# Testing Web Applications

*Slide Set to accompany*

*Software Engineering: A Practitioner's Approach, 7/e*

**by Roger S. Pressman**

## Software Testing Principles

- Consider the following important testing principles:

1. Testing is the process of executing a program with the specific intent of finding errors.
2. A good test is one that has a high probability of detecting an as-yet undiscovered error.
3. A successful test case is one that detects an as-yet undiscovered error.

**[G. Myers, 1979]**

## WebApp Testing

- A collection of (related) activities with a single goal, namely:

  Uncover errors in *WebApp content, function, usability, navigability, performance, capacity, and security.*

- A testing *strategy* that encompasses both *reviews & executable testing* is applied.
- WebApp testing begins by focusing on user-visible aspects of the WebApp, and proceeds to test that exercise technology and infrastructure.

## WebApp Testing

- Consider the following WebApp testing steps:
  1. Content testing.
  2. Interface testing.
  3. Navigation testing.
  4. Component testing.
  5. Configuration testing.
  6. Performance testing.
  7. Security testing.

**Note:**

In every instance, a suite of test cases is developed for every testing step, and an archive of test results is maintained for future use.

# Testing Quality Dimensions-I

- *Content* is evaluated at both a syntactic and semantic level.
  - *syntactic level*—spelling, punctuation and grammar are assessed for text-based documents.
  - *semantic level*—correctness (of information presented), consistency (across the entire content object and related objects) and lack of ambiguity are all assessed.
- *Function* is tested for correctness, instability, and general conformance to appropriate implementation standards (e.g.,Java or XML language standards).
- *Structure* is assessed to ensure that it
  - properly delivers WebApp content and function
  - is extensible
  - can be supported as new content or functionality is added.

## Testing Quality Dimensions-II

- *Usability* is tested to ensure that each category of user
  - is supported by the interface
  - can learn and apply all required navigation syntax and semantics
- *Navigability* is tested to ensure that
  - all navigation syntax and semantics are exercised to uncover any navigation errors (e.g., dead links, improper links, erroneous links).
- *Performance* is tested under a variety of operating conditions, configurations, and loading to ensure that
  - the system is responsive to user interaction
  - the system handles extreme loading without unacceptable operational degradation

# Testing Quality Dimensions-III

- *Compatibility* is tested by executing the WebApp in a variety of different host configurations on both the client and server sides.
  - The intent is to find errors that are specific to a unique host configuration.
- *Interoperability* is tested to ensure that the WebApp properly interfaces with other applications and/or databases.
- *Security* is tested by assessing potential vulnerabilities and attempting to exploit each.
  - Any successful penetration attempt is deemed a security failure.

## Errors in a WebApp

- Because many types of WebApp tests uncover problems that are first evidenced on the client side, you often see a symptom of the error, not the error itself.
- Because a WebApp is implemented in a number of different configurations and within different environments, it may be difficult or impossible to reproduce an error outside the environment in which the error was originally encountered.
- Although some errors are the result of incorrect design or improper HTML (or other programming language) coding, many errors can be traced to the WebApp configuration.
- Because WebApps reside within a client/server architecture, errors can be difficult to trace across three architectural layers: the client, the server, or the network itself.
- Some errors are due to the *static operating environment* (i.e., the specific configuration in which testing is conducted), while others are attributable to the dynamic operating environment (i.e., instantaneous resource loading or time-related errors).

