and the growing popularity of Web-based applications have made security testing increasingly relevant. Security tests determine whether a company's security policies have been properly implemented; they evaluate the functionality of existing systems, not whether the security policies that have been implemented are appropriate.

PRIMARY COMPONENTS REQUIRING SECURITY TESTING

- Application software
- Database
- Servers
- Client workstations
- Networks

System-Level Test

System-level tests consists of batteris of tests that are designed to fully exercise a program as a whole and check that all elements of the integrated system function properly.

Functional System Testing

System tests check that the software functions properly from end-to-end. The components of the system include: A database, Web-enable application software modules, Web servers, Web-enabled application frameworks deploy Web browser software, TCP/IP networking routers, media servers to stream audio and video, and messaging services for email.

A common mistake of test professionals is to believe that they are conducting system tests while they are actually testing a single component of the system. For example, checking that the Web server returns a page is not a system test if the page contains only a static HTML page.

System testing is the process of testing an integrated hardware and software system to verify that the system meets its specified requirements. It verifies proper execution of the entire set of application components including interfaces to other applications. Project teams of developers and test analysts are responsible for ensuring that this level of testing is performed.

System testing checklist include question about:

- Functional completeness of the system or the add-on module
- Runtime behavior on various operating system or different hardware configurantions.
- Installability and configurability on various systems
- Capacity limitation (maximum file size, number of records, maximum number of concurrent users, etc.)
- Behavior in response to problems in the programming environment (system crash, unavailable network, full hard-disk, printer not ready)
- Protection against unauthorized access to data and programs.

"black-box" (or functional) testing

Black Box Testing is testing without knowledge of the internal workings of the item being tested. The Outside world comes into contact with the test items, --only through the application interface ,,, an internal module interface, or the INPUT/OUTPUT description of a batch process. They check whether interface definitions are adhered to in all situation and whether the product conform to all fixed requirements. Test cases are created based on the task descriptions.

Black Box Testing assumes that the tester does not know anything about the application that is going to be tested. The tester needs to understand what the program should do, and this is achieved through the business requirements and meeting and talking with users.

Funcional tests: This type of tests will evaluate a specific operating condition using inputs and validating results. Functional tests are designed to test boundaries. A combination of correst and incorrect data should be used in this type of test.

Scalability and Performance Testing

Scalability and performance testing is the way to understand how the system will handle the load cause by many concurrent users. In a Web environment concurrent use is measured as simply the number of users making requests at the same time.

Performance testing is designed to measure how quickly the program completes a given task. The primary objective is to determine whether the processing speed is acceptable in all parts of the program. If explicit requirements specify program performance, then performance test are often performed as acceptance tests.

As a rule, performance tests are easy to automate. This makes sense above all when you want to make a performance comparison of different system conditions while using the user interface. The capture and automatic replay of user actions during testing eliminates variations in response times.

This type of test should be designed to verify response and excution time. Bottlenecks in a system are generally found during this stage of testing.

Stress Testing

Overwhelm the product for performance, reliability, and efficiency assessment; To find the breakpoint when system is failure; to increase load regressively to gather information for finding out maximum concurrent users.

Stress tests force programs to operate under limited resource conditions. The goal is to push the upper functional limits of a program to ensure that it can function correctly and handle error conditions gracefully. Examples of resources that may be artificially manipulated to create stressful conditions include memory, disk space, and network bandwidth. If other memory-oriented tests are also planned, they should be performed here as part of the stress test suite. Stress tests can be automated.

Breakpoint:

the capabilites and weakness of the product:

- High volunmes of data
- Device connections
- Long transation chains

Stress Test Environment:

As you set up your testing environment for a stress test, you need to make sure you can answer the following questions:

- Will my test be able to support all the users and still maintain performance?
- Will my test be able to simulate the number of transactions that pass through in a matter of hours?
- Will my test be able to uncover whether the system will break?
- Will my server crash if the load continues over and over?

The test should be set up so that you can simulate the load; for example:

- If you have a remote Web site you should be able to monitor up to four Web sites or URLs.
- There should be a way to monitor the load intervals.
- The load test should be able to simulate the SSL (Secure Server)
- The test should be able to simulate when a user submits the Form Data (GET method)
- The test should be set up to simulate and authentical the keyword verification.
- The test should be able to simulate up to six email or pager mail addresses and an alert should occur when there is a failure.

It is important to remember when stressing your Web site to give a certain number of users a page to stress test and give them a certain amount of time in which to run the test.

Some of the key data features that can help you measure this type of stress test, determine the load, and uncover bottlenecks in the system are:

- Amount of memory available and used
- The processor time used
- The number of requests per second
- The amount of time it takes ASP pages to be set up.
- Server timing errors

Load Testing

- The process of modeling application usage conditions and performing them against the application and system under test, to analyze the application and system and determine capacity, throughout speed, transation handling capabilities, scalabilities and reliability while under under stress.
- This tyoe of test is designed to identify possible overloads to the system such as too many users signed on to the system, too many terminals on the network, and network system too slow.
- Load testing a simulation of how a browser will respond to intense use by many individuals. The Web sessions can be recorded live and set up so that the test can be run during peak times and also during slow times. The following are two different types of load tests:
- Single session - A single session should be set up on browser that will have one or multiple responses. The timeing of the data should be put in a file. After the test, you can set up a separate file for report analysis.

- Multiple session - a multiple session should be developed on multiple browsers with one or multiple responses. The multivariate statistical methods may be needed for a complex but general performance model
- When performing stress testing, looping transactions back on themselves so that the system stresses itself simulates stress loads and may be useful for finding synchronization problems and timing bugs, Web priority problems, memory bugs, and Windows problems using API. For example, you may want ot simulate an incoming message that is then put out on a looped-back line; this in turn will generate another incoming message. The nyou can use another system of comparable size to create the stress load.
- Memory leaks are often found under stress testing. A memory leak occurs when a test leaves allocated memory behind and does not correctly return the memory to the memory allocation scheme. The test seems to run correctly, but after several iteration available memory is reduced until the system fails.
- Peak Load and Testing Parameters:
- Determining your peak load is important before beginning the assessment of the Web site test. It may mean more than just using user requests per second to stress the system. There should be a combination of determinants such as requests per second , processor time, and memory usage. There is also the consideration of the type of information that is on your Web page from graphics and code processing, such as scripts, to ASP pages. Then it is important to determine what is fast and what is slow for your system. The type of connection can be a critical component here, such as T1 or T3 versus a modem hookup. After you have selected your threshold, you can stress your system to additional limits.
- As a tester you need to set up test parameters to make sure you can log the number of users coming into and leaving the test. This should be started in a small way and steadily increased. The test should also begin by selecting a test page that may not have a large amount of graphics and steadily increasing the complexity of the test by increasing the number of graphics and image requests. Keep in mind that images will take up additional bandwidth and resources on the server but do not really have a large impact on the server's processor.
- Another important item to remember is that you need to account for the length of time the user will spend surfing each page. As you test, you should set up a log to determine the approximate time spend on each page, whether it is 25 or 30 seconds. It may be recorded that each user spends at least 30 seconds on each page, and that will produce a heightened response for the server. As the request is queued, and this will be analyzed as the test continues

Load/Volume Test

Load/volume tests study how a program handles large amounts of data, excessive calculations, and excessive processing. These tests do not necessarily have to push or exceed upper functional limits. Load/volume tests can, and usually must, be automated.

Focus of Load/Volume Tesing

- Pushing through large amounts of data with extreme processing demands.
- Requesting many processes simulateously.
- Repeating tasks over a long period of time

Load/volume tests, which involve extreme conditions, are normally run after the execution of feature-level tests, which prove that a program functions correctly under normal conditions.

Difference between Load and Strees testing

The idea of stress testing is to find the breaking point in order to find bugs that will make that break potentially harmful. Load testing is merely testing at the highest transaction arrival rate in performance testing to see the resource contention, database locks etc..

Web Capacity Testing Load and Stress

The performance of the load or stress test Web site should be monitored with the following in mind:

- The load test should be able to support all browser
- The load test should be able to support all Web server.
- The tool should be able to simulate up 500 users or playback machines
- The tool should be able to run on WIndows NT, Linux, Solaris, and most Unix variants.
- There should be a way to simulate various users at different connection speeds.
- After the tests are run, you should be able to report the transactions, URL, and number of users who visited the site.
- The test cases should be asssembled in a like fashion to set up test suites.
- There should be a way to test the different server and port addresses.

- There should be a way to account for the user's cookies.

Performance Test

The primary goal of performance-testing is to develop effective enhancement strategies for maintaining acceptable system performance. Performance testing is a capacity analysis and planning process in which measurement data are used to predict when load levels will exhaust system resources.

The Mock Test

It is a good idea to set up s mock test before you begin your actual test. This is a way to measure the server's stressd performance. As you progress with your stress testing, you can set up a measurement of metrics to determine the efficiency of the test.

After the initial test, you can determine the breaking point for the server. It may be a processor problem or even a memory problem. You need to be able to check your log to determine the average amount of time that it takes your provessor to perform the test. Running graphics or even ASP pages can cause processor problems and a limitation every time you run your stress test.

Memory tends to be a problem with the stress test. This may be due to a memary leak or lack of memory. You need to log and monitor the amount of disk capacity during the stress test. As mentioned earlier, the bandwidth can account for the slow down of the processing of the Web site speed. If the test hanges and there is a large waiting period, your processor usage is too low to handle the a,ount of stress on the system.

Simulate Resources
It is important to be able to run system in a high-stress format so that you can actually simulate the resources and understand how to handle a specific load. For example, a bank transaction processing system may be designed to process up to 150 transactions per second, whereas an operating system may be designed to handle up to 200 separate terminals. The different tests need to be designed to ensure that the system can process the expected load. This type of testing usually involves planning a series of tests where the load is gradually increased to reflect the expected usage pattern. The stress tests can steadily increase the load on the system beyond the maximum design load until the system fails.

This type of testing has a dual function of testing the system for failure and looking for a combination of events that occur when a load is placed on the server. Stress testing can then determine if overloading the system results in loss of data or user sevice to the customers The use of stress testing is particularly relevant to an ecommerce system with Web database

Increas Capacity Testing

When you begin your stress testing, you will want to increase your capacity testing to make sure you are able to handle the increased load of data such as ASP pages and graphics. When you test the ASP pages, you may want to create a page similar to the original page that will simulate the same items on the ASP page and have it send the information to a test bed with a process that completes just a small data output. By doing this, you will have your processor still stressing the system but not taking up the bandwidth by sending the HTML code along the full path. This will not stress the entire code but will give you a basis from which to work. Dividing the requests per second by the total number of user or threads will determine the number of transactions per second. It will tell you at what point the server will start becoming less efficient at handling the load. Let's look at an example. Let's say your test with 50 users shows your server can handle 5 requests per seconf, with 100 users it is 10 requests per second, with 200 users it is 15 requests per second, and eventually with 300 users it is 20 requests per second. Your requests per second are continually climbing, so it seems that you are obtaining steadily improving performance. Let's look at the ratios:

05/50 = 0.1
10/100 = 0.1
15/200 = 0.075
20/300 = 0.073

From this example you can see that the performance of the server is becoming less and less efficient as the load grows. This in itself is not necessarily bad (as long as your pages are still returning within your target time frame). However, it can be a useful indicator during your optimization process and does give you some indication of how much leeway you have to handle expected peaks.

Stateful testing

When you use a Web-enabled application to set a value, does the server respond correctly later on?

Privilage testing

What happens when the everyday user tries to access a control that is authorized only for adminstrators?

Speed testing

Is the Web-enabled application taking too long to respond?

Boundary Test

Boundary tests are designed to check a program's response to extreme input values. Extreme output values are generated by the input values. It is important to check that a program handles input values and output results correctly at the lower and upper boundaries. Keep in mind that you can create extreme boundary results from non-extreme input values. It is essential to analyze how to generate extremes of both types. In addition. sometime you know that there is an intermediate variable involved in processing. If so, it is useful to determine how to drive that one through the extremes and special conditions such as zero or overflow condition

Boundary timeing testing

What happens when your Web-enabled application request times out or takes a really long time to respond?

Regression testing

Did a new build break an existing function? Repeat testing after changes for managing risk relate to product enhancement.

A regression test is performded when the tester wishes to see the progress of the testing processs by performing identical tests before and after a bug has been fixed. A regression test allows the tester to compare expeted test results with the actual results.

Regression testing's primary objective is to ensure that all bugfree features stay that way. In addition, bugs which have been fixed once should not turn up again in subsequent program versions.

Regression testing: After every software modification or before next release, we repeat all test cases to check if

fixed bugs are not show up again and new and existing functions are all working correctly.

