Technologies and Tools, and Testing Web Apps

Week 12, 13

Agenda (Lecture)

- Technologies and Tools
- Testing Web Apps

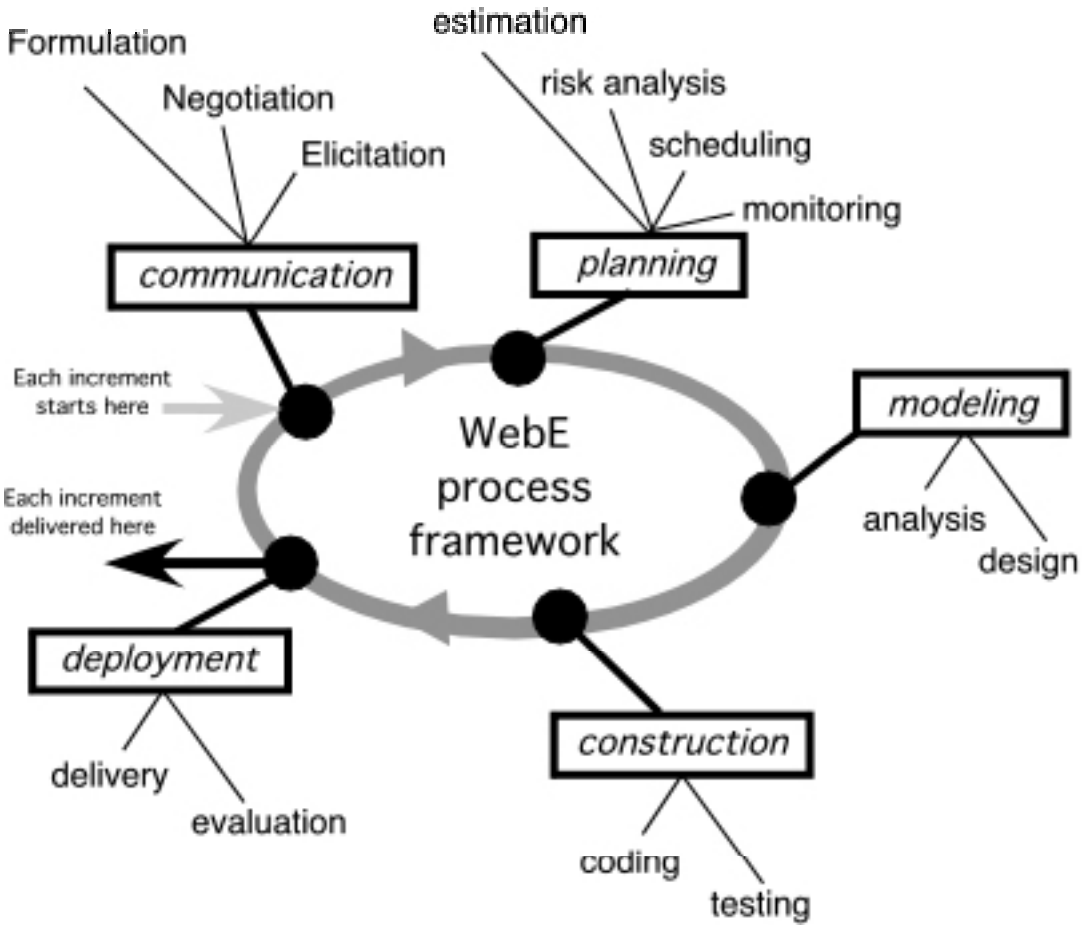
Agenda (Lab)

- Implement your web apps based on design documents
- Weekly progress report

Announcement

- Midterm II
  - Monday, 5/2
  - Short-answer questions and multiple choices
  - Scope
    - Chapters 8, 9, 10 and 11
    - Your project

WebE Process Activities & Actions



Technology and Tools

- ***Warning:*** This topic area is evolving very rapidly. The information in this area will certainly be out-of-date within 2 years – and is often out-of-date within months.
- A good Web Engineer should spend considerable time keeping themselves up-to-date with current trends!