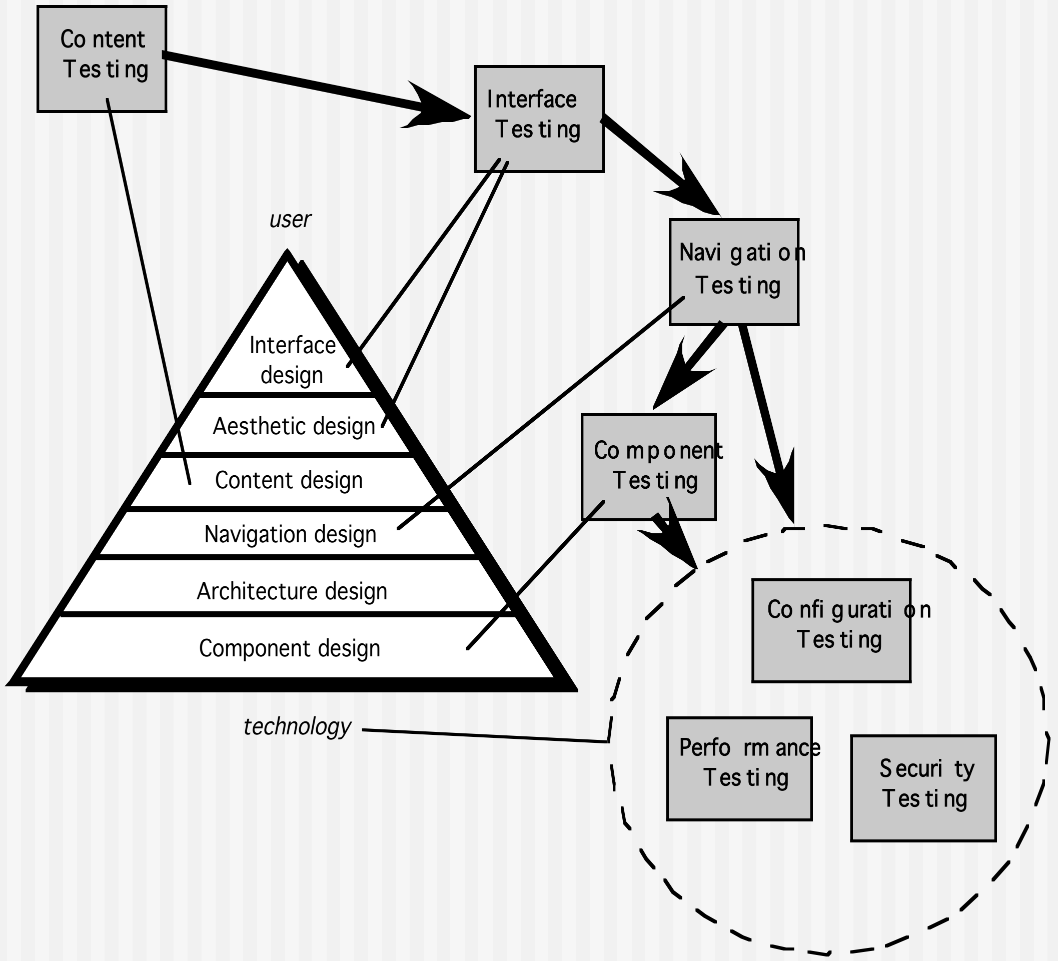
## WebApp Testing Strategy-I

- The content model for the WebApp is *reviewed* to uncover errors.
- The interface model is *reviewed* to ensure that all use-cases can be accommodated.
- The design model for the WebApp is *reviewed* to uncover navigation errors.
- The user interface is *tested* to uncover errors in presentation and/or navigation mechanics.
- Selected functional components are *unit tested.*

# WebApp Testing Strategy-II

- Navigation throughout the architecture *is tested*.
- The WebApp is implemented in a variety of different environmental configurations and is *tested for compatibility* with each configuration.
- *Security tests* are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment.
- *Performance tests* are conducted.
- The *WebApp is tested by a controlled and monitored population of end-users*
  - the results of their interaction with the system are evaluated for content and navigation errors, usability concerns, compatibility concerns, and WebApp reliability and performance.

