ST Week 11 Notes

L1: Web Apps Intro

- Introduction to relevant aspects of web applications.
- Issues in testing of web applications.
- Testing static hyper text web sites.
- Testing dynamic web applications.
  - Client-side testing of web applications.
  - Server-side testing of web applications.

- A web application is a program that is deployed on the web.
  - Uses HTML as the user interface.
  - Web-deployment means they are available worldwide.
  - They accept requests through HTTP and return responses.
- It is like a client-server application where the client runs in a web browser.
- Examples: web mail, online retail sales, wikis etc.

features of web apps.

- Composed of independent, loosely coupled software components.
  - All communication is through messages.
  - Web application messages always go through clients.
  - The only shared memory is through the session object - which is very restricted.
  - The definition of state is quite different.
- Inherently concurrent and often distributed.
- Most components are relatively small.
- Uses numerous new technologies, often mixed together.
  - JSP, ASP, Java, JavaBeans, JavaScript, Ajax, PHP etc.

types

- Bundled: Pre-installed on computer.
- Shrink-wrap: Bought and installed by end-users.
- Contract: Purchaser pays developer to develop and install, usually for a fixed price.
- Embedded: Installed on a hardware device, usually with no direct communication with user.
- Web: Executed across the Internet through HTTP.

Typically, a 3-tier application is developed: The three tiers are presentation, application and storage.

↓ further breakdown of application tiers.

- Presentation layer: HTML, output and UI.
- Data content layer: Computation, data access.
- Data representation layer: In-memory data storage.
- Data storage layer: Permanent data storage.

Testing each layer separately and their integration is also important.

- Web applications are heterogeneous, dynamic and must satisfy very high quality attributes.
- Use of the Web is hindered by low quality web sites and applications.
- Security breaches on web applications are also a major concern.
- HTTP is a stateless protocol.
  - Each request is independent of previous request.
  - *State* is managed by using *cookies*,, *session objects* etc.
- Servers have little information about where a request comes from.
- Web site software is extremely loosely coupled.
  - Coupled through the Internet - separated by space.
  - Coupled to diverse hardware devices.
  - Written in diverse software languages.
- Users can change the flow of control arbitrarily
  - Back button, forward button, URL rewriting, refresh.

→ with the help of buttons

- A web service is a system of software that allows machines to interact with each other through a network.
- Typically used to achieve re-usability of application components like weather reports, currency converters etc.
- Many web services don't have user interfaces.
- A web application is a complete application with a user interface, web services could be used for specific purposes like transfer of data or communication in a web application.

The issues for testing web applications can be divided into three broad categories:

- Testing static hyper text web sites.
- Testing dynamic web applications.
- Testing web services: Not considered in this course.

- A web page contains HTML content that can be viewed in a single browser window.
  - It may be stored as a static HTML file, or may be dynamically generated by software like JSP, ASP etc.
- A web site is a collection of web pages and associated software elements that are related semantically by content and syntactically through links and other control mechanisms.
- A static web page is unvarying and the same to all users.
  - Usually stored as a HTML file on a server.
- A dynamic web page is created by a program on demand.
  - Its contents and structure maybe dependent on previous inputs from the user, the state on the web server and other inputs like user's location, time etc.

A test case for a web application is a sequence of interactions between components on clients and servers.
They are paths of transitions through the web applications.
Generating these paths can be challenging for dynamic web pages.

Static hypertext website testing

- This is *not* program testing, but checking that all the HTML connections are valid.

→ the a tags in HTML.

- The main issue to test for is dead links, i.e., links to URLs that are no longer valid.
- We should also evaluate several non-functional parameters:
  - Load testing
  - Performance evaluation
  - Access control issues

- The usual model is that of a graph.
  - Nodes are web pages.
  - Edges are HTML links.
- The graph is built by starting with an introductory page and recursively doing BFS of all links from that page.
- The graph is then tested for edge coverage: traverse each edge (link) in the graph.

L2, 3: Client Side Testing

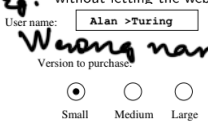
Testing dynamic web applications.
- Client-side testing.
- Server-side testing.