Technology and Tools

- There are two main categories of technologies that we'll discuss:
  - **Implementation tools.** Includes technologies as diverse as Web application servers, content management systems, file-sharing systems, and security management
  - **Development tools.** Includes design modeling, issue tracking, and application testing

- We need to compartmentalize WebApp capabilities and the tools that allow us to achieve those capabilities:
  - Content storage
  - Content adaptation
  - Presentation
  - Presentation adaptation
  - Content structuring and navigation
  - Functionality (e.g., search and workflow management)

Tools: Open Source or Proprietary

- The choice between open-source and proprietary tools can become a significant issue
- In general, the choice between open-source and proprietary WebE technology and tools should be based on your answers to the following questions:
  - Does the tool meet the capabilities that are required and the functionality that is to be deployed?
  - Are the reported quality and extensibility adequate for your needs?
  - Does the evolutionary direction predicted for the tool meet your needs today and in the future?
  - Does the tool have adequate support facilities, online documentation, and help?
  - Does the cost of the tool fall within your project or organizational budget?

Application Frameworks

- A set of libraries and/or components that are used to implement the basic structure of an application
- Provide both an underlying architecture and substantial amounts of code to support this architecture.
  - mechanisms for managing content
  - interfacing with access control systems and databases
  - managing user sessions, and
  - the handling of presentation and styles.
- Simple frameworks have a single primary purpose, such as page generation from database content.
- Complex frameworks address a variety of features and needs.

Content Management

- The functionality supported by content management systems is very diverse (see CMS matrix at **www.cmsmatrix.org/**), and different content management systems support different capabilities:
  - Presentation templates, themes, and skins
  - Monitoring, statistics, and content tracking
  - Content staging and deployment
  - Security management to authenticate users and control access for both editing and viewing specified content
  - Support for diverse applications: wikis, discussion forums, guest books, event calendaring, FAQs, etc.
- More sophisticated CMSs provide version control capabilities (Chapter 16), enabling the WebE team to track changes to content and allowing the state of an application to be “wound back” to a previous version of the content.

Testing WebApps

- Testing is the process of exercising a WebApp with the intent of finding (and ultimately correcting) errors.
- Tests must be design to uncover errors in WebApps that are implemented in:
  - different operating systems
  - browsers [or other interface devices such as set-top boxes, personal digital assistants (PDAs), and mobile phones]
  - hardware platforms
  - communications protocols
  - “backroom” applications

The “Dimensions” of Quality - I

- Reviews and testing examine one or more of the following quality dimensions:
  - *Content* is evaluated at both a syntactic and semantic level.
    - At the syntactic level, spelling, punctuation, and grammar are assessed for text-based documents. At a semantic level, correctness (of information presented), consistency (across the entire content object and related objects), and lack of ambiguity are all assessed.

The “Dimensions” of Quality - II

- *Function* is tested to uncover errors that indicate lack of conformance to stakeholder requirements. Each WebApp function is assessed 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, and can be supported as new content or functionality is added.
  - *Usability* is tested to ensure that each category of user is supported by the interface and can learn and apply all required navigation syntax and semantics.

The “Dimensions” of Quality - III

- *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 and handles extreme loading without unacceptable operational degradation.
- *Usability* is tested to ensure that each category of user is supported by the interface and can learn and apply all required navigation syntax and semantics.