Regression testing is used to confirm that fixed bugs have, in fact, been fixed and that new bugs have not been introduced in the process, and that festures that were proven correctly functional are intact. Depending on the size of a project, cycles of regression testing may be perform once per milestone or once per build. Some bug regression testing may also be performed during each accceptance test cycle, forcusing on only the most important bugs. Regression tests can be automated.

CONDITIONS DURING WHICH REGRESSION TESTS MAY BE RUN Issu fixing cycle. Once the development team has fixed issues, a regression test can be run t ovalidate the fixes. Tests are based on the step-by-step test casess that were originally reported:

- If an issue is confirmeded as fixed, then the issue report status should be changed to Closed.
- If an issue is confirmed as fixed, but with side effects, then the issue report status should be changed to Closed. However, a new issue should be filed to report the side effect.
- If an issue is only partially fixed, then the issue report resolution should be changed back to Unfixed, along with comments outlining the oustanding problems

Open-status regression cycle. Periodic regression tests may be run on all open issue in the issue-tracking database. During this cycle, issue status is confirmed either *the report is reproducible as is with no modification, the report is reproducible with additional comments or modifications, or the report is no longer reproducible*

Closed-fixed regression cycle. In the final phase of testing, a full-regression test cycle should be run to confirm the status of all fixed-closed issues.

Feature regression cycle. Each time a new build is cut or is in the final phase of testing depending on the organizational procedure, a full-regression test cycle should be run to confirm that the proven correctly functional features are still working as expected.

Database Testing

Items to check when testing a database

What to test	*Environment*	*toola/technique*
Seach results	System test environment	Black Box and White Box technique
Response time	System test environment	Sytax Testing/Functional Testing
Data integrity	Development environment	White Box testing
Data validity	Development environment	White Box testing

Query reaponse time
The turnaround time for responding to queries in a database must be short; therefor, query response time is essential for online transactions. The results from this test will help to identify problems, such as possible bottlenecks in the network, sspecific queries, the database structure, or the hardware.

Data integrity
Data stored in the database should include such items as the catalog, pricing, shipping tables, tax tables, order

database, and customer information. Testng must verify the integrity of the stored data. Testing should be done on a regular basis because data changes over time.

Data integrity tests

Data integrity can be tested as follows to ensure that the data is valid and not corrupt:

- Test the creation, modification, and deletion of data in tables as specified in the business requirement.
- Test to make sure that sets of radio buttons represent a fixed set of values. You should also check for NULL or EMPTY values.
- Test to make sure that data is save to the database and that each values gets saved fully. You should watch for the truncation of strings and that numeric values are not rounded off.
- Test to make sure that default values are stored and saved.
- Test the compatibility with old data. You should ensure that all updates do not affect the data you have on file in your database.

Data validity
The most common data errors are due to incorrect data entry, called data validity errors.

Recovery testing

- The system recovers from faukts and resumes processing within a predefined period of time.
- The system is fault-tolerant, which means that processing faults do not halt the overall functioning of the system.
- Data recovery and restart are correct in case of automatic recovery. If recovery requires human intervention, the mean time to repair the database is within predefined acceptable limits.

When testing a SQL server

- If the Web site publishes from inside the SQL Server straight to a Web page, is the data accurate and of the correct data type?
- If the SQL Server reads from a stored procedure to produce a Web page or if the stored procedure is changed, does the data on the page change?
- If you are using FrontPage or interDev is the data connection to your pages secure?
- Does the database have scheduled maintenance with a log so testers can set changes or errors?
- Can the tester check to see how back ups are being handled?
- Is the database secure?

When testing a Access database

- If the database is creating Web pages from the datbase to a URL, is the information correct and updated? If the pages are not dynamic or Active Server pages, they will not update automatically.
- If the tables in the database are linked to another database, make sure that all the links are active and giving reevant information.
- Are the fields such as zip code, phone numbers, dates, currency, and social security number formateed properly?
- If there are formulas in the database, do they work? How will they take care of updates if numbers change (for example, updating taxes)?
- Do the forms populate the correct tables?
- Is the database secure?

When test a FoxPro database

- If the database is linked to other database, are the links secure and working?
- If the database publishes to the Internet, is the data correct?
- When data is deployed, is it still accurate?
- Do the queries give accurate information to the reports?
- If thedatabase performs calculations, are the calculatons accurate?

Other important Database and security feature

- Credite Card Transaction

- Shopping Carts
- Payment Transaction Security

Secure Sockets Layer (SSL)

SSL is leading security protocol on the Internet. When an SSL session is started, the server sends its publice key to the browser, which the browser uses to send a randomly generated secret key back to the server to have a secret key exchange for that session.

SSL is a protocol that is submitted to the WWW consortium (W3C) working group on security for consideration as a standard security hanhshake that is used to initiate the TCP/IP connection. This handshake results in the client and server agreeing on the level of security that they will use, and this will fulfill any authentication requirements for the connection. SSL's role is to encrypt and decrypt the byte stream of the application protocol being used. This means the all the inofrmation in both the HTTP request and the HTTP response are fully encrypted, including the URL the client is requesting, any submitted form contents (such as credit card numbers), anty HTTP access authorization information (user names and passwords), and all the data returned from the server to the client.

Transport Layer Security (TLS)

TLS is a majo security standard on the internet. TLS is backward compatible with SSL and use Triple Data Encryption Standard (DES) encryption.

Three Waves of Software Development

Wave	***Type of application***	***Development Style***	***Test style***
1.	Desktop	Event-Driven framwork surrounds individual procedural functions The common style is to have a hierarchical set of menus presenting a set of commands.	Test each function agaist a written functions specification.
2.	Client/Server	Structured programming command organized into hierachical menu lists with each client function having its own code. The common style is to combine a common dropdown menu bar with graphical windows contain controls.	Test each function against a written functional specification and test the server for its throughput to server clients.
3.	Web-enabled	Visual Integrated development tools facilitate object-oriented design patterns. The common style is to provide multiple paths to accomplishing tasks. (The net effect is that you can't just walk through a set of hierarchical menus and arrive at the same test result any more.)	Capture/record/ playable watches how an application is used and then provides reports comparing how the playback differed from the original recording

Desktop application development and Test automation

The software was written to provide a friendly interface for information workers: Spreadsheet jockeys, business people needing written reports, and game players. The full spectrum of desktop software could pretty well be categorized into spreadsheet, wordprocessor, database, and entertainment categories since desktop computers were rarely networked to other information resources. Desktop applications used the keyboard, and then later a mouse, to navigate through windows and a drop-down menu. Inside a desktop application software package one would find an event-driven framework surrounding individual procedural functions. The automation focused on improving the time it took to test a desktop application for functionality. The test utilities link into desktop applications and try each command as though a user were accessing the menu and window commands. Most QA technicians testing a desktop application compare the function of all the menus and windows to a written functional specification document. The variation from the document to the performance shows the relative health of a desktop application.

Clicent/Server Development and Test automation

The original intent for client/server applications was to separete presentation logic from business logic. In an ideal system design, the client was reponsible for presenting the user interface, command elements (drop-down menus, buttons, controls), displayed results information in a set of windows, charts, and dials. The client connected to a server to process functions and the server responded with data.

In a client/server environment the protocols are cleanly defined so that all the clients use the same protocols to communicate with the server.

The client-side frameworks to provide the same functionality of desktop application frameworks plus most of the needed communication code to issue commands to the server and the code needed to automatically update a client with new functions received from the server.The server-side frameworks provide code needed to received and handle requests from multiple clients, and code to connect to database for data persistence and remote information providers. Additionally, these framworks need to handle stateful transations and intermittent network connections. Stateful transactions require multiple steps to accomplish a task.

Client/server applications are normally transactional in nature and usually several interactions with the user are needed to finish a single request. For example, in a stock trading application the user begins a transaction by identifying themselves to the server, looking up an order code, and then submitting a request to the server, and receives and presents the results to the user. The client-side application normally knows something about the transaction - for example, the client-side application will normally store the user identification and a session code such as cookie value across the user's interaction with the server-based application. Users like it better when the client-side application knows about the transaction because each step in a request can be optimized in the client application. For example. in the stock trading example the client application could calculate a stock trade commission locally without having to communication with server.

Client/server application test automation provides the functionality of desktop application test automation plus these:

Client/server applications operate in a network environment. The tests need to not only check for the function of an application, they need to test how the application handles slow or intermittent network performance.

Automated test are ideal to determine the number of client applications a server is able to efficiently handle at any given time.

- The server is usually a middle tier between the client application and several data sources. Automated tests need to check the server for correct functionality while it communicates with the data source.

Black-Box testing on Window-based Application

Editable Fields Checking and Validation:

- Valid/invalid characters/strings data in all editable fields
- Valid minimum/maximum/mid range values in fields
- Null strings (or no data) in required fields
- Record length (character limit)in text/memo fields
- Cut/copy/paste into/from fields when possible

Not Editable Fields Checking:

- Check for all test/spelling in warnings and error messages/dialogs
- Invoke/check all menu items and their options

Application Usability:

- Appearance an outlook (Placement an alignment of objects on screen)
- User Interface Test (open all menus, check all items)
- Basic functionality checking (File+Open+Save, etc..)
- Right mouse clicking sensitivity
- Resize/min/max/restore app, windows (check min app size)
- Scrollability when applicable (scrollbars, keyboard, autoscrolling)
- Keyboard and mouse navigation, highlighting, dragging, drag/drop
- Print in landscape an portrait modes
- Check F1, What's This , Help menu
- Short-cut and Accelerator keys
- Tab Key order and Navigation in all dialog boxes and menus

Web-Enabled Development and Test automation

Web-Enabled application go further in these areas:

- Web-enabled application are meant to be stateless. HTTP was designed to be stateless. Each request from a Web-enabled application is meant to be atomic and not rely on any previouse requests. This has huge advantages for system architecture and datacenter provisioning. When requests are stateless, then any sserver can respond to the request and any request handler on any server may service the request.
- Web-enabled application are platform independent. The client application may be written for Windows, Macintosh, Linux, and any other platform that is capable of implementing the command protocol and network connection to the server.
- Web-enabled application expect the client application to provide presentation rendering and simple scripting capabilities. The client application is usually a browser, however, it may also be a dedication client application such as a retail cash register, a Windows-based data analysis tool, ot an electronic address book in your mobile phone.

The missing context in a Web-enabled application test automation means that software developers and QA technicians must manually script tests for each Web-enalbled application. Plus, they need to maintain the test scriots as the application changes. Web-enabled application test automation tools focus on making the scriot writing and maintenance tasks easier. The test automation tool offer these features:

- A friendly, graphical user interface to integrate the record, edit, and run-time script functions.
- A recorder that watches how an application is used and writes a test script for you.
- A playback utility that drives a Web-enalbed application by processing the test script and logging. The playback utility also provides the facility to play back several concurrently running copies of the same script to check the system for scalability and load testing.
- A report utility to show how the playback differed from the original recording. The differences may be slower or faster performance times, errors, and incomplete transactions.

Testing Web Sites Applications

Many developers and testers are making the transition from testing traditional client/server, PC, and/or mainframe systems, to testing rapidly changing Web applications.

Web test: This testing forcuses on how well all parts of the web site hold together, whether inside and outside the website are working and whether all parts of the website are connected.

Web sites Server consist of three back-end layer:

- Web server
- Application server
- Data layers
- **FUNCTIONAL SYSTEM TESTING TECHNIQUES**
- **Requirements Testing Technique :**
- **Regression Testing Technique, :**

Usage	" To ensure that system performs correctly " To ensure that correctness can be sustained for a considerable period of time. " System can be tested for correctness through all phases of SDLC but incase of reliability the programs should be in place to make system operational.
Objectives	" Successfully implementation of user requirements " Correctness maintained over considerable period of time " Processing of the application complies with the organization's policies and procedures. " Secondary users needs are fulfilled - Security officer DBA Internal auditors Record retention Comptroller
How to Use	" Test conditions created These test conditions are generalized ones, which becomes test cases as the SDLC progresses until system is fully operational. Test conditions are more effective when created from user's requirements. Test conditions if created from documents then if there are any error in the documents those will get incorporated in Test conditions and testing would not be able to find those errors. Test conditions if created from other sources (other than documents) error trapping is effective. " Functional Checklist created.
When to	" Every application should be Requirement tested

use	" Should start at Requirements phase and should progress till operations and maintenance phase. " The method used to carry requirement testing and the extent of it is important.
Examples	" Creating test matrix to prove that system requirements as documented are the requirements desired by the user. " Creating checklist to verify that application complies to the organizational policies and procedures.

