TESTING WEB APPLICATONS

AGENDA:

- WHAT IS TESTING ?
- TESTING PROCESS
- TYPES OF TESTING
- MVC
- COST ESTIMATION FOR SOFTWARE PROJECTS
- ESTIMATION FOR AGILE AND WEBAPPS
- CONCLUSIONS
- Q & A

TESTING WEB APPLICATIONS

Testing:

Testing is the process of exercising software with the intent of finding of errors.

Web Testing:

Webapp testing is a collection of related activities with a single goal: to uncover errors in webapp content, function, usability, navigability, performance, capacity, and security.

Challenges in Web application testing:

Web based systems and applications reside on network and interoperate with many different

1. operating systems,
2. browsers,
3. hardware platforms,
4. communications protocols,

the search for errors represents a significant challenge

Dimensions of Quality for Web Applications:

Quality is incorporated into a web application as a consequence of good design. Reviews and Testing examine one or more of the following quality dimensions.

1.Content 2.Function 3.Structure

4.Usability 5.Navigability 6.Performance

7.Compatibility 8.Interoperability 9.Security.

Common Errors in Web Applications

- You often see a symptom of the error, not the error itself.
- It may be difficult or impossible to reproduce an error outside the environment in which the error was originally encountered.
- Many errors can be traced to the webapp configuration.
- 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 and others are attributable to the dynamic operating Environment.

Testing Approach for web application:

- The content model for the webapp is reviewed to uncover errors.

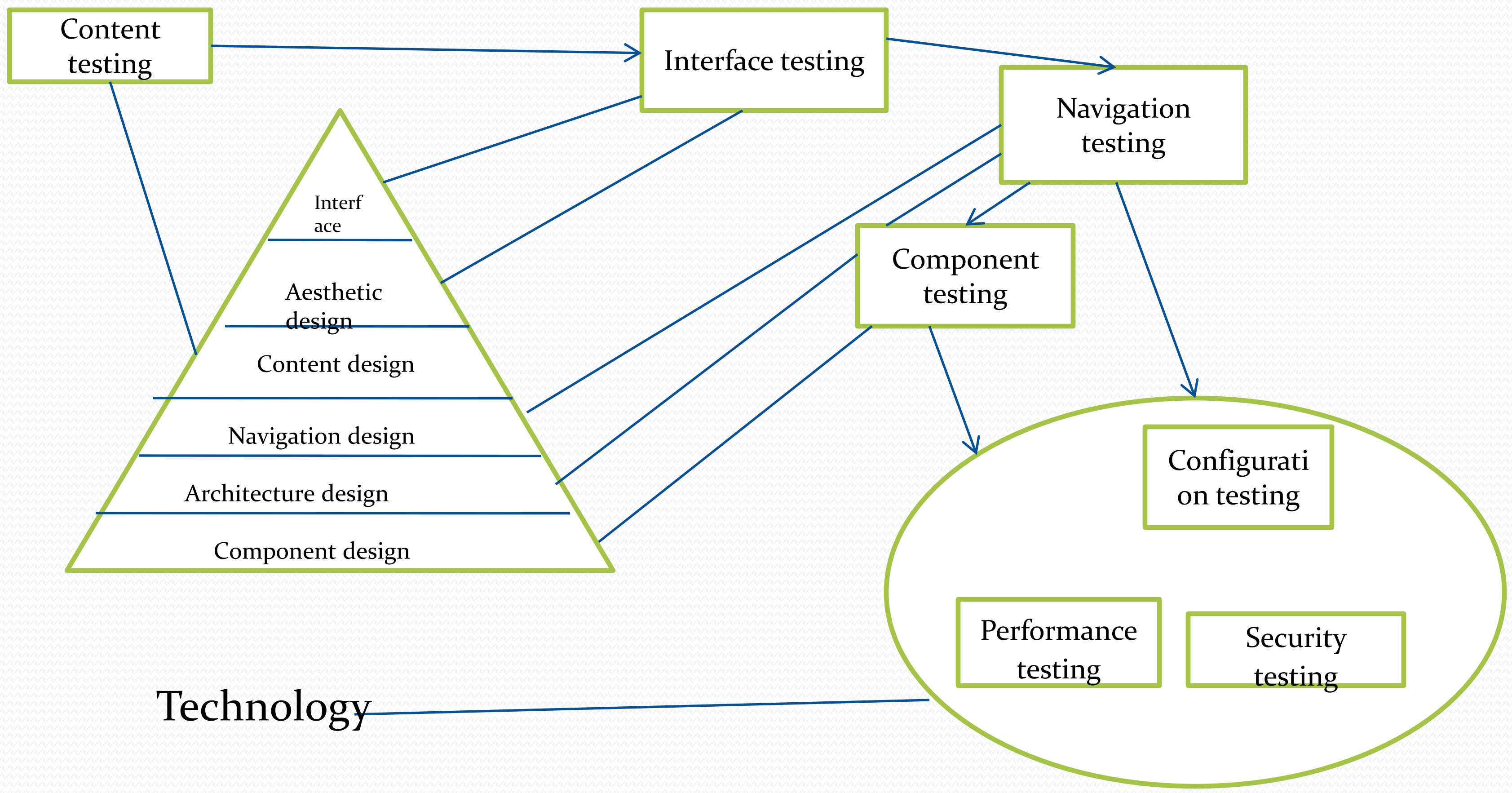
The design model for the webapp is reviewed to uncover navigation errors