- For web applications, the user interface is on the client and the actual software is on the server.
- Clients and server are separated.

→ geographically

- Tester typically has no access to data, state or the source code on the server.
- Clients provide inputs to web software residing on a server.
- Test inputs: HTML form elements.
  - Text boxes, buttons, drop-down lists, links etc.
- Inputs can be generated or chosen.
  - Supplied by the tester.
  - Generated randomly.
  - Generated from user-session data collected from previous users of the software.
  - Bypass testing: Values that violate constraints on the inputs, as defined by client-side information.
- Web applications impose constraints on inputs through HTML forms.
- Constraints come in two forms:
  - Client-side script validation: Programs run on the client to check the syntax of input data before sending it to server.
  - Uses explicit attributes associated with HTML form fields. E.g., a text box can be set to only allow strings up to a maximum length.
- Bypass testing creates inputs that *intentionally* violate these validation rules and constraints.
  - Created inputs are directly submitted to the web application without letting the web page validate them.

Eg.



Wrong name

- User name should be plain text only.
- Age should be between 18 and 110.

- The basic idea in bypass testing is to let a tester *save* and *modify* the HTML.
- This way, client side checking/validation done routinely is by-passed and the modified data is sent to server.
- It can be used to see if the server crashes on the modified data.
  - Checks for security and robustness.
  - Also checks for common mistakes in inputs.
- Bypass testing modifies inputs.
- Can be done at the client side or server side.
- Client side inputs are *safer* and *easier* to handle.
- Server side inputs can be modified too, but, can be risky if they *corrupt* data in the server.

- Client side input validation is performed by HTML form controls, their attributes and client side scripts that access DOM.
- Validation types are categorized as HTML and scripting.
  - HTML supports syntactic validation.
  - Client scripting can perform both syntactic and semantic validation.

HTML	Scripting constraints
Length (max input characters)	Data Type (e.g. integer check)
Value (preset values)	Data Format (e.g. ZIP code format)
Transfer Mode (GET or POST)	Data Value (e.g. age value range)
Field Element (preset fields)	Inter-Value (e.g. credit # + expiry date)
Target URL (links with values)	Invalid Characters (e.g. <,&)

Diff. types of Δs that can be made —

- Violate size restrictions on strings
- Introduce values not included in static choices
  - Radio boxes
  - Select (drop-down) lists
- Violate hard-coded values
- Use values that JavaScript flag as errors
- Change "transfer mode" (get, post, ...)
- Change destination URLs

- Data type conversion
- Data format validation
- Inter-field constraint validation
- Inter-request data fields (cookies, hidden)

Test cases to a web application are HTML files.
- A static file or a dynamic file with two types of inputs:
  - HTML links and forms.
  - A form input unit is a HTML form that specifies the server software component as the `action` attribute within the `form` tag. The input data corresponds to all the input fields within the form.
  - A link input unit is a HTML link in an `<A>` tag, with a target URL specified as an attribute.
- Transfer mode (GET, POST) is also a part of the input, links typically generate only GET requests.

- HTML inputs can include more than one form and each form can include many input fields.
- Automatic input generation is difficult, the various inputs can be very large in number.
- Many web pages contain redundant (identical) input forms, they can be *composed* or merged before generating inputs through bypass testing. For e.g., many web pages contain the form for searching once at the top of the page and again at the end of the page.
- Similarly, some inputs can be optional and they need not be considered when the matching data is not considered.
- These help to reduce the number of different inputs.

Value level bypass testing tries to verify if a web application adequately evaluates invalid inputs.

- Data type conversion violation.
- Built-in length restriction violation.
- Built-in value restriction violation.
- Special input values, especially those that could corrupt data and cause a security vulnerability.

- Parameter level bypass testing tries to check for issues related to relationships among different parameters of an input.
  - For e.g., inter-value constraints like credit card number and an expiry date, built-in data access, built-in input field selection.
- It is difficult to determine the relationships among parameters for dynamically generated HTML files.
- We consider all the possible input formats (after applying possible merging), their relationships and generate violating inputs.
  - Empty input patterns.
  - Universal input patterns for all possible violations.
  - Differential input pattern: Valid values for all parameters along with a value for one parameter that is invalid.