- **Regression Testing Technique, :**

Background	" One segment of system is developed and thoroughly tested. " Another segment is changed which has disastrous effect on the tested segment. " The implemented change, new data or parameters have created change in the already tested segment.
Usage	" All aspects of system remain functional after testing. " Change in one segment does not change the functionality of other segment.
Objectives	" Determine - system documents remain current System test data and test conditions remain current. Previously tested system functions properly without getting effected though changes are made in some other segment of application system.
How to Use	" Test cases, which were used previously for the already tested segment is, re-run to ensure that the results of the segment tested currently and the results of same segment tested earlier are same. " Test automation is needed to carry out the test transactions (test condition execution) else the process is very time consuming and tedious. " In this case of testing cost/benefit should be carefully evaluated else the efforts spend on testing would be more and payback would be minimum.
When to use	" When there is high risk that the new changes may effect the unchanged areas of application system. " In development process: Regression testing should be carried out after the pre-determined changes are incorporated in the application system. " In Maintenance phase : regression testing should be carried out if there is a high risk that loss may occur when the changes are made to the system
Examples	" Re-running of previously conducted tests to ensure that the unchanged portion of system functions properly. " Reviewing previously prepared system documents (manuals) to ensure that they do not get effected after changes are made to the application system. " Obtaining printout of data dictionary to ensure that the data elements reflect the changes being incorporated.
Disadvantage	" Time consuming and tedious if test automation not done.

Black Box testing for web-based application: (1)

1. Browser functionality:

- Is the browser compatible with the application design?
- There are many different types of browsers available.

GUI design components

- Are the scroll bars, buttons, and frames compatible with the browser and functional?
- To check the functionality of the scroll bars on the interface of the Web page to make sure the the user can scroll through items and make the correct selection from a list of items.
- The button on the interface need to be functional and the correct hyperlink should go to the correct page.
- If frames are used on the interface, they should be checked for the correct size and whether all of the components fit within the viewing screen of the monitor.

2. User Interface
One of the reasons the web browser is being used as the front end to applications is the ease of use. Users who have been on the web before will probably know how to navigate a well-built web site. While you are concentrating on this portion of testing it is important to verify that the application is easy to use. Many will believe that this is the least important area to test, but if you want to be successful, the site better be easy to use.

3.Instructions
You want to make sure there are instructions. Even if you think the web site is simple, there will always be someone who needs some clarification. Additionally, you need to test the documentation to verify that the instructions are correct. If you follow each instruction does the expected result occur?

4. Site map or navigational bar
Does the site have a map? Sometimes power users know exactly where they want to go and don't want to wade through lengthy introductions. Or new users get lost easily. Either way a site map and/or an ever-present navigational bar can help guide the user. You need to verify that the site map is correct. Does each link on the map actually exist? Are there links on the site that are not represented on the map? Is the navigational bar present on every screen? Is it consistent? Does each link work on each page? Is it organized in an intuitive manner?

5. Content
To a developer, functionality comes before wording. Anyone can slap together some fancy mission statement later, but while they are developing, they just need some filler to verify alignment and layout. Unfortunately, text produced like this may sneak through the cracks. It is important to check with the public relations department on the exact wording of the content.
You also want to make sure the site looks professional. Overuse of bold text, big fonts and blinking (ugh) can turn away a customer quickly. It might be a good idea to consult a graphic designer to look over the site during User Acceptance Testing. You wouldn't slap together a brochure with bold text everywhere, so you want to handle the web site with the same level of professionalism.
Finally, you want to make sure that any time a web reference is given that it is hyperlinked. Plenty of sites ask you to email them at a specific address or to download a browser from an address. But if the user can't click on it, they are going to be annoyed.

6. Colors/backgrounds
Ever since the web became popular, everyone thinks they are graphic designers. Unfortunately, some developers are more interested in their new backgrounds, than ease of use. Sites will have yellow text on a purple picture of a fractal pattern. (If you've never seen this, try most sites at GeoCities or AOL.) This may seem "pretty neat", but it's not easy to use.
Usually, the best idea is to use little or no background. If you have a background, it might be a single color on the left side of the page, containing the navigational bar. But, patterns and pictures distract the user.

Black Box testing for web-based application: (2)

7. Images
Whether it's a screen grab or a little icon that points the way, a picture is worth a thousand words. Sometimes, the best way to tell the user something is to simply show them. However, bandwidth is precious to the client and the server, so you need to conserve memory usage. Do all the images add value to each page, or do they simply waste bandwidth? Can a different file type (.GIF, .JPG) be used for 30k less?
In general, you don't want large pictures on the front page, since most users who abandon a page due to a large load will do it on the front page. If you can get them to see the front page quickly, it will increase the chance they will stay.

8. Tables
You also want to verify that tables are setup properly. Does the user constantly have to scroll right to see the price of the item? Would it be more effective to put the price closer to the left and put miniscule details to the right? Are the columns wide enough or does every row have to wrap around? Are certain columns considerably longer than others?

9. Wrap-around
Finally, you will want to verify that wrap-around occurs properly. If the text refers to "a picture on the right", make sure the picture is on the right. Make sure that widowed and orphaned sentences and paragraphs don't layout in an awkward manner because of pictures.

10. Functionality
The functionality of the web site is why your company hired a developer and not just an artist. This is the part that interfaces with the server and actually "does stuff".

11. Links
A link is the vehicle that gets the user from page to page. You will need to verify two things for each link: that the link brings you to the page it said it would and that the pages you are linking to actually exists. It may sound a little silly but I have seen plenty of web sites with internal broken links.

12. Forms
When a user submits information through a form it needs to work properly. The submit button needs to work. If the form is for an online registration, the user should be given login information (that works) after successful completion. If the form gathers shipping information, it should be handled properly and the customer should receive their package. In order to test this, you need to verify that the server stores the information properly and that systems down the line can interpret and use that information.

13. Data verification

If the system verifies user input according to business rules, then that needs to work properly. For example, a State field may be checked against a list of valid values. If this is the case, you need to verify that the list is complete and that the program actually calls the list properly (add a bogus value to the list and make sure the system accepts it).

14. Cookies
Most users only like the kind with sugar, but developers love web cookies. If the system uses them, you need to check them. If they store login information, make sure the cookies work. If the cookie is used for statistics, verify that totals are being counted properly. And you'll probably want to make sure those cookies are encrypted too, otherwise people can edit their cookies and skew your statistics. Application specific functional requirements Most importantly, you want to verify the application specific functional requirements. Try to perform all functions a user would: place an order, change an order, cancel an order, check the status of the order, change shipping information before an order is shipped, pay online, ad naseum.
This is why your users will show up on your doorstep, so you need to make sure you can do what you advertise

Black Box testing for web-based application: (3)

16. Interface Testing
Many times, a web site is not an island. The site will call external servers for additional data, verification of data or fulfillment of orders.

16. Server interface
The first interface you should test is the interface between the browser and the server. You should attempt transactions, then view the server logs and verify that what you're seeing in the browser is actually happening on the server. It's also a good idea to run queries on the database to make sure the transaction data is being stored properly.

17. External interfaces
Some web systems have external interfaces. For example, a merchant might verify credit card transactions real-time in order to reduce fraud. You will need to send several test transactions using the web interface. Try credit cards that are valid, invalid, and stolen. If the merchant only takes Visa and MasterCard, try using a Discover card. (A script can check the first digit of the credit card number: 3 for American Express, 4 for Visa, 5 for MasterCard, or 6 for Discover, before the transaction is sent.) Basically, you want to make sure that the software can handle every possible message returned by the external server.

18. Error handling
One of the areas left untested most often is interface error handling. Usually we try to make sure our system can handle all of our errors, but we never plan for the other systems' errors or for the unexpected. Try leaving the site mid-transaction - what happens? Does the order complete anyway? Try losing the internet connection from the user to the server. Try losing the connection from the server to the credit card verification server. Is there proper error handling for all these situations? Are charges still made to credit cards? Is the interruption is not user initiated, does the order get stored so customer service reps can call back if the user doesn't come back to the site?

19. Compatibility
You will also want to verify that the application can work on the machines your customers will be using. If the product is going to the web for the world to use, you will need to try different combinations of operating system, browser, video setting and modem speed.

20. Operating systems
Does the site work for both MAC and IBM-Compatibles? Some fonts are not available on both systems, so make sure that secondary fonts are selected. Make sure that the site doesn't use plug-ins only available for one OS, if your users will use both.

21. Browsers
Does your site work with Netscape? Internet Explorer? Lynx? Some HTML commands or scripts only work for certain browsers. Make sure there are alternate tags for images, in case someone is using a text browser. If you're using SSL security, you only need to check browsers 3.0 and higher, but verify that there is a message for those using older browsers.

22. Video settings
Does the layout still look good on 640x400 or 600x800? Are fonts too small to read? Are they too big? Does all the text and graphic alignment still work?

23. Modem/connection speeds
Does it take 10 minutes to load a page with a 28.8 modem, but you tested hooked up to a T1? Users will expect long download times when they are grabbing documents or demos, but not on the front page. Make sure that the images aren't too large. Make sure that marketing didn't put 50k of font size -6 keywords for search engines.

Black Box testing for web-based application: (4)

23. Printers
Users like to print. The concept behind the web should save paper and reduce printing, but most people would rather read on paper than on the screen. So, you need to verify that the pages print properly. Sometimes images and text align on the screen differently than on the printed page. You need to at least verify that order confirmation screens can be printed properly.

24. Combinations
Now you get to try combinations. Maybe 600x800 looks good on the MAC but not on the IBM. Maybe IBM with

Netscape works, but not with Lynx.
If the web site will be used internally it might make testing a little easier. If the company has an official web browser choice, then you just need to verify that it works for that browser. If everyone has a T1 connection, then you might not need to check load times. (But keep in mind, some people may dial in from home.) With internal applications, the development team can make disclaimers about system requirements and only support those systems setups. But, ideally, the site should work on all machines so you don't limit growth and changes in the future.

25. Load/Stress
You will need to verify that the system can handle a large number of users at the same time, a large amount of data from each user, and a long period of continuous use. Accessibility is extremely important to users. If they get a "busy signal", they hang up and call the competition. Not only must the system be checked so your customers can gain access, but many times crackers will attempt to gain access to a system by overloading it. For the sake of security, your system needs to know what to do when it's overloaded and not simply blow up.
Many users at the same time
If the site just put up the results of a national lottery, it better be able to handle millions of users right after the winning numbers are posted. A load test tool would be able to simulate large number of users accessing the site at the same time.
Large amount of data from each user
Most customers may only order 1-5 books from your new online bookstore, but what if a university bookstore decides to order 5000 different books? Or what if grandma wants to send a gift to each of her 50 grandchildren for Christmas (separate mailing addresses for each, of course.) Can your system handle large amounts of data from a single user?
Long period of continuous use
If the site is intended to take orders for flower deliveries, then it better be able to handle the week before Mother's Day. If the site offers web-based email, it better be able to run for months or even years, without downtimes.
You will probably want to use an automated test tool to implement these types of tests, since they are difficult to do manually. Imagine coordinating 100 people to hit the site at the same time. Now try 100,000 people. Generally, the tool will pay for itself the second or third time you use it. Once the tool is set up, running another test is just a click away.

26. Security
Even if you aren't accepting credit card payments, security is very important. The web site will be the only exposure some customers have to your company. And, if that exposure is a hacked page, they won't feel safe doing business with you

Black Box testing for web-based application: (5)

27. Directory setup
The most elementary step of web security is proper setup of directories. Each directory should have an index.html or main.html page so a directory listing doesn't appear.
One company I was consulting for didn't observe this principal. I right clicked on an image and found the path "...com/objects/images". I went to that directory manually and found a complete listing of the images on that site. That wasn't too important. Next, I went to the directory below that: "...com/objects" and I hit the jackpot. There were plenty of goodies, but what caught my eye were the historical pages. They had changed their prices every month and kept the old pages. I browsed around and could figure out their profit margin and how low they were willing to go on a contract. If a potential customer did a little browsing first, they would have had a definite advantage at the bargaining table.
SSL Many sites use SSL for secure transactions. You know you entered an SSL site because there will be a browser warning and the HTTP in the location field on the browser will change to HTTPS. If your development group uses SSL you need to make sure there is an alternate page for browser with versions less than 3.0, since SSL is not compatible with those browsers. You also need to make sure that there are warnings when you enter and leave the secured site. Is there a timeout limit? What happens if the user tries a transaction after the timeout?

28 Logins
In order to validate users, several sites require customers to login. This makes it easier for the customer since they don't have to re-enter personal information every time. You need to verify that the system does not allow invalid usernames/password and that it does allow valid logins. Is there a maximum number of failed logins allowed before the server locks out the current user? Is the lockout based on IP? What if the maximum failed login attempts is three, and you try three, but then enter a valid login? What are the rules for password selection?

