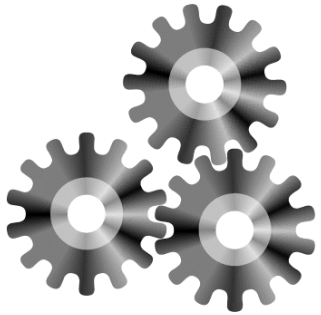


Software Engineering

Lecture 10

Testing Web Applications (Part-1) and Technical Debt (Part-2)

Testing the Web application

- There is always an urgency to get our WebApp finished quickly for a range of reasons such as:
  - Business pressure from within
  - Market competition
- As a result testing gets sidelined.
- But doing so can be a huge mistake!

Testing shouldn't wait until the project is finished. Start testing before you write one line of code. Test constantly and effectively, and you will develop a much more durable website.

- You cant do traditional testing in design phases – but you can do things like pair walkthroughs! (chapter 5)

Different types of testing

- We will cover more of the testing process for WebApps, and will endeavour to look at:
  - Content testing
  - User interface testing
  - Navigation testing
  - Component-level testing:
    - Black and White box testing, unit testing
    - Integration testing
  - Compatibility and Configuration testing
  - Security testing
  - Performance testing

Testing Quality Dimensions-1

- *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 vagueness 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.