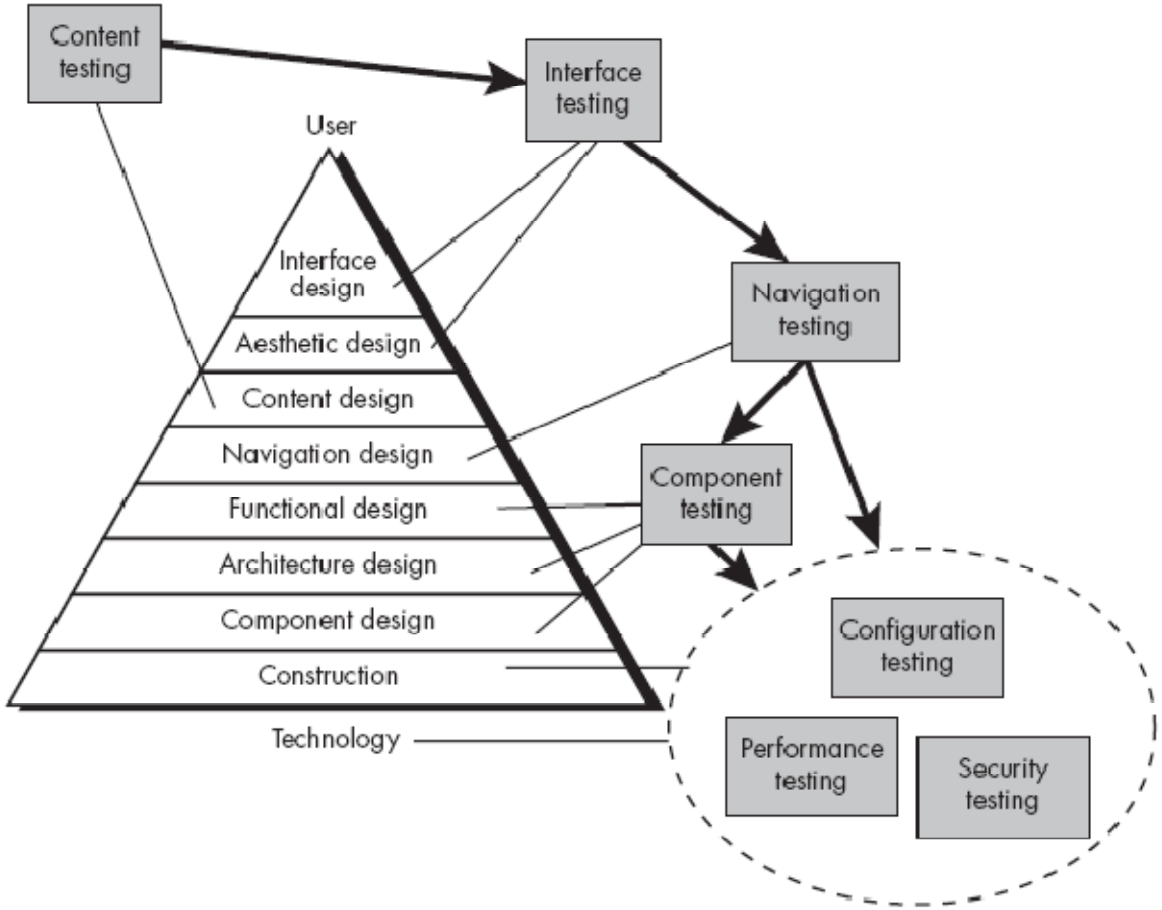
The "Dimensions" of Quality - IV

- *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 and handles extreme loading without unacceptable operational degradation.

Testing Strategy

1. The content model for the WebApp is reviewed to uncover errors.
2. The interface model is reviewed to ensure that all use cases have been accommodated.
3. The design model for the WebApp is reviewed to uncover navigation errors.
4. The user interface is tested to uncover errors in presentation and/or navigation mechanics.
5. Selected functional components are unit tested.
6. Navigation throughout the architecture is tested.
7. The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration.
8. Security tests are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment.
9. Performance tests are conducted.
10. 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 combines both reviews and the generation of executable test cases.
  - Reviews are applied to uncover semantic errors in content.
  - Executable testing is used to uncover content errors that can be traced to dynamically derived content that is driven by data acquired from one or more databases.

- 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.