29. Log files
Behind the scenes, you will need to verify that server logs are working properly. Does the log track every transaction? Does it track unsuccessful login attempts? Does it only track stolen credit card usage? What does it store for each transaction? IP address? User name?

30. Scripting languages
Scripting languages are a constant source of security holes. The details are different for each language. Some exploits allow access to the root directory. Others allow access to the mail server. Find out what scripting

languages are being used and research the loopholes. It might also be a good idea to subscribe to a security newsgroup that discusses the language you will be testing.

31. Web Server Testing Features

- Feature: *Definition*
- Transactions: *The nunber of times the test script requested the current URL*
- Elapsed time: *The number of seconds it took to run the request*
- Bytes transferred: *The total number of bytes sent or received, less HTTP headers*
- Response time: *The average time it took for the server to respond to each individual request.*
- Transaction rate: *The average number of transactions the server was able to handle per second.*
- Transferance: *The average number of bytes transferred per second.*
- Concurrency: *The average number of simultaneous connections the server was able to handle during the test session.*
- Status code nnn: *This indicates how many times a particular HTTP status code was seen.*

Automated Software Testing

Mercury Quick Test Professional (QTP) Tutorial
Stress Testing of Websites
Load Testing of Websites
Test Director Tutorial
Win Runner 7.0 Tutorial
Rational Suite Tutorial
Silk Test Tutorial
Rational Robot Tutorial
Perl Test Tutorial

Check List for Software Testing

Win Runner Questions and Answers
Quick Test Professional (QTP) Questions and Answers
ORACLE Questions and Answers
PeopleSoft Interview Questions and Answers
Silk Test Questions and Answers

Web Testing Checklist about Compatibility and Portability

Overall
1. Are requirements driven by business needs and not technology?

Audience
1. Has the audience been defined?
2. Is there a process for identifying the audience?
3. Is the process for identifying the audience current?
4. Is the process reviewed periodically?
5. Is there appropriate use of audience segmentation?
6. Is the application compatible with the audience experience level?
7. Where possible, has the audience readiness been ensured?
8. Are text version and/or upgrade links present?

Testing Process
1. Does the testing process include appropriate verifications? (e.g., reviews, inspections and walkthroughs)
2. Is the testing environment compatible with the operating systems of the audience?
3. Does the testing process and environment legitimately simulate the real world?

Operating systems Environment/ Platform
1. Has the operating environments and platforms been defined?
2. Have the most critical platforms been identified?
3. Have audience expectations been properly managed?
4. Have the business users/marketing been adequately prepared for what will be tested?
5. Have sign-offs been obtained?

Risk
1. Has the risk tolerance been assessed to identify the vital few platforms to test?

Hardware
1. Is the test hardware compatible with all screen types, sizes, resolution of the audience?
2. Is the test hardware compatible with all means of access, modems, etc of the audience?
3. Is the test hardware compatible will all languages of the audience?
4. Is the test hardware compatible with all databases of the audience?
5. Does the test hardware contain the compatible plug-ins and DLLs of the audience?

General
1. Is the application compatible with standards and conventions of the audience?
2. Is the application compatible with copyright laws and licenses?

Web Testing Checklist about Security

Access Control
1. Is there a defined standard for login names/passwords?
2. Are good aging procedures in place for passwords?
3. Are users locked out after a given number of password failures?
4. Is there a link for help (e.g., forgotten passwords?)
5. Is there a process for password administration?
6. Have authorization levels been defined?
7. Is management sign-off in place for authorizations?

Disaster Recovery
1. Have service levels been defined. (e.g., how long should recovery take?)
2. Are fail-over solutions needed?
3. Is there a way to reroute to another server in the event of a site crash?
4. Are executables, data, and content backed up on a defined interval appropriate for the level of risk?
5. Are disaster recovery process & procedures defined in writing? If so, are they current?
6. Have recovery procedures been tested?
7. Are site assets adequately Insured?
8. Is a third party "hot-site' available for emergency recovery?
9. Has a Business Contingency Plan been developed to maintain the business while the site is being restored?
10. Have all levels in organization gone through the needed training & drills?
11. Do support notification procedures exist & are they followed?
12. Do support notification procedures support a 24/7 operation?
13. Have criteria been defined to evaluation recovery completion / correctness?

Firewalls
1. Was the software installed correctly?
2. Are firewalls installed at adequate levels in the organization and architecture? (e.g., corporate data, human resources data, customer transaction files, etc.)
3. Have firewalls been tested? (e.g., to allow & deny access).
4. Is the security administrator aware of known firewall defects?
5. Is there a link to access control?
6. Are firewalls installed in effective locations in the architecture? (e.g., proxy servers, data servers, etc.)

Proxy Servers
1. Have undesirable / unauthorized external sites been defined and screened out? (e.g. gaming sites, etc.)
2. Is traffic logged?
3. Is user access defined?

Privacy
1. Is sensitive data restricted to be viewed by unauthorized users?
2. Is proprietary content copyrighted?
3. Is information about company employees limited on public web site?
4. Is the privacy policy communicated to users and customers?
5. Is there adequate legal support and accountability of privacy practices?

Web Testing Checklist about Security

Data Security
1. Are data inputs adequately filtered?
2. Are data access privileges identified? (e.g., read, write, update and query)
3. Are data access privileges enforced?
4. Have data backup and restore processes been defined?
5. Have data backup and restore processes been tested?
6. Have file permissions been established?
7. Have file permissions been tested?
8. Have sensitive and critical data been allocated to secure locations?
9. Have date archival and retrieval procedures been defined?
10. Have date archival and retrieval procedures been tested?

Monitoring
1. Are network monitoring tools in place?
2. Are network monitoring tool working effectively?
3. Do monitors detect
- Network time-outs?
- Network concurrent usage?
- IP spoofing?
4. Is personnel access control monitored?
5. Is personnel internet activity monitored?
- Sites visited
- Transactions created
- Links accessed

Security Administration
1. Have security administration procedures been defined?
2. Is there a way to verify that security administration procedures are followed?
3. Are security audits performed?
4. Is there a person or team responsible for security administration?
5. Are checks & balances in place?
6. Is there an adequate backup for the security administrator?

Encryption
1. Are encryption systems/levels defined?
2. Is there a standard of what is to be encrypted?
3. Are customers compatible in terms of encryption levels and protocols?
4. Are encryption techniques for transactions being used for secured transactions?
- Secure socket layer (SSL)
- Virtual Private Networks (VPNs)
5. Have the encryption processes and standards been documented?

Viruses
1. Are virus detection tools in place?
2. Have the virus data files been updated on a current basis?
3. Are virus updates scheduled?
4. Is a response procedure for virus attacks in place?
5. Are notification of updates to virus files obtained from anti-virus software vendor?
6. Does the security administrator maintain an informational partnership with the anti-virus software vendor?
7. Does the security administrator subscribe to early warning e-mail services? (e.g., www.fooorg or www.bar.net)
8. Has a key contact been defined for the notification of a virus presence?
9. Has an automated response been developed to respond to a virus presence?
10. Is the communication & training of virus prevention and response procedures to users adequate?

Web Testing Checklist about Performance

Tools
1. Are virus detection tools in place?
2. Have the virus data files been updated on a current basis?
3. Are virus updates scheduled?
4. Is a response procedure for virus attacks in place?
5. Are notification of updates to virus files obtained from anti-virus software vendor?
6. Does the security administrator maintain an informational partnership with the anti-virus software vendor?
7. Does the security administrator subscribe to early warning e-mail services? (e.g., www.foo.org or www.bar.net)
8. Has a key contact been defined for the notification of a virus presence?

9. Has an automated response been developed to respond to a virus presence?
10. Is the communication & training of virus prevention and response procedures to users adequate?

Tools
1. Has a load testing tool been identified?
2. Is the tool compatible with the environment?
3. Has licensing been identified?
4. Have external and internal support been identified?
5. Have employees been trained?

Number of Users
1. Have the maximum number of users been identified?
2. Has the complexity of the system been analyzed?
3. Has the user profile been identified?
4. Have user peaks been identified?
5. Have languages been identified?, i.e. English, Spanish, French, etc. for global wide sites
6. Have the length of sessions been identified by the number of users?
7. Have the number of users configurations been identified?

Expectations/Requirements
1. Have the response time been identified?
2. Has the client response time been identified?
3. Has the expected vendor response time been identified?
4. Have the maximum and acceptable response times been defined?
5. Has response time been met at the various thresholds?
6. Has the break point been identified been identified for capacity planning?
7. Do you know what caused the crash if the application was taken to the breaking point?
8. How many transactions for a given period of time have been identified (bottlenecks)?
9. Have availability of service levels been defined?

Architecture
1. Has the database campacity been identified?
2. Has anticipated growth data been obtained?
3. Is the database self-contained?
4. Is the system architecture defined?
" Tiers
" Servers
" Network
5. Has the anticipated volume for initial test been defined - with allowance for future growth?
6. Has plan for vertical growth been identified?
7. Have the various environments been created?
8. Has historical experience with the databases and equipment been documented?
9. Has the current system diagram been developed?
10.Is load balancing available?
11.Have the types of programming languages been identified?
12.Can back end processes be accessed

Web Testing Checklist about Performance

Resources
1. Are people with skill sets available?
2. Have the following skill sets been acquired?
" DBA
" Doc
" BA
" QA
" Tool Experts
" Internal and external support
" Project manager
" Training

Time Frame
1. When will the application be ready for performance testing?
2. How much time is available for performance testing?
3. How many iterations of testing will take place?

Test Environment
1. Does the test environment exist?
2. Is the environment self-contained?
3. Can one iteration of testing be performed in production?
4. Is a copy of production data available for testing?
5. Are end-users available for testing and analysis?

6. Will the test use virtual users?
7. Does the test environment mirror production?
8. Have the differences documented? (constraints)
9. Is the test available after production?
10. Have version control processes been used to ensure the correct versions of applications and data in the test environment?
11. Have the times been identified when you will receive the test data (globally) time frame?
12. Are there considerations for fail-over recovery? Disaster recovery?
13. Are replacement servers available?
14. Have back-up procedures been written?

Web Testing Checklist about Correctness

Data

1. Does the application write to the database properly?
2. Does the application record from the database correctly?
3. Is transient data retained?
4. Does the application follow concurrency rules?
5. Are text fields storing information correctly?
6. Is inventory or out of stock being tracked properly?
7. Is there redundant info within web site?
8. Is forward/backward cashing working correctly?
9. Are requirements for timing out of session met?

Presentation

1. Are the field data properly displayed?
2. Is the spelling correct?
3. Are the page layouts and format based on requirements?
(e.g., visual highlighting, etc.)
4. Does the URL show you are in secure page?
5. Is the tab order correct on all screens?
6. Do the interfaces meet specific visual standards (internal)?
7. Do the interfaces meet current GUI standards?
8. Do the print functions work correctly?

Navigation

1. Can you navigate to the links correctly?
2. Do Email links work correctly?

Functionality

1. Is the application recording the number of hits correctly?
2. Are calculations correct?
3. Are edits rules being consistently applied?
4. Is the site listed on search engines properly?
5. Is the help information correct?
6. Do internal searches return correct results?
7. Are follow-up confirmations sent correctly?
8. Are errors being handled correctly?
9. Does the application properly interface with other applications?

Web Testing Checklist about Correctness

Environment

- Are user sessions terminated properly?
- Is response time adequate based upon specifications?
- Is a complete software requirements specification available?
- Are requirements bounded?
- Have equivalence classes been defined to exercise input?
- Have boundary tests been derived to exercise the software at its boundaries.
- Have test suites been developed to validate each software function?
- Have test suites been developed to validate all data structures?
- Have test suites been developed to assess software performance?
- Have test suites been developed to test software behavior?
- Have test suites been developed to fully exercise the user interface?
- Have test suites been developed to exercise all error handling?

- Are use-cases available to perform scenario testing?
- Is statistical use testing (SEPA, 5/e, Chapter 26) being considered as an element of validation?
- Have tests been developed to exercise the software against procedures defined in user documentation and help facilities?
- Have error reporting and correction mechanisms been established?
- Has a deficiency list been created?