Control flow level bypass testing tries to verify web applications by executing test cases that break the normal execution sequence.

- Begin by identifying "normal" control flow in the HTML page, can be done by doing simple parsing.
- Need to extend it to identify normal control flows for all possible inputs (can be difficult).
- Two types of control flow alterations:
  - Backward and forward control flow alteration.
  - Arbitrary control flow alteration.

- A testing approach that uses data captured during user sessions to create test cases.
- Reduces the effort involved when test engineers are required to generate test cases.
- Empirically proven to find faults in web sites related to first time users, sudden spike in the number of users etc.

- A web server responds to client requests through HTML files.
- Requests could be just a URL or data sent through a form.
  - Data is a set of name-value pairs along with a POST/GET.
  - A database server is involved here.

Collect all client request information. This can be accomplished based on the underlying web application development technologies used.

- An Apache web server can be used to log all the received GET requests.
- All the name-value pairs can be captured by adding snippets to invoke a server-side logging script.
- Use Java servelet filters.
- Incorporate cookies into the HTML files that are loaded in an interaction.

Ensure that the web application performance does not get impacted.

- Three stand-alone variants of the basic approach involving direct use of the session data.
  1. Directly reuse entire sessions,
  2. Replay a mixture of sessions, and
  3. Replay sessions with some targeted modifications.
- Two hybrid variants that combine the basic approach with other functional testing techniques.

Let $U = \{u_1, u_2, \ldots u_m\}$ be a set of user sessions, with u_i consisting of n requests $r_1; r_2; \ldots r_n$, where each r_i consists of $url[name - value]^*$.
For simplicity, we define a user session as beginning when a request from a new IP address reaches the server and ending when the user leaves the website or the session times out.

1

- Simplest of all techniques.
- Transform each $u_i \in U$ into a test case by formatting each of its associated requests, $r_1, r_2, \ldots r_n$ into an http request that can be sent to a web server.
- The resulting test suite contains m test cases, one for each user session.

2

- Select an unused session u_a from U.
- Copy requests r_1 through r_i from u_a, where i is a random number, $1 \leq i \leq n$, into the test case.
- Randomly select session u_b from U, where $b \neq a$, and search for any r_j in u_b with the same URL as r_i.
- If an r_j with the same URL is not found in u_b, select another session u_b. If there is no such u_b, then consider direct reuse.
- If an r_j with the same URL as r_i is found in u_b, then add all the requests following r_j from u_b into the test case after r_i.
- Mark u_a as "used", repeat the process until no more unused sessions are available in U.

3

Main idea: Replay user sessions by modifying the input forms that can alter the behaviour of the web application.

- Select an unused session u_a from U.
- Randomly select an unused request r_i from u_a. If there are no more unused r_i in u_a, then reuse u_a directly as a test case.
- If r_i does not contain at least one name-value pair, mark r_i as used and repeat previous step.
- If r_i has one or more name-value pairs, then modify the name-value pairs.
  1. Create one test case for each name-value pair by deleting a random character in the value string.
  2. Create one test case by modifying the values of all the pairs at once by deleting a random character in each value string.
  3. .....
- Mark u_a as "used" and repeat the process until no more unused sessions are available in U.

The effectiveness of the test case for a web application can be determined by the response of the server on the test cases.
Broad categorization of server responses:

- Valid responses: Invalid inputs are adequately processed by the server.
  - Server provides an explicit message regarding the violation.
  - Server provides a generic error message.
  - Server ignores the invalid input.
- Faults and failures: Invalid inputs cause abnormal server error.
- Exposure: Invalid inputs are not recognized by the server and abnormal software behavior is exposed to users.
  - Could even result in corruption of data in the server.

- If server-side source code is available, we an use graph models to test the server.
- Control flow graph exhibits only *static* models, not effective for web applications.
- Presentation layer of a web application contains software that is useful to do testing.
- For software in the presentation layer, two graph models exist:
  - Component Interaction Model (CIM)
  - Application Transition Graph (ATG)
- Graph models of server software is restricted to presentation layer only.
- Two abstract graph models are available.
- Component Interaction Model (CIM): As a graph, CIM
  - Models individual components.
  - Combines atomic sections.
  - Intra-component.
- Application Transition Graph (ATG): As a graph, ATG is
  - Each node is one CIM.
  - Edges are transitions among CIMs.
  - Inter-component.