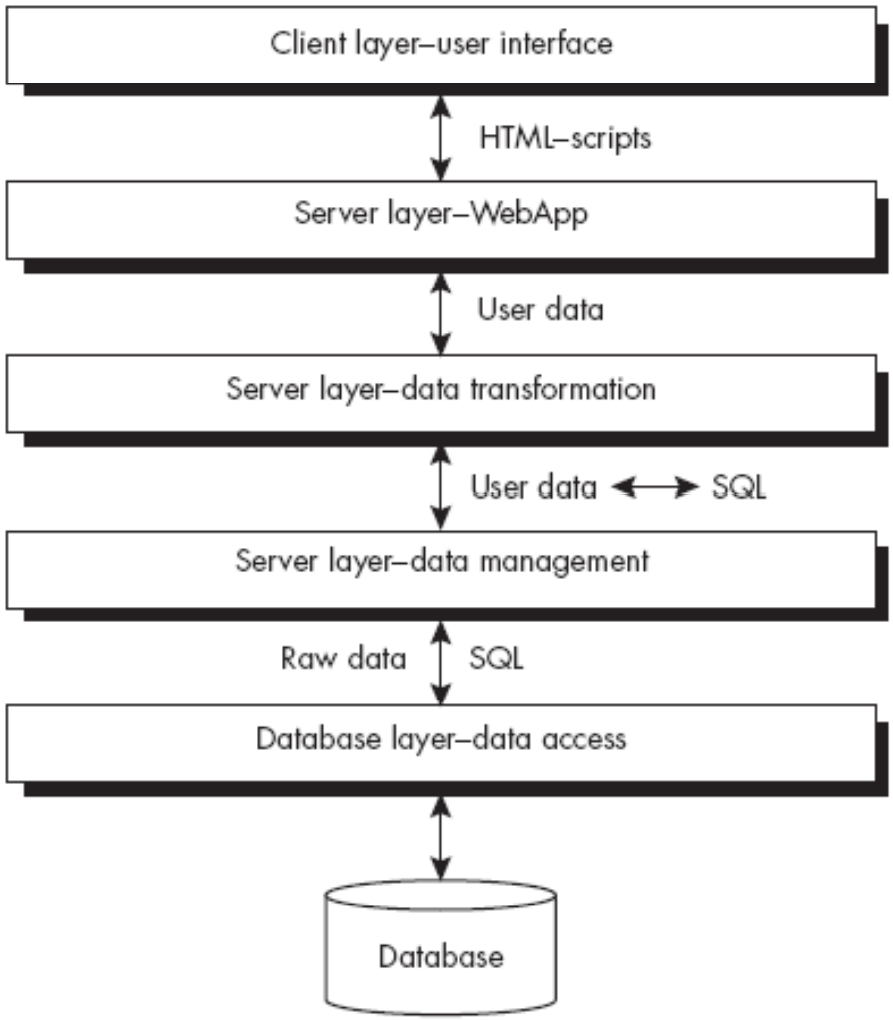
Content Testing - Checklist

- Is the information up to date and 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?
- Can the content be interpreted as being offensive or 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?

Content Testing – Dynamic Content

- When content is created dynamically using information maintained within a database, the following issues are considered:
  - The original client-side request for information is rarely presented in the form [e.g., structured query language (SQL)] that can be input to a database management system (DBMS).
  - The database may be remote to the server that houses the WebApp.
    - *What happens if the WebApp is accessible but the database is not?*
  - Raw data acquired from the database must be transmitted to the WebApp server and properly formatted for subsequent transmittal to the client.
  - The dynamic content object(s) must be transmitted to the client in a form that can be displayed to the end user.

Content Testing - Database



User Interface Testing

- Verification and validation of a WebApp user interface occurs at three distinct points in the WebE process.
  - During communication (Chapter 4) and modeling (Chapter 7), the interface model is reviewed to ensure that it conforms to customer requirements and to other elements of the analysis model.
  - During design (Chapter 9), the interface design 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.
  - During testing (Chapter 15), the focus shifts to the execution of application-specific aspects of user interaction as they are manifested by interface syntax and semantics. In addition, testing provides a final assessment of usability.

UI Testing Strategy

- Interface features are tested to ensure that design rules, aesthetics, and related visual content are available to the user without error.
- Individual interface mechanisms are tested in a manner that is analogous to unit testing.
- Each interface mechanism is tested within the context of a use case or navigation pathway for a specific user category.
- The complete interface is tested against selected use cases and navigation pathways to uncover errors in the semantics of the interface.
- The interface is tested within a variety of environments (e.g., operating systems, browsers) to ensure that it will be compatible.