Check list for Conducting Unit

- Is the number of input parameters equal to number of arguments?
- Do parameter and argument attributes match?
- Do parameter and argument units system match?
- Is the number of arguments transmitted to called modules equal to number of parameters?
- Are the attributes of arguments transmitted to called modules equal to attributes of parameters?
- Is the units system of arguments transmitted to called modules equal to units system of parameters?
- Are the number of attributes and the order of arguments to built-in functions correct?
- Are any references to parameters not associated with current point of entry?
- Have input only arguments altered?
- Are global variable definitions consistent across modules?
- Are constraints passed as arguments?
- When a module performs external I/O, additional interface tests must be conducted.
- File attributes correct?
- OPEN/CLOSE statements correct?
- Format specification matches I/O statement?
- Buffer size matches record size?
- Files opened before use?
- End-of-file conditions handled?
- Any textual errors in output information?
- improper or inconsistent typing
- erroneous initialization or default values
- incorrect (misspelled or truncated) variable names
- inconsistent data types
- underflow, overflow and addressing exceptions
- Has the component interface been fully tested?
- Have local data structured been exercised at their boundaries?
- Has the cyclomatic complexity of the module been determined?
- Have all independent basis paths been tested?
- Have all loops been tested appropriately?
- Have data flow paths been tested?
- Have all error handling paths been tested?

General

- Pages fit within the resolution(800x600)
- Design works with liquid tables to fill the user's window size.
- Separate print versions provided for long documents (liquid tables may negate this necessity). Accommodates A4 size paper.
- Site doesn't use frames.
- Complex tables are minimized.

- Newer technologies are generally avoided for 1-2 years from release, or if used alternative traditional forms of content are easily available.

Home vs. Subsequent Pages & Sections

- Home page logo is larger and more centrally placed than on other pages.
- Home page includes navigation, summary of news/promotions, and a search feature.
- Home page answers: Where am I; What does this site do; How do I find what I want?
- Larger navigation space on home page, smaller on subsequent pages.
- Logo is present and consistently placed on all subsequent pages (towards upper left hand corner).
- "Home" link is present on all subsequent pages (but not home page).
- If subsites are present, each has a home page, and includes a link back to the global home page.

Navigation

- Navigation supports user scenarios gathered in the User Task Assessment phase (prior to design).
- Users can see all levels of navigation leading to any page.
- Breadcrumb navigation is present (for larger and some smaller sites).
- Site uses DHTML pop-up to show alternative destinations for that navigation level.
- Navigation can be easily learned.
- Navigation is consistently placed and changes in response to rollover or selection.
- Navigation is available when needed (especially when the user is finished doing something).
- Supplimental navigation is offered appropriately (links on each page, a site map/index, a search engine).
- Navigation uses visual hierarchies like movement, color, position, size, etc., to differentiate it from other page elements.
- Navigation uses precise, descriptive labels in the user's language. Icon navigation is accompanied by text descriptors.
- Navigation answers: Where am I (relative to site structure); Where have I been (obvious visited links); Where can I go (embedded, structural, and associative links)?
- Redundant navigation is avoided.

Check list about General

Functional Items

- Terms like "previous/back" and "next" are replaced by more descriptive labels indicating the information to be found.
- Pull-down menus include a go button.
- Logins are brief.
- Forms are short and on one page (or demonstrate step X of Y, and why collecting a larger amount of data is important and how the user will benefit).
- Documentation pages are searchable and have an abundance of examples. Instructions are task-oriented and step-by-step. A short conceptual model of the system is available, including a diagram that explains how the different parts work together. Terms or difficult concepts are linked to a glossary.

Linking

- Links are underlined.
- Size of large pages and multi-media files is indicated next to the link, with estimated dowload times.
- Important links are above the fold.
- Links to releated information appear at bottom of content or above/near the top.
- Linked titles make sense out of context.
- If site requires registration or subscription, provides special URLs for free linking. Indicates the pages are freely linkable, and includes and easy method to discover the URL.
- If site is running an ad, it links to a page with the relevant content, not the corporate home page.
- Keeps linked phrases short to aid scanning (2-4 words).

- Links on meaningful words and phrases. Avoids phrases like, "click here."
- Includs a brief description of what the user should expect on the linked page. In code:
- Uses relative links when linking between pages in a site. Uses absolute links to pages on unrelated sites.
- Uses link titles in the code for IE users (preferably less than 60 characters, no more than 80).

Search Capabilities

- A search feature appears on every page (exceptions include pop-up forms and the like).
- Search box is wide to allow for visible search parameters.
- Advanced Search, if included, is named just that (to scare off novices).
- Search system performs a spelling check and offers synonym expansion.
- Site avoids scoped searching. If included it indicates scope at top of both query and results pages, and additionally offers an automatic extended site search immediately with the same parameters.
- Results do not include a visible scoring system.
- Eliminates duplicate occurances of the same results (e.g., foo.com/bar vs. foo.com/bar/ vs. foo.com/bar/index.html).

Check list about General

Page Design

- Content accounts for 50% to 80% of a page's design (what's left over after logos, navigation, non-content imagery, ads, white space, footers, etc.).
- Page elements are consistent, and important information is above the fold.
- Pages load in 10 seconds or less on users bandwidth.
- Pages degrade adequately on older browsers.
- Text is over plain background, and there is high contrast between the two.
- Link styles are minimal (generally one each of link, visited, hover, and active states). Additional link styles are used only if necessary.
- Specified the layout of any liquid areas (usually content) in terms of percentages.

Fonts and Graphics

- Graphics are properly optimized.
- Text in graphics is generally avoided.
- Preferred fonts are used: Verdana, Arial, Geneva, sans-serif.
- Fonts, when enlarged, don't destroy layout.
- Images are reused rather than rotated.
- Page still works with graphics turned off.
- Graphics included are necessary to support the message.
- Fonts are large enough and scalable.
- Browser chrome is removed from screen shots.
- Animation and 3D graphics are generally avoided.

Content Design

- Uses bullets, lists, very short paragraphs, etc. to make content scannable.
- Articles are structured with scannable nested headings.
- Content is formatted in chunks targeted to user interest, not just broken into multiple pages.
- No moving text; most is left-justified; sans-serif for small text; no upper-case sentences/paragraphs; italics and bold are used sparingly.
- Dates follow the international format (year-month-day) or are written out (August 30, 2001).

Writing

- Writing is brief, concise, and well edited.
- Information has persistent value.
- Avoids vanity pages.
- Starts each page with the conclusion, and only gradually added the detail supporting that conclusion.

- One idea per paragraph.
- Uses simple sentence structures and words.
- Gives users just the facts. Uses humor with caution.
- Uses objective language.

Check list about General

Folder Structure

- Folder names are all lower-case and follow the alpha-numeric rules found under "Naming Conventions" below.
- Segmented the site sections according to:
  Root directory (the "images" folder usually goes at the top level within the root folder)
  Sub-directories (usually one for each area of the site, plus an images folder at the top level within the root directory)
  Images are restricted to one folder ("images") at the top level within the root directory (for global images) and then if a great number of images are going to be used only section-specifically, those are stored in local "images" folders

Naming Conventions

- Uses clients preferred naming method. If possible, uses longer descriptive names (like "content_design.htm" vs. "contdesi.htm").
- Uses alphanumeric characters (a-z, 0-9) and - (dash) or _ (underscore)
- Doesn't use spaces in file names.
- Avoids characters which require a shift key to create, or any punctuation other than a period.
- Uses only lower-case letters.
- Ends filenames in .htm (not .html).

Multimedia

- Any files taking longer than 10 seconds to download include a size warning (> 50kb on a 56kbps modem, > 200kb on fast connections). Also includes the running time of video clips or animations, and indicate any non-standard formats.
- Includes a short summary (and a still clip) of the linked object.
- If appropriate to the content, includes links to helper applications, like Adobe Acrobat Reader if the file is a .pdf.

Page Titles

- Follows title strategy ... Page Content Descriptor : Site Name, Site section (E.g.: Content Implementation Guidelines : CDG Solutions, Usability Process)
- Tries to use only two to six words, and makes their meaning clear when taken out of context.
- The first word's) are important information-carrying one's).
- Avoids making several page titles start with the same word.
- Checklist: Graphical User Interface

Test Type	Description	Purpose	Considerations	Variations
Transfer Functions	Navigate from each different window to all possible windows	Test interrelated processing between windows	- All Sequences? - Important Combinations? - Negative - No Transfers	Menu Bar-Mouse click
				RMB
				Toolbar
				Buttons - Push
				Buttons-Hot Key
				Buttons-Keyboard
				Menu Bar - Hot Keys
				Menu Bar - Keyboard
Data Conditions for Window Transfer Functions	Test transfers with general (record level) data conditions	Test data row retrieval and transfer functions using data	- Different for list windows vs. one record display windows	List window with no data
				List window one record in list (row)
				List window >1 row - last row
				List window >1 row - not first or last row
				One row display window
				Select inquiry entity in list window

				(not from list)
Verify Window Display Data	Verify inquiry data displays	Tests stored procedure/ GUI retrieval of data		Lists of Columns Single Row Display Drop Down List Box- Contents Drop Down List Box - Selection Retrieval Specific Data Retrieval Conditions- Max, Null, etc. Field Edit Formats
Field Level Data Entry	Test data entry for a single column	Test GUI field edits	(PBEdit040's within Data Windows)	Required Field - no data Maximum Data Length Valid Value Invalid Value Invalid data format
Row Data Maintenance	Test data row handling from GUI to database	Test stored procedure/GUI add/change/delete functions	Note: do an inquiry after update to verify database update	New Change to non-key field Change to key field (delete and add) Delete
Application Window Controls	Test Buttons, Scroll Bars and other windows types of controls	Test GUI processing	- Controls which do transfers are under transfer functions - Retrieve or OK which retrieves need to do inquiry to do data check of retrieval - Link, Unlink, Change, Delete need to do inquiry to check database updates - New test will be for data entry in field	Transfer Buttons OK, Miscellaneous NEW CLOSE/CANCEL RETRIEVE Database Updates LINK, UNLINK, CHANGE, DELETE Data Entry - NEW Radio Buttons Scroll Bars (Vertical/Horizontal)
Standard Window Controls/Functions				Window Control Menu Max, Min, Print Functions (Print, Printer Setup) Edit Functions (Cut, Copy, Paste) Window Functions (Previous Window, Close All, Open Window List, Tile, Layer, Cascade)
Application HELP				Micro help Balloon Notes Help- Index Help-Table of Contents Help-Jump Words Help-Text
Miscellaneous Application Specific				Job Status Online Report/s Informational Windows - Content Informational Windows - Button Fatal Application Errors

Section 1 - Windows Compliance Testing

1.1. Application

Start Application by Double Clicking on its ICON. The Loading message should show the application name, version number, and a bigger pictorial representation of the icon (a 'splash' screen).

No Login is necessary

The main window of the application should have the same caption as the caption of the icon in Program Manager.

Closing the application should result in an "Are you Sure" message box

Attempt to start application Twice
This should not be allowed - you should be returned to main Window

Try to start the application twice as it is loading.

On each window, if the application is busy, then the hour glass should be displayed. If there is no hour glass (e.g. alpha access enquiries) then some enquiry in progress message should be displayed.

All screens should have a Help button, F1 should work doing the same.

1.2. For Each Window in the Application

If Window has a Minimise Button, click it.

Window Should return to an icon on the bottom of the screen
This icon should correspond to the Original Icon under Program Manager.

Double Click the Icon to return the Window to its original size.

The window caption for every application should have the name of the application and the window name - especially the error messages. These should be checked for spelling, English and clarity , especially on the top of the screen. Check does the title of the window makes sense.

If the screen has an Control menu, then use all ungreyed options. (see below)

Check all text on window for Spelling/Tense and Grammar

Use TAB to move focus around the Window. Use SHIFT+TAB to move focus backwards.

Tab order should be left to right, and Up to Down within a group box on the screen. All controls should get focus - indicated by dotted box, or cursor. Tabbing to an entry field with text in it should highlight the entire text in the field.

The text in the Micro Help line should change - Check for spelling, clarity and non-updateable etc.

If a field is disabled (greyed) then it should not get focus. It should not be possible to select them with either the mouse or by using TAB. Try this for every greyed control.

Never updateable fields should be displayed with black text on a grey background with a black label.

All text should be left-justified, followed by a colon tight to it.

In a field that may or may not be updateable, the label text and contents changes from black to grey depending on the current status.

List boxes are always white background with black text whether they are disabled or not. All others are grey.

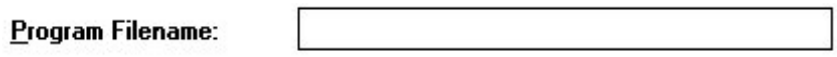
In general, do not use goto screens, use gosub, i.e. if a button causes another screen to be displayed, the screen should not hide the first screen, with the exception of tab in 2.0

When returning return to the first screen cleanly i.e. no other screens/applications should appear.

In general, double-clicking is not essential. In general, everything can be done using both the mouse and the keyboard.