- 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
- Functional components are unit tested.

Testing Approach for web application:

- Navigation throughout the architecture should be 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 should be 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 the webapp security, reliability, and performance.

Testing process



Content Testing:

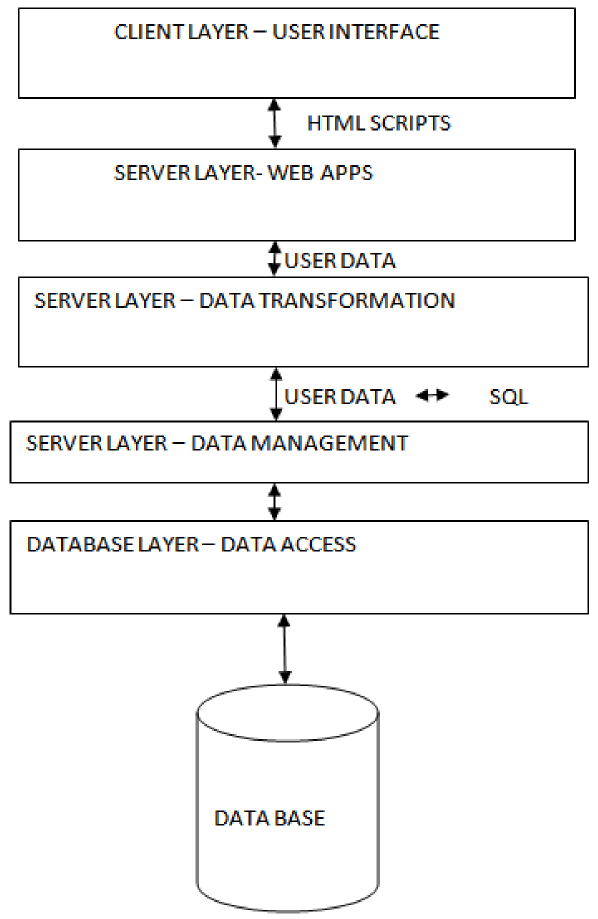
Content testing has three important objectives

- 1. To uncover syntactic errors(for eg., typos, grammar mistakes) in the text-based documents, graphical representations, and other media
- 2. To uncover semantic errors(i.e., focuses on the information presented within each content object)
- 3. To find errors in the organization or structure of the content that is presented to the end user.

Database testing:

- Tests should be designed to uncover errors made in translating the user's request into a form that can be processed by the DBMS.
- Tests that uncover errors in communication between the webapp and the remote database must be developed.
- Raw data acquired from the database must be transmitted to the webapp server and properly formatted for subsequent transmittal to the client.
- Tests that demonstrate the validity of the transformations applied to the raw data to create valid content objects must also be created.
- Content and compatibility testing will be done after the dynamic content object is transmitted to the client in a form that can be displayed to end user

Layers of interaction


CLIENT LAYER – USER INTERFACE
HTML SCRIPTS
SERVER LAYER- WEB APPS
USER DATA
SERVER LAYER – DATA TRANSFORMATION
USER DATA
SQL
SERVER LAYER – DATA MANAGEMENT
DATABASE LAYER – DATA ACCESS
DATA BASE

User Interface Testing

- Interface features include type fonts, the use of color, frames, images, borders, tables, and related interface features that are generated as webapp execution proceeds should be tested
- Individual interface mechanisms are tested in a manner that is analogous to unit testing(client-side scripting, dynamic HTML, scripts, streaming content).
- Each interface mechanism is tested within the context of a use case for a specific user category which is analogous to integration testing
- The complete interface is tested against selected use cases and NSUs to uncover errors in the semantics of the interface which is analogous to validation testing

Interface mechanisms

When a user interacts with a webapp, the interaction occurs through one or more interface mechanisms.