# The Testing Process



## Content Testing

- Content testing has three important objectives:
  - to uncover *syntactic errors* (e.g., typos, grammar mistakes) in text-based documents, graphical representations, and other media
  - to uncover *semantic errors* (i.e., errors in the accuracy or completeness of information) in any content object presented as navigation occurs, and
  - to find *errors in the organization or structure of content* that is presented to the end-user.

## Assessing Content *Semantics*

- Is the information factually accurate?
- Is the information concise and to the point?
- Is the layout of the content object easy for the user to understand?
- Can information embedded within a content object be found easily?
- Have proper references been provided for all information derived from other sources?
- Is the information presented consistent internally and consistent with information presented in other content objects?
- Is the content offensive, misleading, or does it open the door to litigation?
- Does the content infringe on existing copyrights or trademarks?
- Does the content contain internal links that supplement existing content? Are the links correct?
- Does the aesthetic style of the content conflict with the aesthetic style of the interface?

## Assessing Content *Semantic*

- Failure to uncover semantic errors will shake the user's faith in the WebApp, and can lead to failure of the Web-based application.
- During content testing, the structure and organisation of the content architecture is tested to ensure that required content is presented to the end user in the proper order and with the correct relationships.

# Database Testing

- Modern WebApps do much more than present *static* content objects.
- In many application domains, WebApps interface with sophisticated *database management systems, (DBMS),* and build *dynamic* content objects that are created *in real time* using the data acquired from a database.
- Consider the following, typical, steps:
  1. A large database is queried.
  2. Relevant data are extracted from the database.
  3. The extracted data must be organised as a content object.
  4. The content object is transmitted to the client environment for display.

  Errors can, and do, occur as a consequence of ***each*** of the above-listed steps.