All tab buttons should have a distinct letter.

1.3. Text Boxes



Move the Mouse Cursor over all Enterable Text Boxes. Cursor should change from arrow to Insert Bar.
If it doesn't then the text in the box should be grey or non-updateable. Refer to previous page.

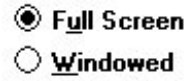
Enter text into Box

Try to overflow the text by typing to many characters - should be stopped Check the field width with capitals W.

Enter invalid characters - Letters in amount fields, try strange characters like + , - * etc. in All fields.

SHIFT and Arrow should Select Characters. Selection should also be possible with mouse. Double Click should select all text in box.

1.4. Option (Radio Buttons)



Left and Right arrows should move 'ON' Selection. So should Up and Down.. Select with mouse by clicking.

1.5. Check Boxes

Clicking with the mouse on the box, or on the text should SET/UNSET the box. SPACE should do the same.

1.6. Command Buttons

If Command Button leads to another Screen, and if the user can enter or change details on the other screen then
the Text on the button should be followed by three dots.

All Buttons except for OK and Cancel should have a letter Access to them. This is indicated by a letter underlined
in the button text. The button should be activated by pressing ALT+Letter. Make sure there is no duplication.

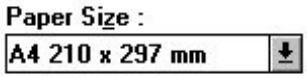
Click each button once with the mouse - This should activate
Tab to each button - Press SPACE - This should activate
Tab to each button - Press RETURN - This should activate
The above are **VERY IMPORTANT,** and should be done for **EVERY** command Button.

Tab to another type of control (not a command button). One button on the screen should be default (indicated by
a thick black border). Pressing Return in ANY no command button control should activate it.

If there is a Cancel Button on the screen , then pressing <Esc> should activate it.

If pressing the Command button results in uncorrectable data e.g. closing an action step, there should be a message
phrased positively with Yes/No answers where Yes results in the completion of the action.

1.7. Drop Down List Boxes



Pressing the Arrow should give list of options. This List may be scrollable. You should not be able to type text in the box.

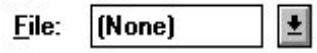
Pressing a letter should bring you to the first item in the list with that start with that letter. Pressing 'Ctrl - F4' should open/drop down the list box.

Spacing should be compatible with the existing windows spacing (word etc.). Items should be in alphabetical order with the exception of blank/none which is at the top or the bottom of the list box.

Drop down with the item selected should be display the list with the selected item on the top.

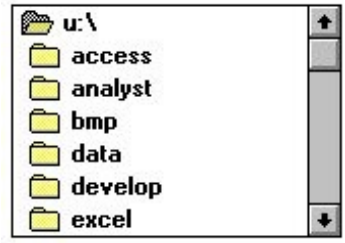
Make sure only one space appears, shouldn't have a blank line at the bottom.

1.8. Combo Boxes



Should allow text to be entered. Clicking Arrow should allow user to choose from list

1.9. List Boxes



Should allow a single selection to be chosen, by clicking with the mouse, or using the Up and Down Arrow keys.

Pressing a letter should take you to the first item in the list starting with that letter.

If there is a 'View' or 'Open' button beside the list box then double clicking on a line in the List Box, should act in the same way as selecting and item in the list box, then clicking the command button.

Force the scroll bar to appear, make sure all the data can be seen in the box.

GUI Testing Checklist (2)

Section 2 - Screen Validation Checklist

2.1. Aesthetic Conditions:

1. Is the general screen background the correct color?
2. Are the field prompts the correct color?
3. Are the field backgrounds the correct color?
4. In read-only mode, are the field prompts the correct color?
5. In read-only mode, are the field backgrounds the correct color?
6. Are all the screen prompts specified in the correct screen font?
7. Is the text in all fields specified in the correct screen font?
8. Are all the field prompts aligned perfectly on the screen?
9. Are all the field edit boxes aligned perfectly on the screen?
10. Are all group boxes aligned correctly on the screen?
11. Should the screen be resizable?
12. Should the screen be minim sable?
13. Are all the field prompts spelt correctly?
14. Are all character or alpha-numeric fields left justified? This is the default unless otherwise specified.
15. Are all numeric fields right justified? This is the default unless otherwise specified.
16. Is all the micro help text spelt correctly on this screen?

17. Is all the error message text spelt correctly on this screen?
18. Is all user input captured in UPPER case or lower case consistently?
19. Where the database requires a value (other than null) then this should be defaulted into fields. The user must either enter an alternative valid value or leave the default value intact.
20. Assure that all windows have a consistent look and feel.
21. Assure that all dialog boxes have a consistent look and feel.

2.2. Validation Conditions:

1. Does a failure of validation on every field cause a sensible user error message?
2. Is the user required to fix entries which have failed validation tests?
3. Have any fields got multiple validation rules and if so are all rules being applied?
4. If the user enters an invalid value and clicks on the OK button (i.e. does not TAB off the field) is the invalid entry identified and highlighted correctly with an error message.?
5. Is validation consistently applied at screen level unless specifically required at field level?
6. For all numeric fields check whether negative numbers can and should be able to be entered.
7. For all numeric fields check the minimum and maximum values and also some mid-range values allowable?
8. For all character/alphanumeric fields check the field to ensure that there is a character limit specified and that this limit is exactly correct for the specified database size?
9. Do all mandatory fields require user input?
10. If any of the database columns don't allow null values then the corresponding screen fields must be mandatory. (If any field which initially was mandatory has become optional then check whether null values are allowed in this field.)

2.3. Navigation Conditions:

1. Can the screen be accessed correctly from the menu?
2. Can the screen be accessed correctly from the toolbar?
3. Can the screen be accessed correctly by double clicking on a list control on the previous screen?
4. Can all screens accessible via buttons on this screen be accessed correctly?
5. Can all screens accessible by double clicking on a list control be accessed correctly?
6. Is the screen modal. i.e. Is the user prevented from accessing other functions when this screen is active and is this correct?
7. Can a number of instances of this screen be opened at the same time and is this correct?

2.4. Usability Conditions:

1. Are all the dropdowns on this screen sorted correctly? Alphabetic sorting is the default unless otherwise specified.
2. Is all date entry required in the correct format?
3. Have all pushbuttons on the screen been given appropriate Shortcut keys?
4. Do the Shortcut keys work correctly?
5. Have the menu options which apply to your screen got fast keys associated and should they have?
6. Does the Tab Order specified on the screen go in sequence from Top Left to bottom right? This is the default unless otherwise specified.
7. Are all read-only fields avoided in the TAB sequence?
8. Are all disabled fields avoided in the TAB sequence?
9. Can the cursor be placed in the micro help text box by clicking on the text box with the mouse?
10. Can the cursor be placed in read-only fields by clicking in the field with the mouse?
11. Is the cursor positioned in the first input field or control when the screen is opened?
12. Is there a default button specified on the screen?
13. Does the default button work correctly?
14. When an error message occurs does the focus return to the field in error when the user cancels it?

15. When the user Alt+Tab's to another application does this have any impact on the screen upon return to The application?
16. Do all the fields edit boxes indicate the number of characters they will hold by there length? e.g. a 30 character field should be a lot longer

2.5. Data Integrity Conditions:

1. Is the data saved when the window is closed by double clicking on the close box?
2. Check the maximum field lengths to ensure that there are no truncated characters?
3. Where the database requires a value (other than null) then this should be defaulted into fields. The user must either enter an alternative valid value or leave the default value intact.
4. Check maximum and minimum field values for numeric fields?
5. If numeric fields accept negative values can these be stored correctly on the database and does it make sense for the field to accept negative numbers?
6. If a set of radio buttons represent a fixed set of values such as A, B and C then what happens if a blank value is retrieved from the database? (In some situations rows can be created on the database by other functions which are not screen based and thus the required initial values can be incorrect.)
7. If a particular set of data is saved to the database check that each value gets saved fully to the database. i.e. Beware of truncation (of strings) and rounding of numeric values.

2.6. Modes (Editable Read-only) Conditions:

1. Are the screen and field colors adjusted correctly for read-only mode?
2. Should a read-only mode be provided for this screen?
3. Are all fields and controls disabled in read-only mode?
4. Can the screen be accessed from the previous screen/menu/toolbar in read-only mode?
5. Can all screens available from this screen be accessed in read-only mode?
6. Check that no validation is performed in read-only mode.

General Conditions:

1. Assure the existence of the "Help" menu.
2. Assure that the proper commands and options are in each menu.
3. Assure that all buttons on all tool bars have a corresponding key commands.
4. Assure that each menu command has an alternative (hot-key) key sequence which will invoke it where appropriate.
5. In drop down list boxes, ensure that the names are not abbreviations / cut short
6. In drop down list boxes, assure that the list and each entry in the list can be accessed via appropriate key / hot key combinations.
7. Ensure that duplicate hot keys do not exist on each screen
8. Ensure the proper usage of the escape key (which is to undo any changes that have been made) and generates a caution message "Changes will be lost - Continue yes/no"
9. Assure that the cancel button functions the same as the escape key.
10. Assure that the Cancel button operates as a Close button when changes have be made that cannot be undone.
11. Assure that only command buttons which are used by a particular window, or in a particular dialog box, are present. - i.e. make sure they don't work on the screen behind the current screen.
12. When a command button is used sometimes and not at other times, assure that it is grayed out when it should not be used.
13. Assure that OK and Cancel buttons are grouped separately from other command buttons.
14. Assure that command button names are not abbreviations.
15. Assure that all field labels/names are not technical labels, but rather are names meaningful to system users.
16. Assure that command buttons are all of similar size and shape, and same font & font size.
17. Assure that each command button can be accessed via a hot key combination.
18. Assure that command buttons in the same window/dialog box do not have duplicate hot keys.

19. Assure that each window/dialog box has a clearly marked default value (command button, or other object) which is invoked when the Enter key is pressed - and NOT the Cancel or Close button
20. Assure that focus is set to an object/button which makes sense according to the function of the window/dialog box.
21. Assure that all option buttons (and radio buttons) names are not abbreviations.
22. Assure that option button names are not technical labels, but rather are names meaningful to system users.
23. If hot keys are used to access option buttons, assure that duplicate hot keys do not exist in the same window/dialog box.
24. Assure that option box names are not abbreviations.
25. Assure that option boxes, option buttons, and command buttons are logically grouped together in clearly demarcated areas "Group Box"
26. Assure that the Tab key sequence which traverses the screens does so in a logical way.
27. Assure consistency of mouse actions across windows.
28. Assure that the color red is not used to highlight active objects (many individuals are red-green color blind).
29. Assure that the user will have control of the desktop with respect to general color and highlighting (the application should not dictate the desktop background characteristics).
30. Assure that the screen/window does not have a cluttered appearance
31. Ctrl + F6 opens next tab within tabbed window
32. Shift + Ctrl + F6 opens previous tab within tabbed window
33. Tabbing will open next tab within tabbed window if on last field of current tab
34. Tabbing will go onto the 'Continue' button if on last field of last tab within tabbed window
35. Tabbing will go onto the next editable field in the window
36. Banner style & size & display exact same as existing windows
37. If 8 or less options in a list box, display all options on open of list box - should be no need to scroll
38. Errors on continue will cause user to be returned to the tab and the focus should be on the field causing the error. (i.e the tab is opened, highlighting the field with the error on it)
39. Pressing continue while on the first tab of a tabbed window (assuming all fields filled correctly) will not open all the tabs.
40. On open of tab focus will be on first editable field
41. All fonts to be the same
42. Alt+F4 will close the tabbed window and return you to main screen or previous screen (as appropriate), generating "changes will be lost" message if necessary.
43. Micro help text for every enabled field & button
44. Ensure all fields are disabled in read-only mode
45. Progress messages on load of tabbed screens
46. Return operates continue
47. If retrieve on load of tabbed window fails window should not open

Specific Field Tests

2.8.1. Date Field Checks

- Assure that leap years are validated correctly & do not cause errors/miscalculations
- Assure that month code 00 and 13 are validated correctly & do not cause errors/miscalculations
- Assure that 00 and 13 are reported as errors
- Assure that day values 00 and 32 are validated correctly & do not cause errors/miscalculations
- Assure that Feb. 28, 29, 30 are validated correctly & do not cause errors/ miscalculations
- Assure that Feb. 30 is reported as an error
- Assure that century change is validated correctly & does not cause errors/ miscalculations

- Assure that out of cycle dates are validated correctly & do not cause errors/miscalculations

2.8.2. Numeric Fields

- Assure that lowest and highest values are handled correctly
- Assure that invalid values are logged and reported
- Assure that valid values are handles by the correct procedure
- Assure that numeric fields with a blank in position 1 are processed or reported as an error
- Assure that fields with a blank in the last position are processed or reported as an error an error
- Assure that both + and - values are correctly processed
- Assure that division by zero does not occur
- Include value zero in all calculations
- Include at least one in-range value
- Include maximum and minimum range values
- Include out of range values above the maximum and below the minimum
- Assure that upper and lower values in ranges are handled correctly

2.8.3. Alpha Field Checks

- Use blank and non-blank data
- Include lowest and highest values
- Include invalid characters & symbols
- Include valid characters
- Include data items with first position blank
- Include data items with last position blank
- **Section 3 - Validation Testing - Standard Actions**
- *3.1. Examples of Standard Actions - Substitute your specific commands*
- Add
  View
  Change
  Delete
  Continue - (i.e. continue saving changes or additions)
- Add
  View
  Change
  Delete
  Cancel - (i.e. abandon changes or additions)
- Fill each field - Valid data
  Fill each field - Invalid data
- Different Check Box / Radio Box combinations
- Scroll Lists / Drop Down List Boxes
- Help
- Fill Lists and Scroll
- Tab
- Tab Sequence
- Shift Tab
- **3.2. Shortcut keys / Hot Keys**
- *Note: The following keys are used in some windows applications, and are included as a guide.*

- **3.3. Control Shortcut Keys**

- * These shortcuts are suggested for text formatting applications, in the context for which they make sense. Applications may use other modifiers for these operations.