Links:

Each navigation link is tested to ensure that the proper content object or function is reached.

Forms:

- At a macroscopic level, tests are performed to ensure that Labels correctly identify fields within the form and that mandatory fields are identified visually for the user
- The server receives all information contained within the form and that no data are lost in the transmission between client and server
- Appropriate defaults are used when the user does not select from a pull-down menu or set of buttons
- Browser functions(e.g., back arrow) do not corrupt data entered in a form

Interface mechanisms

Client-side scripting:

1. Black-box tests are conducted to uncover any errors in processing as the script is executed.
2. These tests are often coupled with form testing, because script input is often derived from data provided as part of forums processing.
3. A compatibility test should be conducted to ensure that the scripting language that has been chosen will work properly in the environmental configurations that the webapp.

Dynamic HTML:

1. Tests should be conducted to ensure that dynamic HTML is displayed correctly.
2. Compatibility test should be conducted to ensure that dynamic HTML works properly in the environmental configurations that support the webapp

Interface mechanisms

Pop-up windows:

A series of tests ensure that

- The pop-up is properly sized and positioned
- The pop-up does not cover the original webapp window
- The aesthetic design of the pop-up is consistent with aesthetic design of the interface
- Scroll bars and other control mechanisms appended to the pop-up are properly located and function is required.

Interface mechanisms

Streaming content:

Tests should demonstrate that streaming data are up-to-date properly displayed, and can be suspended without error and restarted without difficulty.

Cookies:

<u>On the server side</u>, tests should ensure that a cookie is properly constructed and properly transmitted to the client side when specific content or functionality is requested and to ensure that its expiration date is correct.

<u>On the client side,</u> tests determine whether the webapp properly attaches existing cookies to a specific request sent to server

Compatibility tests:

- A series of compatibility validation tests are derived, often adapted from existing interface tests, navigation tests, performance tests, and security tests.
- The intent of the test is to uncovers the errors or execution problem that can be traced to configuration differences.

Component level testing

- Component-level testing, also called function testing, focuses on a set of tests that attempt to uncover errors in webapp functions. Each webapp function is a software component (implemented in one of a variety of programming or scripting languages) and can be tested using black box or sometimes white box techniques

- Black box techniques are equivalence partitioning, boundary value analysis

- White box techniques : path testing

- Forced error testing:

- It is used to derive test cases that purposely drive the webapp component into an error condition.
- The purpose is to uncover errors that occur during error handling (eg incorrect or nonexistent error message, webapp failure as consequence of the error, erroneous output driven by erroneous input, side effects that are related to component processing).

Navigation testing

- To ensure that the mechanisms that allow the webapp user to travel through the webapp are all functional and to validate the each NSU can be achieved by the appropriate user category
- Navigation mechanisms are tested to ensure that each performs its intended function.

Navigation mechanisms

- Navigation

  Internal, External links and anchors within a specific web page should be tested to ensure that proper content or functionality is reached when the link is chosen.

- Bookmarks

  Even it is a browser function, the webapp should be tested to ensure that a meaningful page title can be extracted as the bookmark is created.

- Redirects

  Redirects should be tested by requesting incorrect internal links or external URLs and assessing how the webapp handles these requests.

Navigation mechanisms

- Frames and framesets
  - Each frame contains the content of a specific web page, and a frameset contains multiple frames and enables the display of multiple web pages at the same time.
  - Test should ensure these functions works properly.
- 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(eg., the use of Boolean operators in the search field)
- Site Maps

  1. It provides a complete table of contents for all web pages.
  2. Each site map entry should be tested to ensure that the links take the user to the proper content or functionality

Configuration Testing

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.

Server-side issues:

- As server side configuration tests are designed, you should consider each component of the server configuration.
- Is the webapp fully compatible with the server OS?
- Are system files, directories, and related system data created correctly when the webapp is operational?

Configuration testing

CONTINUATION…

- 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?
- 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 effect on webapp operations?