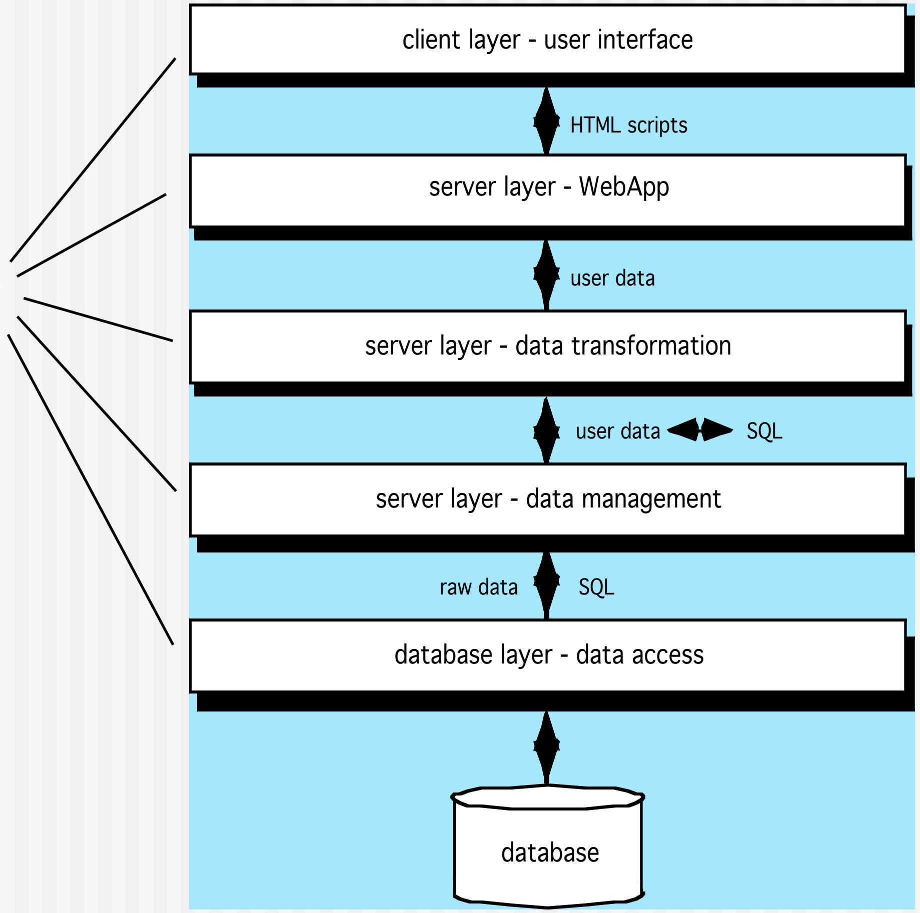
# Database Testing

- Database testing is complicated by a variety of factors, including:
  1. *The original client-side request for information is rarely presented in the form, (e.g. SQL), that can be input to a DBMS.* Thus, tests should be designed to uncover errors made in translating the user's request into a form that can be processed by DBMS.
  2. *The database may be remote to the server that houses the WebApp*. Thus, tests that uncover errors in communication between WebApp and the remote database must be developed.

## Database Testing

3. *Raw data acquired from the database must be transmitted to the WebApp server and properly formatted for subsequent transmittal to the client.* Thus tests that demonstrate the validity of the raw data received by the WebApp server must be developed, and additional tests that demonstrate the validity of the transformations applied to the raw data to create valid content objects must also be created.
4. *The dynamic content object(s) must be transmitted to the client in a form that can be displayed to the end user.* Thus, a series of tests must be designed to (i) uncover errors in content object format, and (ii) test compatibility with different client environment configurations.

# Database Testing


Tests are defined for each layer
client layer - user interface
HTML scripts
server layer - WebApp
user data
server layer - data transformation
user data
SQL
server layer - data management
raw data
SQL
database layer - data access
database

## Database Testing

- *User interface layer* is tested to ensure that scripts are properly constructed for each user-query and properly transmitted to the server side.
- *WebApp layer* on the server side is tested to ensure that user data are properly extracted from scripts and properly transmitted to the data transformation layer on the server side.
- *Data transformation functions* are tested to ensure that the correct SQL is created and passed to appropriate data management components.

## User Interface Testing

- *Verification* and *Validation* of a WebApp user interface occurs at <u>three</u> distinct points:

1. During *requirements analysis*, *the interface model is reviewed* to ensure that it conforms to stakeholder requirements and to other elements of the requirements model.
2. During *design*, *the interface model is reviewed* to ensure that generic quality criteria, established for all user interfaces, have been achieved and that application-specific interface design issues have been properly addressed.
3. During *testing*, the focus shifts to *the execution of application-specific aspects of user interaction* as they are manifested by interface syntax and semantics. Testing also provides the final assessment of *usability.*

## User Interface Testing

- *Interface testing* exercises interaction mechanisms and validates aesthetic aspects of the user interface.
- Consider the following overall strategy:
  1. Uncover errors related to specific interface mechanisms, e.g. the way that data is entred in a form.
  2. Uncover errors in the way that the interface implements the *semantics of navigation, WebApp functionality, or content display*.

## User Interface Testing

Consider the following *tactical steps:*

- Interface features are tested to ensure that design rules, aesthetics, and related visual content is available for the user without error, e.g. use of colour, type fonts, borders, tables, . .. .
- Individual interface mechanisms are tested in a manner that is *analogous to unit testing*, e.g. exercise all forms, client-side scripting, streaming content, . . . .
- Each interface mechanism is tested within the context of a use-case or NSU for a specific user category – *analogous to integration testing,* e.g. interface mechanisms are integrated to allow a use case to be "executed"

## User Interface Testing

- The complete interface is tested against selected use-cases and NSUs to uncover errors in the semantics of the interface – *analogous to validation testing* as purpose is to demonstrate conformance to the specific use case or NSU semantics. Typically, at this stage, a series of usability tests is conducted.
- The interface is tested within a variety of environments (e.g., browsers) to ensure that it will be compatible. Such tests can also be considered to be part of *configuration testing.*

## Testing Interface Mechanisms

- *Links—each navigation link is tested* to ensure that the proper content object or function is reached. Tester should *build a list of all links* associated with the interface layout, e.g. menu bars, and then *execute each individually*. Additionally, links within each content object must be exercised to uncover bad URLs or links to improper content objects or functions. Finally, links to external WebApps should be tested for accuracy and also evaluated to determine the risk that they will become invalid over time.