Key	No Modifier	Shift	CTRL	ALT
F1	**Help**	**Enter Help Mode**	**n\a**	**n\a**
F2	**n\a**	**n\a**	**n\a**	**n\a**
F3	**n\a**	**n\a**	**n\a**	**n\a**
F4	**n\a**	**n\a**	**Close Document / Child window.**	**Close Application.**
F5	**n\a**	**n\a**	**n\a**	**n\a**
F6	**n\a**	**n\a**	**n\a**	**n\a**
F7	**n\a**	**n\a**	**n\a**	**n\a**
F8	**Toggle extend mode, if supported.**	**Toggle Add mode, if supported.**	**n\a**	**n\a**
F9	**n\a**	**n\a**	**n\a**	**n\a**
F10	**Toggle menu bar activation.**	**n\a**	**n\a**	**n\a**
F11, F12	**n\a**	**n\a**	**n\a**	**n\a**
Tab	**Move to next active/editable field.**	**Move to previous active/editable field.**	**Move to next open Document or Child window. (Adding SHIFT reverses the order of movement).**	**Switch to previously used application. (Holding down the ALT key displays all open applications).**
Alt	**Puts focus on first menu command (e.g. 'File').**	**n\a**	**n\a**	**n\a**

- **3.3. Control Shortcut Keys**

- * These shortcuts are suggested for text formatting applications, in the context for which they make sense. Applications may use other modifiers for these operations.

Key	Function
CTRL + Z	Undo
CTRL + X	Cut
CTRL + C	Copy
CTRL + V	Paste
CTRL + N	New
CTRL + O	Open
CTRL + P	Print

CTRL + S	Save
CTRL + B	Bold*
CTRL + I	Italic*
CTRL + U	Underline*

- * These shortcuts are suggested for text formatting applications, in the context for which they make sense. Applications may use other modifiers for these operations.

Checklist: Numeric Entry

The following edits, questions, and checks should be considered for all numeric fields.

Edit / Question	**Example**
Maximum Value & Minimum Value	• Edit Picture (z, 9, #, etc.) • Field Width • Boundaries (Upper Limit, Lower Limit) • Positive and Negative Numbers • Precision (Whole Numbers and Decimal Places) • Signed or Unsigned
Delta - Smallest increment used by system	• Whole Numbers • Fractions • Decimal Value
Other Tests	• Overflow • Underflow • Rounding • Floating Point Errors
Formats	• Currency (Symbol, Separators, Commas & Period) • Input • Storage • Output • Display • Print • Integer (16, 32, 64) • Floating Point • Binary • Packed • Hex, Octal, Scientific Notation • Placement of Negative Indicator • -, CB, ( ) Leading or Trailing • Word Boundaries
Attributes	• Position (Display or Print) • Color (Red for Negative)

	• Intensity • Blinking • Font Size • Italics
Zero	• Leading 0123 • Trailing 123.0 • Absent 123.
Spaces Before or After Entry	• Permitted? • Self Correcting?
Alternative Formats	• Display values in thousands or millions • Display as words "One Thousand" • Roman numerals
Error Message	• When displayed • Where displayed • Should they be acknowledged? • Automatic Recovery?
Initialization	• Starting Value • Null Value • Reset Value
Reasonableness Checks	.
Entry Format	• Character • Numeric
Match other formats	• Usage in a calculator • Appended to another field
Display Issues	• Blank on either side of field to prevent touching another field • 123 123 vs. 123123 • Display with leading zero
Will conversion take place?	.
Source of Value	• Will it change? • Multiple? • Alternative Source
Can value be computed?	.
Balancing instructions	Audit Issues
Encrypted storage	.
Is a check digit required?	Check digit computation
Validation	• Table • Computation

	• Other report
Does field have other use?	• SSN, SIN, Employee ID • Salary, Speed • Date • Lookup Key
Naming Conventions	.
Compiler Requirements	.

Note the edits that are performed by the programming language, tests that should be handled during unit testing, and checks that should be done via integration or system testing.

Other issues:

1. Will boundaries and limits change over time?
2. Are they influenced by something else?
3. Will field accept operators? +, -, /, *, !, **, ^, %
4. Will the value change format?
   64 bit to 16 bit
   Character to numeric
   Display to packed
   Display to scientific notation
   Display to words
5. Will value move across platforms?
6. Why is the field being treated as numeric?
7. Will voice recognition be necessary?

Checklist: Additional Testing Concerns

- * Memory: availability, high, low, virtual, swapping, page size
- * Resource competition on the system
- * Processing: batch, on-line, multiple input sources
- * Conflicts: anti-viral software, automated test tools, TSR's, security software, IRQs
- * Backup: disaster recovery, backups, rerun capability
- * Connectivity: band width, interoperability, modem speed, ISDN lines, distributed applications
- * Security: passwords, fire walls
- * CD-Rom access and speed
- * File conversions
- * Design of client server architecture
- * Version Controls
- * Display issues: graphics, monitor size, graphic cards
- * Printer: type, speed, color, resolution, paper weight, paper size, envelopes, multiple printers
- * Platform: mainframe, minicomputer, microcomputer, number of platforms
- * Multiple operating systems
- * Transactions: size, quantity, rate
- * Error processing: message source, location, timing, acknowledgements

Checklist: Date Entry

The following edits, questions, and checks should be considered for all date fields. Be aware that many programming languages combine date and time into one data type.

Edit / Question	Example
Required entry	.

Century display	1850, 1999, 2001
Implied century	Display last two digits of year (96,02). All dates are assumed to be between 1950 and 2049.
Date display format	mm-dd-yy (12/01/96) mm-dd-ccyy (12/01/1996) dd-mm-yy (01/12/96) dd-mm-ccyy (01/12/1996) dd-mmm-yy (01-Jan-96) dd-mmm-ccyy (01-Jan-1996) dd-mm (day and month only) Complete date (December 1, 1996) Date, abbreviated month (Dec 1, 1996) Day included in date (Monday, November 7, 1996) yymmdd (960105) ccyymmdd (20011231) No year, text month and day (May 30th) Financial calculator (12.0196) System format (provided through system)
Date separator	Slash (/), dash (-), Period (.), space Enter separator fields Move over separators Automatic skip over separators
Leading zeros in day field	01, 02, 09 (12/05/96 vs 12/5/96)
Leading zeros in month field	01, 02, 09 (05/17/97 vs 5/17/97)
Abbreviate month name	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
Can day numbers exceed actual?	May 32, Feb 30 (Accounting systems may use for adjusting transactions.)
Embedded spaces	No leading spaces No trailing spaces One space permitted after month and comma
Font attributes	Color Italics Bold Size Blink Intensity
Leap year computations	Any year number evenly divisible by 4, but not by 100, unless it is also divisible by 400. 1996 leap year 2000 leap year 2004 leap year 2100 not a leap year
Relational edits	Compare hire date to birth date.
Use financial calendars	30/360 30 days per month, 360 days per year Actual/360 actual days per month, 360 days per year
Entry mechanism	Display calendar (+/-) to change day number Function key / PF key
Default date	System date Last date entered Other date (order date, closing date, etc.)
Latest / earliest permissible date	Actual date Computed date
Authorization / Security required	Add Modify Delete View
Formats	Entry Storage (date, or relative day number) Print Display
Null date	00/00/00 zeros bb/bb/bb spaces
Is the programming responsible for managing	.

dates?	
Are autofill features utilized?	.
Will the date field be used again elsewhere?	.
Is this a standard date entry routine that is already tested?	.
Are there other mechanisms to date stamp fields or records?	.
Is the position of the date important?	On screen In a report
Are other events triggered by this date?	.
Permissible dates	Holiday local, regional, national, international Weekend Specific day(s) of week
Is the Julian date required?	.
Sorting requirements	Normal Relative day number Unusual 9's compliment, yymmdd, ccyymmdd
Time zone issues	.
Is system voice enabled for date entry?	.
Is the date encrypted?	Encryption technique
Testing	Must entry dates correspond to dates in the test bed?
Risk factors	What is the risk inherent in not entering the date correctly.
Edit date	On entry When screen is complete When record is complete After other event
Are incomplete dates permissible?	12/00/1996 12/??/1996 12/01/???? 12/01/??
Font	Acceptable fonts Largest font size that will display properly Default font
Correction	Can erroneous dates be automatically corrected?
Error messages	Content Placement When displayed Can processing continue with bad date?

Note the edits that are performed by the programming language, tests that should be handled during unit testing, and checks that should be done via integration or system testing.

Other issues:

1. Can invalid dates be passed to this routine? Should they be accepted?
2. Is there a standard date entry routine in the library?
3. Can new date formats be easily added and edited?
4. What is the source of the date: input documents, calendar on the wall, or field on another document?
5. Are there other mechanisms to change dates outside of this program?
6. Is this a date and time field?
7. **Checklist: Developing Windows Application**
8. **Modal Windows** - Often times modal windows which must be acted upon end up hidden behind standard windows. This gives the user the impression that the system has locked up. **Special Characters** - Special characters may not be used on some windows entry screens, there also may be some conflicts with converting data or using data from other systems. **Printer Configuration** - Although Windows is designed to handle the printer setup for most applications, there are formatting differences between printers and printer types. LaserJet printers do not behave the same as inkjets, nor do 300, 600, or 1200 DPI laser printers behave the same across platforms. **Date Formats** - The varying date formats sometimes cause troubles when they are being displayed in windows entry screens. This situation could occur when programs are designed to handle a YY/MM/DD format and the date format being used is YYYY/MMM/DD. **Screen Savers** - Some screen savers such as After Dark are memory or resource 'hogs' and have been known to

cause troubles when running other applications. **Speed Keys** - Verify that there are no conflicting speed keys on the various screens. This is especially important on screens where the buttons change.

9. **Virus Protection Software** - Some virus protection software can be configured too strictly. This may cause applications to run slowly or incorrectly. **Disk Compression Tools** - Some disk compression software may cause our applications to run slowly or incorrectly. **Multiple Open Windows** - How does the system handle having multiple open windows, are there any resource errors. **Test Multiple Environments** - Programs need to be tested under multiple configurations. The configurations seem to cause various results. **Test Multiple Operating Systems** - Programs running under Win 95, Win NT, and Windows 3.11 do not behave the same in all environments. **Corrupted DLL's** - Corrupted DLL's will sometime cause applications not to execute or more damaging to run sporadically. **Incorrect DLL Versions** - Corrupted DLL's will sometime cause our applications not to execute or more damaging to run sporadically. **Missing DLL's** - Missing DLL's will usually cause our applications not to execute. **Standard Program Look & Feel** - The basic windows look & feel should be consistent across all windows and the entire application. Windows buttons, windows and controls should follow the same standards for sizes.**Tab Order** - When pressing the TAB key to change focus from object to object the procession should be logical. **Completion of Edits** - The program should force the completion of edits for any screen before users have a change to exit program. **Saving Screen Sizes** - Does the user have an opportunity to save the current screen sizes and position? **Operational Speed** - Make sure that the system operates at a functional speed, databases, retrieval, and external references. **Testing Under Loaded Environments** - Testing system functions when running various software programs "RESOURCE HOGS" (MS Word, MS Excel, WP, etc.). **Resource Monitors** - Resource monitors help track Windows resources which when expended will cause GPF's. **Video Settings** - Programmers tend to program at a 800 x 600 or higher resolution, when you run these programs at a default 640 x 480 it tends to overfill the screen. Make sure the application is designed for the resolution used by customers. **Clicking on Objects Multiple Times** - Will you get multiple instances of the same object or window with multiple clicks? **Saving Column Orders** - Can the user save the orders of columns of the display windows? **Displaying Messages saying that the system is processing** - When doing system processing do we display some information stating what the system is doing? **Clicking on Other Objects While the System is Processing** - Is processing interrupted? Do unexpected events occur after processing finishes? **Large Fonts / Small Fonts** - When switching between windows font sizes mixed results occur when designing in one mode and executing in another. **Maximizing / Minimizing all windows** - Do the actual screen elements resize? Do we use all of the available screen space when the screen is maximized. **Setup Program** - Does your setup program function correctly across multiple OS's. Does the program prompt the user before overwriting existing files. **Consistency in Operation** - Consistent behavior of the program in all screens and the overall application. **Multiple Copies of the same Window** - Can the program handle multiple copies of the same window? Can all of these windows be edited concurrently? **Confirmation of Deletes** - All deletes should require confirmations of the process before execution. **Selecting alternative language options** - Will your program handle the use of other languages (FRENCH, SPANISH, ITALIAN, etc.)