Configuration testing

Client-side issues:

- On the client side, configuration test 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, Safari, IE, Opera, Chrome, and others
- User interface components- Active X, Java applets, and others
- Plug-ins- QuickTime, RealPlayer, and many others
- Connectivity – cable, DSL, regular modem, T1,WiFi

Security Testing

Security tests are 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.

<u>Client side vulnerabilities:</u>

- On the client side, vulnerabilities can often be traced to preexisting bugs in browsers, e-mail programs, or communication software.
- For eg : one of the commonly mentioned bugs is Buffer Overflow

Security Testing

Network vulnerabilities:

- Data communicated between the client and server are vulnerable to spoofing. Spoofing occurs when one end of the communication pathway is subverted by and entity with malicious intent.
- For e.g., A user can be spoofed by a malicious website that acts as if it is the legitimate webapp server. The intent is to steal passwords, proprietary information, or credit data.

Security Testing

Server side vulnerabilities:

Vulnerabilities include denial- of-service attacks and malicious scripts that can be passed along to the client side or used to disable server operation

For eg: server –side databases can be accessed without authorization(data theft).

Security Testing

To protect against these vulnerabilities, one or more of the following security elements is 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. Encryption is strengthened by using digital certificates that allow the client to verify the destination to which th 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)

Performance testing

It is used to uncover performance problems that can result from:

- Lack of server-side resources
- Inappropriate network bandwidth
- Inadequate database capabilities, faulty or weak operating system capabilities
- Poorly designed webapp functionality

<u>The intent is twofold</u>

- ❖ To understand how the system responds as loading (i.e. ., number of users, number of transactions, or overall data volume) increases
- ❖ To collect metrics that will lead to design modifications to improve performance

Load testing

- The intent of load testing is to determine how the webapp and its server-sie environment will respond to various loading conditions. As testing proceeds, permutations to the following variables define a set of test conditions:
- N, number of concurrent users
- T, number of online transactions per unit of time
- D, data load processed by the server per transaction.
- Load testing can also be used to assess recommended connection speeds for user of the webapp. Overall throughput, P, is computed in the following manner:
- P= N* T*D
- As an example, consider a popular sports news site. At a given moment, 20,000 concurrent users submit a request(a transaction, T) once every 2 minutes on average. Each transaction requires the webapp to download a new article that averages 3 bytes in length. Therefore, throughput can be calculated as

P=[20,000*0.5*3kb]/60=500kbytes/sec=4 megabits per second

Stress testing

- 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. The intent of test is to check:
- Does the system degrade ”gently”, or does the server shut down as capacity is exceeded?
- Does server software generate “server not available” message? More generally, are users aware that they cannot reach the server?
- Does the server queue resource requests and empty the queue once capacity demands diminish?
- Are transactions lost as capacity is exceeded?
- If the system does fail, how long will it take to come back online?
- 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?
- Are certain webapp functions discontinued as capacity reaches the 80 or 90 percent level?