## Testing Interface Mechanisms

- *Forms*—At a *macroscopic* level, tests are performed to ensure that:
  1. Labels correctly identify fields within the form and that mandatory fields are identified visually for the user.
  2. The server receives all information contained within the form and that no data are lost in the transmission between client and server.
  3. Appropriate defaults are used when the user does not select from a pull-down menu or set of buttons.
  4. Browser functions, e.g. "back arrow", do not corrupt data entered in a form.
  5. Scripts that perform error checking on data entered work properly, and provide meaningful error messages.

## Testing Interface Mechanisms

- At a *more targeted level,* tests should ensure that:

1. Form fields have proper width and data types.
2. The form establishes appropriate safeguards that preclude the user from entering text strings longer than some predefined maximum.
3. All appropriate options for pull-down menus are specified and ordered in a way that is meaningful to the end user.
4. Browser "auto-fill" features do not lead to data input errors.
5. Tab key/other designated key(s) initiates proper movement between form fields.

## Testing Interface Mechanisms

- *Client-side scripting*—*Black box tests* are conducted to uncover errors in processing as scripts are executed. Such tests are often coupled with *forms testing*, since script-input is often derived from data provided as part of forms processing. A compatibility test should be conducted to ensure that the scripting language chosen works properly with the environmental configuration(s) that supports the WebApp.
- *Dynamic HTML*—Each Web page that contains dynamic HTML is executed to ensure that the dynamic display is correct. Additionally, a compatibility test should be conducted to ensure that the dynamic HTML works properly in the environmental configuration(s) that supports the WebApp.

## Testing Interface Mechanisms

- *Client-side pop-up windows*—A series of tests ensures that:
  1. The pop-up is properly sized and positioned.
  2. The pop-up does not cover the original WebApp window.
  3. The aesthetic design of the pop-up is consistent with the aesthetic design of the interface.
  4. Scroll bars and other control mechanisms appended to the pop-up are properly located and function as required.

## Testing Interface Mechanisms

- *CGI scripts*—*Black box tests* are conducted with an emphasis on data integrity, (as data are passed to the CGI script), and script processing,(once validated data have been received). In addition, *performance testing* can be conducted to ensure that the server-side configuration can accommodate the processing demands of multiple invocations of CGI scripts.
- *Streaming content*—Tests should demonstrate that streaming data are up-to-date, properly displayed, and can be suspended without error and restarted without difficulty.

## Testing Interface Mechanism

- *Cookies*—Both server-side and client-side testing are required.

1. On the server side, tests should ensure that a cookie is properly constructed, (contains the correct data), and properly transmitted to the client side when specific content or functionality is requested. Additionally, the proper persistence of the cookie is tested to ensure that its expiration-date is correct.
2. On the client side, tests determine if the WebApp properly attaches existing cookies to a specific request, (sent to the server).

## Testing Interface Mechanisms-II

- *Application specific interface mechanisms*—Tests conform to a checklist of functionality and features that are defined by the interface mechanism, e.g. a shopping cart, credit card processing, or a shipping cost calculator.
- Consider the following checklist for *shopping cart functionality*, defined for an e-commerce application:

1. *Boundary test* the max and min number of items that can be placed in the shopping cart.
2. Test a "checkout" request for an empty cart.
3. Test proper deletion of an item from the cart.
4. Test to determine if a purchase empties the cart of its contents.
5. Test to determine the persistence of cart contents (should be specified as part of customer requirements).
6. Test to determine whether the WebApp can recall cart contents at some future data, (assuming that no purchase was made).