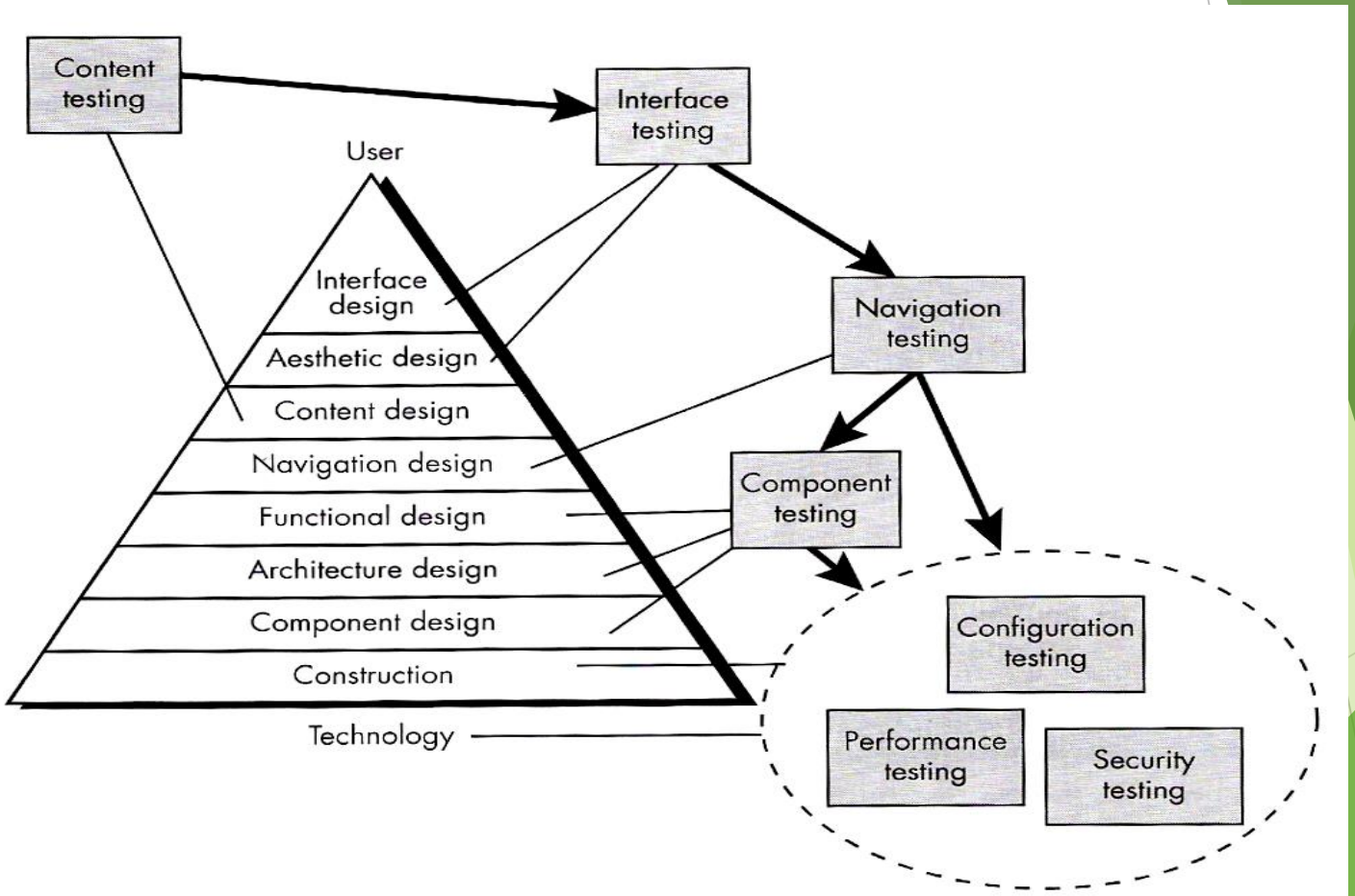
Testing Quality Dimensions-2

- *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 pathways 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

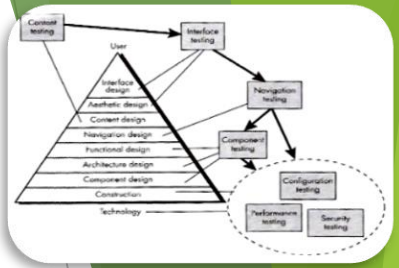
Testing Quality Dimensions-3

- *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.

The Testing Process



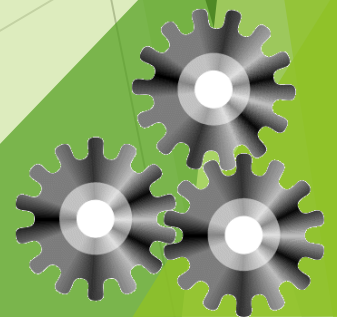
The Testing Process



- The WebApp testing process works with the WebApp design pyramid – to be unified!
  - Helps us to figure out when aspects should be tested.
- It describes the key actions! E.g. Content testing
- As the testing process flows from left to right and top to bottom – various visible elements of the WebApp design are tested first.
  - Followed by infrastructure design elements.
- *Lets take a look at the Pyramid and then decipher each key design action.*

Content Testing

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



Usability Tests

- Design by WebE team and executed by end-users
- Testing sequence
- Define a set of usability testing categories and identify goals for each.
  1. Design tests that will enable each goal to be evaluated.
  2. Select participants who will conduct the tests.
  3. Use tools to analyse participants' interaction with the WebApp while testing is conducted.
  4. Develop a mechanism for assessing the usability of the WebApp
- Different levels of usability:
  - 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.

User Interface Testing

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

Testing Interface Mechanisms-1

- *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. The data contained in the fields are used as input to one or more WebApp functions.
- *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*—leads to content objects that are manipulated on the client side using scripting or cascading style sheets (CSS).
- *Client-side pop-up windows*—small windows that pop-up without user interaction. These windows can be content-oriented and may require some form of user interaction.

Testing Interface Mechanisms-2

- *CGI scripts*—a common gateway interface (CGI) script implements a standard method that allows a Web server to interact dynamically with users (e.g., a WebApp that contains forms may use a CGI script to process the data contained in the form once it is submitted by the user).
- *Streaming content*—rather than waiting for a request from the client-side, content objects are downloaded automatically from the server side. This approach is sometimes called “push” technology because the server pushes data to the client.
- *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.

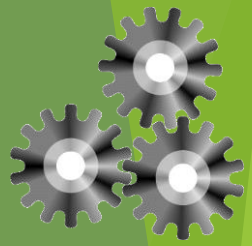
Navigation Testing

- 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

- 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.
- 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?

Component-Level Testing



- Also referred to as function testing.
- Focuses on a set of tests that attempt to uncover errors in WebApp functions.
- Every WebApp function is a software module (with a range of programming/scripting languages).
  - And can be tested using conventional black-box and white-box test case design methods.
- Testing functionality is an integral part of the component-testing regime.
  - E.g. Testing input values and expected outputs provided by the function.

Component-Level Testing

Black-box testing:, also called *behavioral testing*, focuses on the functional requirements of the software.

That is, black-box testing techniques enable you to derive sets of input conditions that will fully exercise all functional requirements for a web applictaion.

Black-box testing attempts to find errors in the following categories:

(1) incorrect or missing functions,

(2) interface errors,

(3) errors in data structures or external database access,

(4) behavior or performance errors, and

(5) initialization and termination errors.

Tests are designed to answer the following questions:

- How is functional validity tested?
- How are system behavior and performance tested?
- What classes of input will make good test cases?
- Is the system particularly sensitive to certain input values?
- How are the boundaries of a data class isolated?
- What data rates and data volume can the system tolerate?
- What effect will specific combinations of data have on system operation?