User Interface – Testing specific elements (1)

- When a user interacts with a WebApp, the interaction occurs through one or more interface mechanisms. Each mechanism must be tested:
  - **Links.** Navigation mechanisms that link the user to some other content object or function.
  - **Forms.** A structured document containing blank fields that are filled in by the user.
  - **Client-side scripting**. A list of programmed commands in a scripting language (e.g., JavaScript) that handle information input via forms or other user interactions.
  - **Dynamic HTML**. Provides access to content objects that are manipulated on the client side using scripting or cascading style sheets (CSSs).
  - **Client-side pop-up windows**. Small windows that pop up without user interaction.
  - **Server-side scripts**. Black-box tests are conducted with an emphasis on data integrity and script processing once validated data has been received. In addition, performance testing can be conducted.

User Interface – Testing specific elements (2)

- When a user interacts with a WebApp, the interaction occurs through one or more interface mechanisms. Each mechanism must be tested:
  - **Streaming and push content**. *Streaming content* is encountered when material (usually audio or video) is downloaded in a manner that allows it to be displayed while it is still being downloaded (rather than having to wait for the entire content to be downloaded). *Push content* is encountered when content objects are downloaded automatically from the server side rather than waiting for a request from the client side. Both streaming and push content present testing challenges.
  - **Cookies.** A block of data sent by the server and stored by a browser as a consequence of a specific user interaction. The content of the data is WebApp-specific (e.g., user identification data or a list of items that have been selected for purchase by the user).
  - **Application-specific interface mechanisms**. Include one or more "macro" interface mechanisms such as a shopping cart, credit card processing, or a shipping cost calculator.

Usability Testing

- Similar to interface semantics testing in the sense that it evaluates:
  - the degree to which users can interact effectively with the WebApp
  - the degree to which the WebApp guides users' actions, provides meaningful feedback and enforces a consistent interaction approach.
- Determines the degree to which the WebApp interface makes the user's life easy

Usability Testing

- 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.

- Log the details of the participants' interaction with the WebApp while testing is conducted.

- Develop a mechanism for assessing the usability of the WebApp.
- Usability testing can occur at a variety of different levels of abstraction:
  - (1) the usability of a specific interface mechanism (e.g., a form) can be assessed
  - (2) the usability of a complete Web page (encompassing interface mechanisms, data objects, and related functions) can be evaluated, or
  - (3) the usability of the complete WebApp can be considered.