## Usability Tests

- *Design by WebE team … executed by end-users!*
- *Testing sequence …*
  - Define a set of usability testing categories and identify goals for each.
  - Design tests that will enable each goal to be evaluated.
  - Select participants who will conduct the tests.
  - Instrument participants' interaction with the WebApp while testing is conducted.
  - Develop a mechanism for assessing the usability of the WebApp
- Can occur at different levels of abstraction:
  - the usability of *a specific interface mechanism* (e.g., a form) can be assessed
  - the usability of *a complete Web page* (encompassing interface mechanisms, data objects and related functions) can be evaluated
  - the usability of *the complete WebApp* can be considered.

## Usability Testing

- First step is to identify a set of usability categories and then to establish testing objectives for each category.
- Consider the following *example* test categories and objectives:

1. *Interactivity* – Are interaction mechanisms, e.g. pull-down menus, buttons, pointers, easy to understand and use?
2. *Layout* – Are navigation mechanisms, content, and functions placed in a manner that allows the user to find them quickly?
3. *Readability* – Is text well written and understandable? Are graphic representations easy to understand?
4. *Aesthetics* – Do layout, colour, typeface, and related characteristics lead to ease of use? Do users "feel comfortable" with the look and feel of the WebApp?

## Usability Testing

5. *Display characteristics* – Does the WebApp make optimal use of screen size and resolution?
6. *Time sensitivity* – Can important features, functions, and content be used or acquired in a timely manner?
7. *Personalisation* – Does the WebApp tailor itself to the specific needs of different user categories or individual users?
8. *Accessibility* – Is the WebApp accessible to people who have disabilities?

## Usability Testing

**Note:**

A series of tests is designed within each of the above-listed categories.

**Constantine & Lockwood, 1999,** suggest the following list of interface features should be reviewed and tested for usability:

*Animation, Buttons, Colour, Control, Dialogue, Fields, Forms, Frames, Graphics, Labels, Links, Menus, Messages, Navigation, Pages, Selectors, Text, Toolbars.*

## Compatibility Testing

- Compatibility testing is to define a set of “commonly encountered” client side computing configurations and their variants
- Create a tree structure identifying
  - each computing platform
  - typical display devices
  - the operating systems supported on the platform
  - the browsers available
  - likely Internet connection speeds
  - similar information.
- Derive a series of compatibility validation tests
  - derived from existing interface tests, navigation tests, performance tests, and security tests.
  - intent of these tests is to uncover errors or execution problems that can be traced to configuration differences.

## Component-Level Testing

- Focuses on a set of tests that attempt to uncover errors in WebApp functions
- Conventional black-box and white-box test case design methods can be used
- Database testing is often an integral part of the component-testing regime

## Navigation Testing

- The following navigation mechanisms should be tested:
  - *Navigation links*—these mechanisms include internal links within the WebApp, external links to other WebApps, and anchors within a specific Web page.
  - *Redirects*—these links come into play when a user requests a non-existent URL or selects a link whose destination has been removed or whose name has changed.
  - *Bookmarks*—although bookmarks are a browser function, the WebApp should be tested to ensure that a meaningful page title can be extracted as the bookmark is created.
  - *Frames and framesets*—tested for correct content, proper layout and sizing, download performance, and browser compatibility
  - *Site maps*—Each site map entry should be tested to ensure that the link takes the user to the proper content or functionality.
  - *Internal search engines*—Search engine testing validates the accuracy and completeness of the search, the error-handling properties of the search engine, and advanced search features

# Testing Navigation Semantics-I

- Is the NSU achieved in its entirety without error?
- Is every navigation node (defined for a NSU) reachable within the context of the navigation paths defined for the NSU?
- If the NSU can be achieved using more than one navigation path, has every relevant path been tested?
- If guidance is provided by the user interface to assist in navigation, are directions correct and understandable as navigation proceeds?
- Is there a mechanism (other than the browser 'back' arrow) for returning to the preceding navigation node and to the beginning of the navigation path.
- Do mechanisms for navigation within a large navigation node (i.e., a long web page) work properly?
- If a function is to be executed at a node and the user chooses not to provide input, can the remainder of the NSU be completed?