We need the notion of atomic sections to understand the graph models.

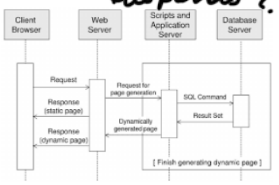
- An atomic section is a section of HTML with the property that if any part of the section is sent to a client, the entire section is.
  - May include JavaScript
  - All or nothing property
- A HTML file is an atomic section.
- A content variable is a program variable that provides data to an atomic section.
- Atomic sections may be empty.

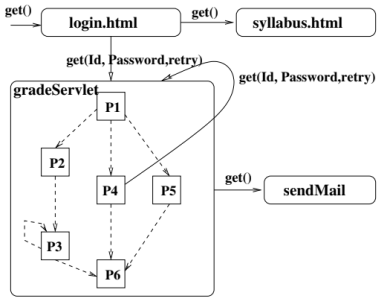
	PrintWriter out = response.getWriter();
P1 =	out.println ("<HTML>") out.println ("<HEAD><TITLE>"+title+"</TITLE></HEAD>)" out.println ("<BODY>")
	if(User) {
P2 =	out.println ("<CENTER> Welcome!</CENTER>");
	for (int i=0; i<myVector.size(); i++) if (myVector.elementAt(i).size > 10)
P3 =	out.println (" "+myVector.elementAt(i)+" ");
	else
P4 =	out.println(" "+myVector.elementAt(i)+" ");
	} else
P5 =	{ }
P6 =	out.println ("</BODY></HTML>"); out.close();

Atomic sections are coloured in red. P5 is an empty atomic section. Content variables are coloured green.

- Atomic sections are *combined* to model dynamically generated web pages.
- There are four ways to combine and generate component expressions.
  - Sequence: P1 · P2.
  - Selection: P1 | P2.
  - Iteration: P1*.
  - Aggregation: P1 {P2}, P2 is included inside P1.
- The example we just saw gives the component expression P1 · (P2 · (P3 | P4)* | P5) · P6.
- There are algorithms to generate these expressions automatically.

- Γ: Finite set of web components.
- Θ: Set of transitions among web software components
  - Includes type of HTTP request and data.
- Σ: Set of variables that define the web application state.
- α: Set of start pages.
- Γ = {login.html, gradeServlet, sendMail, syllabus.html},
- Starting page is "login.html",
- Θ = {login.html → syllabus.html[get,()], login.html → gradeServlet [get,(Id,Password,Retry)], gradeServlet.P4 → sendMail[get,()], gradeServlet.P4 → gradeServlet [get, (Retry)] }
- Σ = {Is, Password, Retry}.
- α = {login.html}.





There are five types of transitions/edges.

- Simple Link Transition: An HTML link (<A> tag).
- Form Link Transition: Form submission link.
- Component Expression Transition: Execution of a software component causes a component expression to be sent to the client.
- Operational Transition: A transition out of the software's control.
  - Back button, Forward button, Refresh button, User edits the URL, Browser reloads from cache
- Redirect Transition: Server side transition, invisible to user.

- Testing based on atomic sections, in turn, ATGs and CIMs provide a fundamental mechanism to model system level behavior of the presentation layer of web applications.
- Such testing can handle
  - Distributed integration.
  - Dynamically generated HTML web pages.
  - Operational transitions.
- The graphs CIM and ATG need to be generated by manual analysis of software source, can be non-trivial.
- Once generated, they provide a precise model of behavior of web software and is valuable for testing.
  - There is no notion of data-flow based testing that is known for web applications. Static definition+use is difficult to determine.
  - ATG cannot be automatically generated.
    - Issues involving multiple users, session data, concurrency, dynamic integration etc. cannot be handled.

- The Web provides a new way to deploy software
- Web applications:
  - offer many advantages
  - use many new technologies
  - introduce fascinating new problems
- Web Software Engineering is just beginning
- Two very useful techniques:
  - Bypass testing: Easy to automate, no source needed
  - Atomic sections: A fundamental model
- This is a very active research area.