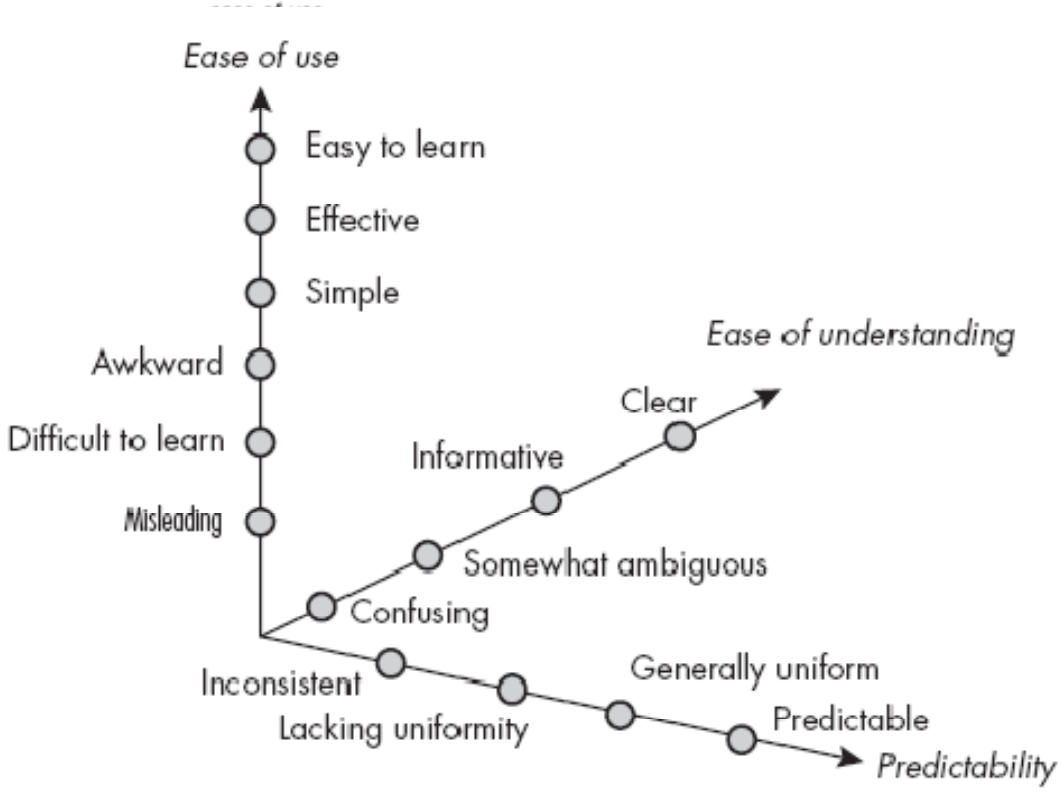
Usability Test Categories

- **Interactivity.** Are interaction mechanisms (e.g., pull-down menus, buttons, pointers) easy to understand and use?
- **Layout.** Are navigation mechanisms, content, and functions placed in a manner that allows the user to find them quickly?
- **Readability.** Is text well written and understandable? Are graphic representations intuitive and easy to understand?
- **Aesthetics.** Do the layout, color, typeface, and related characteristics lead to ease of use? Do users "feel comfortable" with the look and feel of the WebApp?
- **Display characteristics.** Does the WebApp make optimal use of screen size and resolution?
- **Time sensitivity.** Can important features, functions, and content be used or acquired in a timely manner?
- **Personalization.** Does the WebApp appropriately tailor itself to the specific needs of different user categories or individual users?

Usability Evaluation: Checklist

- Is the system usable without continual help or instruction?
- Do the rules of interaction help a knowledgeable user to work efficiently?
- Do interaction mechanisms become more flexible as users become more knowledgeable?
- Has the system been tuned to the physical and social environment in which it will be used?
- Are users aware of the state of the system? Do users know where they are at all times?
- Is the interface structured in a logical and consistent manner?
- Are interaction mechanisms, icons, and procedures consistent across the interface?
- Does the interaction anticipate errors and help the user correct them?
- Is the interface tolerant of errors that are made?
- Is the interaction simple?

Qualitative Assessment of Usability



Compatability Testing

- WebApps operate in complex (and often unpredictable) environments
  - Different browsers, screen resolutions, operating systems, plug-ins, access bandwidths, etc.
- Serious errors can be caused by obscure combinations
- Most common problem is deterioration in usability:
  - Download speeds may become unacceptable
  - Missing plug-ins may make content unavailable
  - Browser differences can change page layout or legibility
  - Forms may be improperly organized.
- Compatibility testing strives to uncover these problems before the WebApp goes online.
  - First step is to define a set of “commonly encountered” client-side configurations and their variants.
  - Next, derive a series of compatibility validation tests (from existing interface tests, navigation tests, performance tests, and security tests).

Component-Level Testing

- *Component-level testing,* also called *function testing,* focuses on a set of tests that attempt to uncover errors in WebApp functions
- Applies the following test-case design methods:
  - Equivalence partitioning
  - Boundary value analysis
  - Path testing

Selecting Components to Test

- Which functionality in the Web site is most critical to its purpose?
- Which areas of the site require the heaviest database interaction?
- Which aspects of the site's CGI, applets, ActiveX components, and so on are most complex?
- What types of problems would cause the most complaints or the worst publicity?
- What areas of the site will be the most popular?
- What aspects of the site have the highest security risks?

Navigation Testing - I

- Each of the following navigation mechanisms should be tested [Spl01]:
  - **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 nonexistent 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 and that dynamic pages are bookmarked appropriately.