## Testing Navigation Semantics-II

- If a function is executed at a node and an error in function processing occurs, can the NSU be completed?
- Is there a way to discontinue the navigation before all nodes have been reached, but then return to where the navigation was discontinued and proceed from there?
- Is every node reachable from the site map? Are node names meaningful to end-users?
- If a node within an NSU is reached from some external source, is it possible to process to the next node on the navigation path. Is it possible to return to the previous node on the navigation path?
- Does the user understand his location within the content architecture as the NSU is executed?

## Configuration Testing

- Server-side
  - Is the WebApp fully compatible with the server OS?
  - Are system files, directories, and related system data created correctly when the WebApp is operational?
  - Do system security measures (e.g., firewalls or encryption) allow the WebApp to execute and service users without interference or performance degradation?
  - Has the WebApp been tested with the distributed server configuration (if one exists) that has been chosen?
  - Is the WebApp properly integrated with database software? Is the WebApp sensitive to different versions of database software?
  - Do server-side WebApp scripts execute properly?
  - Have system administrator errors been examined for their affect on WebApp operations?
  - If proxy servers are used, have differences in their configuration been addressed with on-site testing?

## Configuration Testing

- Client-side
  - *Hardware*—CPU, memory, storage and printing devices
  - *Operating systems*—Linux, Macintosh OS, Microsoft Windows, a mobile-based OS
  - *Browser software*—Internet Explorer, Mozilla/Netscape, Opera, Safari, and others
  - *User interface components*—Active X, Java applets and others
  - *Plug-ins*—QuickTime, RealPlayer, and many others
  - *Connectivity*—cable, DSL, regular modem, T1
- The number of configuration variables must be reduced to a manageable number

## Security Testing

- Designed to probe vulnerabilities of the client-side environment, the network communications that occur as data are passed from client to server and back again, and the server-side environment
- On the client-side, vulnerabilities can often be traced to pre-existing bugs in browsers, e-mail programs, or communication software.
- On the server-side, vulnerabilities include denial-of-service attacks and malicious scripts that can be passed along to the client-side or used to disable server operations

## Performance Testing

- Does the server response time degrade to a point where it is noticeable and unacceptable?
- At what point (in terms of users, transactions or data loading) does performance become unacceptable?
- What system components are responsible for performance degradation?
- What is the average response time for users under a variety of loading conditions?
- Does performance degradation have an impact on system security?
- Is WebApp reliability or accuracy affected as the load on the system grows?
- What happens when loads that are greater than maximum server capacity are applied?

## Load Testing

- The intent is to determine how the WebApp and its server-side environment will respond to various loading conditions
  - *N*, the number of concurrent users
  - *T*, the number of on-line transactions per unit of time
  - *D*, the data load processed by the server per transaction
- Overall throughput, *P*, is computed in the following manner:
    - $P = N \times T \times D$

## Stress Testing

- Does the system degrade ʻgentlyʼ or does the server shut down as capacity is exceeded?
- Does server software generate "server not available" messages? More generally, are users aware that they cannot reach the server?
- Does the server queue requests for resources and empty the queue once capacity demands diminish?
- Are transactions lost as capacity is exceeded?
- Is data integrity affected as capacity is exceeded?
- What values of *N, T,* and *D* force the server environment to fail? How does failure manifest itself? Are automated notifications sent to technical support staff at the server site?
- If the system does fail, how long will it take to come back on-line?
- Are certain WebApp functions (e.g., compute intensive functionality, data streaming capabilities) discontinued as capacity reaches the 80 or 90 percent level?