How to be a Software Tester

What makes a good software tester?

1. Know Programming. Might as well start out with the most controversial one. There's a popular myth that testing can be staffed with people who have little or no programming knowledge. It doesn't work, even though it is an unfortunately common approach. There are two main reasons why it doesn't work.

(1) They're testing software. Without knowing programming, they can't have any real insights into the kinds of bugs that come into software and the likeliest place to find them. There's never enough time to test "completely", so all software testing is a compromise between available resources and thoroughness. The tester must optimize scarce resources and that means focusing on where the bugs are likely to be. If you don't know programming, you're unlikely to have useful intuition about where to look.
(2) All but the simplest (and therefore, ineffectual) testing methods are tool- and technology-intensive. The tools, both as testing products and as mental disciplines, all presume programming knowledge. Without programmer training, most test techniques (and the tools based on those techniques) are unavailable. The tester who doesn't know programming will always be restricted to the use of ad-hoc techniques and the most simplistic tools.

Taking entry-level programmers and putting them into a test organization is not a good idea because:

(1) Loser Image.
Few universities offer undergraduate training in testing beyond "Be sure to test thoroughly." Entry-level people expect to get a job as a programmer and if they're offered a job in a test group, they'll often look upon it as a failure on their part: they believe that they didn't have what it takes to be a programmer in that organization. This unfortunate perception exists even in organizations that values testers highly.

(2) Credibility With Programmers.
Independent testers often have to deal with programmers far more senior than themselves. Unless they've

been through a coop program as an undergraduate, all their programming experience is with academic toys: the novice often has no real idea of what programming in a professional, cooperative, programming environment is all about. As such, they have no credibility with their programming counterpart who can sluff off their concerns with "Look, kid. You just don't understand how programming is done here, or anywhere else, for that matter." It is setting up the novice tester for failure.
(3) Just Plain Know-How.
The programmer's right. The kid doesn't know how programming is really done. If the novice is a "real" programmer (as contrasted to a "mere tester") then the senior programmer will often take the time to mentor the junior and set her straight: but for a non-productive "leech" from the test group? Never! It's easiest for the novice tester to learn all that nitty-gritty stuff (such as doing a build, configuration control, procedures, process, etc.) while working as a programmer than to have to learn it, without actually doing it, as an entry-level tester.

2. Know the Application.
That's the other side of the knowledge coin. The ideal tester has deep insights into how the users will exploit the program's features and the kinds of cockpit errors that users are likely to make. In some cases, it is virtually impossible, or at least impractical, for a tester to know both the application and programming. For example, to test an income tax package properly, you must know tax laws and accounting practices. Testing a blood analyzer requires knowledge of blood chemistry; testing an aircraft's flight control system requires control theory and systems engineering, and being a pilot doesn't hurt; testing a geological application demands geology. If the application has a depth of knowledge in it, then it is easier to train the application specialist into programming than to train the programmer into the application. Here again, paralleling the programmer's qualification, I'd like to see a university degree in the relevant discipline followed by a few years of working practice before coming into the test group.

3. Intelligence.
Back in the 60's, there were many studies done to try to predict the ideal qualities for programmers. There was a shortage and we were dipping into other fields for trainees. The most infamous of these was IBM's programmers' Aptitude Test (PAT). Strangely enough, despite the fact the IBM later repudiated this test, it continues to be (ab)used as a benchmark for predicting programmer aptitude. What IBM learned with follow-on research is that the single most important quality for programmers is raw intelligence-good programmers are really smart people-and so are good testers.

What makes a good software tester?

4. Hyper-Sensitivity to Little Things.
Good testers notice little things that others (including programmers) miss or ignore. Testers see symptoms, not bugs. We know that a given bug can have many different symptoms, ranging from innocuous to catastrophic. We know that the symptoms of a bug are arbitrarily related in severity to the cause. Consequently, there is no such thing as a minor symptom-because a symptom isn't a bug. It is only after the symptom is fully explained (i.e., fully debugged) that you have the right to say if the bug that caused that symptom is minor or major. Therefore, anything at all out of the ordinary is worth pursuing. The screen flickered this time, but not last time-a bug. The keyboard is a little sticky-another bug. The account balance is off by 0.01 cents-great bug. Good testers notice such little things and use them as an entree to finding a closely-related set of inputs that will cause a catastrophic failure and therefore get the programmers' attention. Luckily, this attribute can be learned through training.

5. Tolerance for Chaos.
People react to chaos and uncertainty in different ways. Some cave in and give up while others try to create order out of chaos. If the tester waits for all issues to be fully resolved before starting test design or testing, she won't get started until after the software has been shipped. Testers have to be flexible and be able to drop things when blocked and move on to another thing that's not blocked. Testers always have many (unfinished) irons in the fire. In this respect, good testers differ from programmers. A compulsive need to achieve closure is not a bad attribute in a programmer-certainly serves them well in debugging-in testing, it means nothing gets finished. The testers' world is inherently more chaotic than the programmers'.

A good indicator of the kind of skill I'm looking for here is the ability to do crossword puzzles in ink. This skill, research has shown, also correlates well with programmer and tester aptitude. This skill is very similar to the kind of unresolved chaos with which the tester must daily deal. Here's the theory behind the notion. If you do a crossword puzzle in ink, you can't put down a word, or even part of a word, until you have confirmed it by a compatible cross-word. So you keep a dozen tentative entries unmarked and when by some process or another, you realize that there is a compatible cross-word, you enter them both. You keep score by how many corrections you have to make-not by merely finishing the puzzle, because that's a given. I've done many informal polls of this aptitude at my seminars and found a much higher percentage of crossword-puzzles-in-ink aficionados than you'd get in a normal population.

6. People Skills.
Here's another area in which testers and programmers can differ. You can be an effective programmer even if you are hostile and anti-social; that won't work for a tester. Testers can take a lot of abuse from outraged programmers. A sense of humor and a thick skin will help the tester survive. Testers may have to be diplomatic when confronting a senior programmer with a fundamental goof. Diplomacy, tact, a ready smile-all work to the independent tester's advantage. This may explain one of the (good) reasons that there are so many women in

testing. Women are generally acknowledged to have more highly developed people skills than comparable men-whether it is something innate on the X chromosome as some people contend or whether it is that without superior people skills women are unlikely to make it through engineering school and into an engineering career, I don't know and won't attempt to say. But the fact is there and those sharply-honed people skills are important

7. Tenacity.
An ability to reach compromises and consensus can be at the expense of tenacity. That's the other side of the people skills. Being socially smart and diplomatic doesn't mean being indecisive or a limp rag that anyone can walk all over. The best testers are both-socially adept and tenacious where it matters. The best testers are so skillful at it that the programmer never realizes that they've been had. Tenacious-my picture is that of an angry pitbull fastened on a burglar's rear-end. Good testers don You can't intimidate them-even by pulling rank. They'll need high-level backing, of course, if they're to get you the quality your product and market demands.

8. Organized.
I can't imagine a scatter-brained tester. There's just too much to keep track of to trust to memory. Good testers use files, data bases, and all the other accouterments of an organized mind. They make up checklists to keep themselves on track. They recognize that they too can make mistakes, so they double-check their findings. They have the facts and figures to support their position. When they claim that there's a bug-believe it, because if the developers don't, the tester will flood them with well-organized, overwhelming, evidence.

A consequence of a well-organized mind is a facility for good written and oral communications. As a writer and editor, I've learned that the inability to express oneself clearly in writing is often symptomatic of a disorganized mind. I don't mean that we expect everyone to write deathless prose like a Hemingway or Melville. Good technical writing is well-organized, clear, and straightforward: and it doesn't depend on a 500,000 word vocabulary. True, there are some unfortunate individuals who express themselves superbly in writing but fall apart in an oral presentation- but they are typically a pathological exception. Usually, a well-organized mind results in clear (even if not inspired) writing and clear writing can usually be transformed through training into good oral presentation skills.

9. Skeptical.
That doesn't mean hostile, though. I mean skepticism in the sense that nothing is taken for granted and that all is fit to be questioned. Only tangible evidence in documents, specifications, code, and test results matter. While they may patiently listen to the reassuring, comfortable words from the programmers ("Trust me. I know where the bugs are.")-and do it with a smile-they ignore all such in-substantive assurances.

10. Self-Sufficient and Tough.
If they need love, they don't expect to get it on the job. They can't be looking for the interaction between them and programmers as a source of ego-gratification and/or nurturing. Their ego is gratified by finding bugs, with few misgivings about the pain (in the programmers) that such finding might engender. In this respect, they must practice very tough love.

11. Cunning.
Or as Gruenberger put it, "low cunning." "Street wise" is another good descriptor, as are insidious, devious, diabolical, fiendish, contriving, treacherous, wily, canny, and underhanded. Systematic test techniques such as syntax testing and automatic test generators have reduced the need for such cunning, but the need is still with us and undoubtedly always will be because it will never be possible to systematize all aspects of testing. There will always be room for that offbeat kind of thinking that will lead to a test case that exposes a really bad bug. But this can be taken to extremes and is certainly not a substitute for the use of systematic test techniques. The cunning comes into play after all the automatically generated "sadistic" tests have been executed.

12. Technology Hungry.
They hate dull, repetitive, work-they'll do it for a while if they have to, but not for long. The silliest thing for a human to do, in their mind, is to pound on a keyboard when they're surrounded by computers. They have a clear notion of how error-prone manual testing is, and in order to improve the quality of their own work, they'll f ind ways to eliminate all such error-prone procedures. I've seen excellent testers re-invent the capture/playback tool many times. I've seen dozens of home-brew test data generators. I've seen excellent test design automation done with nothing more than a word processor, or earlier, with a copy machine and lots of bottles of white-out. I've yet to meet a tester who wasn't hungry for applicable technology. When asked why didn't they automate such and such-the answer was never "I like to do it by hand." It was always one of the following: (1) "I didn't know that it could be automated", (2) "I didn't know that such tools existed", or worst of all, (3) "Management wouldn't give me the time to learn how to use the tool."

13. Honest.
Testers are fundamentally honest and incorruptible. They'll compromise if they have to, but they'll righteously agonize over it. This fundamental honesty extends to a brutally realistic understanding of their own limitations as a human being. They accept the idea that they are no better and no worse, and therefore no less error-prone than their programming counterparts. So they apply the same kind of self-assessment procedures that good programmers will. They'll do test inspections just like programmers do code inspections. The greatest possible crime in a tester's eye is to fake test results.

Personal Requirements For Software Quality Assurance Engineers

Challenges
Rapidly changing requirements
Foresee defects that are likely to happen in production
Monitor and Improve the software development processes
Ensure that standards and procedures are being followed
Customer Satisfaction and confidence
Compete the Market

Identifying Software Quality Assurance Personnel Needs:
Requirement Specification
Functional Specification
Technical Specification
Standards document and user manuals – If applicable (e.g. Coding standards document)
Test Environment Setup

Professional Characteristics of a good SQA Engineer
Understanding of business approach and goals of the organization
Understanding of entire software development process
Strong desire for quality
Establish and enforce SQA methodologies, processes and Testing Strategies
Judgment skills to assess high-risk areas of application
Communication with Analysis and Development team
Report defects with full evidence
Take preventive actions
Take actions for Continuous improvement
Reports to higher management
Say No when Quality is insufficient
Work Management
Meet deadlines

Personal Characteristics of a good SQA Engineer
Open Minded
Observant
Perceptive
Tenacious
Decisive
Diplomatic
Keen for further training/trends in QA