Component-Level Testing

Using white-box testing methods, you can derive test cases that

(1) guarantee that all independent paths within a module have been exercised at least once,

(2) exercise all logical decisions on their true and false sides,

(3) execute all loops at their boundaries and within their operational bounds, and

(4) exercise internal data structures to ensure their validity.

(5)Judging test suite thoroughness based on the *structure* of the program itself

- If part of a program is not executed by any test case in the suite, faults in that part cannot be exposed
- But what's a "part"?
  - Typically, a control flow element or combination:
  - Statements (or CFG nodes), Branches (or CFG edges)
  - Fragments and combinations: Conditions, paths

Component-Level Testing

#Statement testing

- Adequacy criterion: each statement (in the CFG) must be executed at least once
- Coverage: $\frac{\text{\# executed statements}}{\text{\# statements}}$

- **#Branch testing**
- Adequacy criterion: each branch (in the CFG) must be executed at least once
- Coverage: $\frac{\text{\# executed branches}}{\text{\# branches}}$

#Condition testing: Branch coverage exposes faults in how a computation has been decomposed into cases, Condition coverage considers case analysis in more detail, *individual conditions* in a compound boolean expression are evaluated.

- Adequacy criterion: each basic condition must be executed at least once
- Coverage: $\frac{\text{\# truth values taken by all basic conditions}}{2 * \text{\# basic conditions}}$

Component level testing is often called as Unit testing

- Each module is tested alone in an attempt to discover any errors in its code, also called module testing.
- This can be individual function or methods within an object, or functions in client/server side scripts
- Component can be an object classes that have several attributes and methods or a composite component made up of several different objects or functions

Integration testing

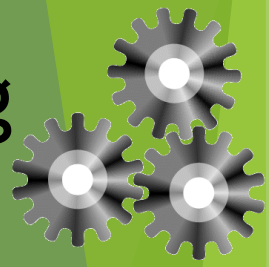
- Integration Testing
  - Tests the behaviour of a group of modules or methods
  - The error could not be detected in unit test, but from some other problem as: Interface incompatibility: passes a variable of wrong data type to subordinate module
  - Run-time exception: any error happen in the run-time (e.g. out of memory)
  - Unexpected state interaction: the states of two/three modules interact to cause a complex failure

Compatibility and configuration Testing

- Compatibility testing is to define a set of “commonly encountered” client side computing configurations and their alternatives.
- Create a tree structure identifying
  1. Each computing platform
  2. Typical display devices
  3. The operating systems supported on the platform
  4. The browsers available
  5. Likely Internet connection speeds
  6. 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.

Compatibility and Configuration Testing



- Web engineering can be challenging due to configuration variability and instability.
  - Why? Because hardware, operating system, browsers, storage capacity, network communication speeds and various other client side issues are difficult to predict.
- Additionally the configuration for a given user can change often (e.g. operating system updates, new internet provider / speeds).
  - The end result is a client side environment which may be prone to errors that can impact our WebApp.
- The goal of configuration testing is not to test every possible configuration but….
  - It is to test a set of probable client-side and server-side configurations to ensure that the user experience will be the same for each and errors isolated.

Compatibility and 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?
    - The WebApp components are shared among multiple servers to improve efficiency and performance.
  - 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?

Compatibility and 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

Security Testing

- *To protect against these, security elements are implemented:*
- *Firewall*—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.

- *Authorization*—a filtering mechanism that allows access to the client or server environment only by those individuals with appropriate authorization codes (e.g., userID and password)

Performance Testing

- Performance testing focuses on the operating characteristics of the WebApp and on whether those operating characteristics meet the needs of the end users.
- Nothing is more frustrating for users than a WebApp that takes minutes to load content!
  - It can put us at a disadvantage if there are competitor sites which load in seconds.
  - Users don't want to see a 'server busy message' or inconsistent load times (fast in some situations and massively slow in others).
- Performance testing is used to uncover performance problems (degraded client/server performance) that can result from lack of:
  - Server side resources, inappropriate bandwidth, inadequate database capabilities, faulty or weak operating system capabilities, poorly designed WebApp functionality and other hardware/software issues.

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 E.g. DoS attacks?
- 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?

Additional Readings

R. S. Pressman and D. Lowe: ***Web Engineering, A Practitioner's Approach,*** McGraw-Hill, 2009.

- Chapter 15: Testing WebApps

(concentrate on the topics covered in the lecture)