Navigation Testing - II

- **Frames and framesets.** Each frame contains the content of a specific Web page; a frameset contains multiple frames and enables the display of multiple Web pages at the same time.
- **Site maps.** A site map provides a complete table of contents for all Web pages.
- **Internal search engines.** An internal (local) search engine allows the user to perform a key word search within the WebApp to find needed content.

Navigation Semantics

- As navigation design is conducted, you create “a set of information and related navigation structures that collaborate in the fulfillment of a subset of related user requirements” [Cac02].
- These are sometimes referred to as *navigation semantic units* (NSUs) and are defined by a set of navigation paths (called “ways of navigating”) that connect navigation nodes (e.g., Web pages, content objects, or functionality).
- Taken as a whole, each NSU allows a user to achieve specific requirements defined by one or more use cases for a user category.
- Navigation testing exercises each NSU to ensure that these requirements can be achieved.

Navigation Semantic Testing - I

- Is the NSU achieved in its entirety without error?
- Is every *navigation node* (a destination defined for an 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 (e.g., anchor point links for 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?

Navigation Semantic Testing - 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?
- Do users understand their location within the content architecture as the NSU is executed?

Configuration Testing

- Configuration variability and instability are important factors that make Web engineering a challenge.
  - Hardware, operating system(s), browsers, storage capacity, network communication speeds, and a variety of other client-side factors are difficult to predict for each user.
- The job of configuration testing is to test a set of probable client-side and server-side configurations to ensure that the user experience will be the same on all of them and to isolate errors that may be specific to a particular configuration.

Testing Strategy

- **Server-side.** configuration test cases are designed to verify that the projected server configuration [i.e., WebApp server, database server, operating system(s), firewall software, concurrent applications] can support the WebApp without error.

- **Client-side.** On the client side, configuration tests focus more heavily on WebApp compatibility with configurations that contain one or more permutations of the following components:
  - **Hardware.** CPU, memory, storage, and printing devices
  - **Operating systems.** Linux, Macintosh OS, Microsoft Windows, a mobile-based OS
  - **Browser software.** FireFox, Internet Explorer, Safari, Mozilla/Netscape, Opera, and others
  - **User interface components.** Active X, Java applets, and others
  - **Plug-ins.** QuickTime, RealPlayer, and many others
  - **Connectivity.** Cable, DSL, regular modem, industry-grade connectivity (e.g., T1 lines)

Security and Performance Testing

- Security and performance testing address the three distinct elements of the WebApp infrastructure
  - the server-side environment that provides the gateway to Internet users
  - the network communication pathway between the server and the client machine
  - the client-side environment that provides the end user with a direct interface to the WebApp.
- **Security testing** focuses on unauthorized access to WebApp content and functionality along with other systems that cooperate with the WebApp on the server side.
- **Performance testing** focuses on the operating characteristics of the WebApp and on whether those operating characteristics meet the needs of end users.

Security Testing

- One or more of the following security elements is implemented [Ngu01]:
  - **Firewalls.** A filtering mechanism that is a combination of hardware and software that examines each incoming packet of information to ensure that it is coming from a legitimate source, blocking any data that are suspect.
  - **Authentication.** A verification mechanism that validates the identity of all clients and servers, allowing communication to occur only when both sides are verified.
  - **Encryption.** An encoding mechanism that protects sensitive data by modifying it in a way that makes it impossible to read by those with malicious intent. Encryption is strengthened by using *digital certificates* that allow the client to verify the destination to which the data are transmitted.
  - **Authorization.** A filtering mechanism that allows access to the client or server environment only by those individuals with appropriate authorization codes (e.g., user ID and password).
- Security tests should be designed to probe each of these security technologies in an effort to uncover security holes that can be exploited by those with malicious intent.

Performance Testing

- Objectives:
  - 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?
  - What is the impact of poor performance on company revenues?
- ***Load testing*** determines how the WebApp and its server-side environment will respond to various loading conditions.
- ***Stress testing*** is a continuation of load testing, but in this instance the variables, *N, T,* and *D* are forced to meet and then exceed operational limits.