MODEL-VIEW-CONTROLLER

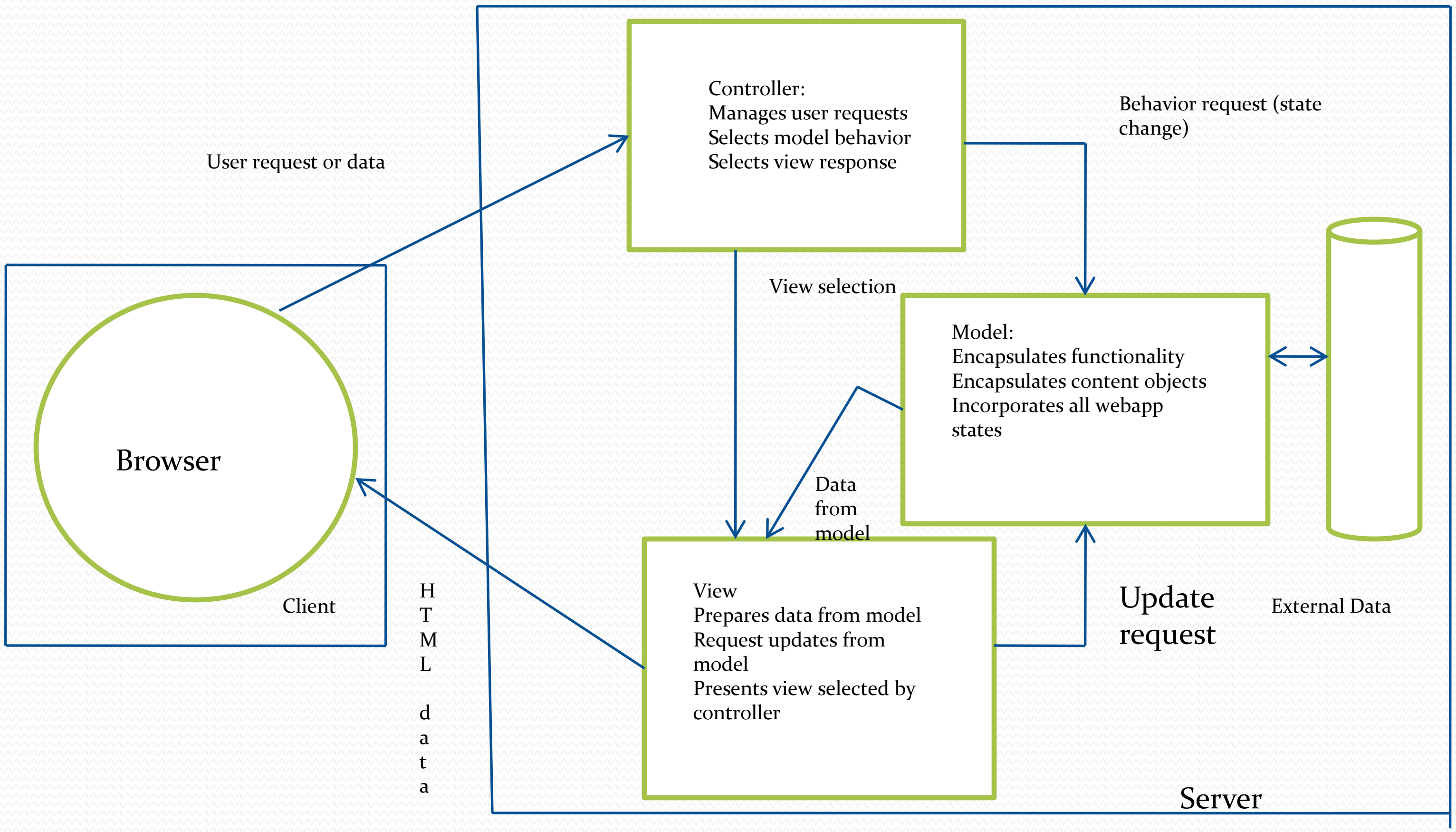
Webapp architecture

- WebApp architecture describes an infrastructure the enables a web-based system or application to achieve its business objectives
- Applications should be built using layers in which different concerns are taken into account.
- In particular, application data should be separated from the page's contents(navigation nodes) and these contents, in turn , should be clearly separated from the interface look-and–feel(pages)
- The Model-View-Controller(MVC) architecture is one of a number of suggested webapp infrastructure models that decouple the user interface from the webapp functionality and informational content.

MODEL-VIEW-CONTROLLER

- The model (sometimes referred to as the “model object”) contains all application-specific content and processing logic, including all content objects, access to external data/information sources, and all processing functionality that is application specific.
- The View contains all interface specific functions and enables the presentation of content and processing logic, including all content objects, access to external data/information sources, and all processing functionality required by the end user.
- The controller manages access to the model and the view and coordinates the flow of data between them.
- In webapp, “ the view is updated by the controller with data from the model based on user input”

The MVC Architecture



MODEL-VIEW-CONTROLLER

- Referring to the figure, user requests or data are handled by the controller. The controller also selects the view object that is applicable based on the user request. Once the type of request is determine, a behavior request is transmitted to the model which implements the functionality or retrieves the content required to accommodate the request.
- The model object can access data stored in a corporate database, as part of a local data store, or as a collection of independent files.
- The data developed by the model must be formatted and organized by the appropriate view object and then transmitted from the application server back to the client-based browser for display on the customer’s machine

ESTIMATION FOR SOFTWARE PROJECTS

Software cost and effort estimation will never be an exact science. To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project

   Unfortunately, it is not practical.

2. Base estimates on similar projects that have already been completed.

   The second option can reasonably well, if the current project is quite similar to past efforts and other project influences(e.g., The customer, business conditions, the software engineering environment, deadlines) are roughly equivalent. Unfortunately, past experience has not always been a good indicator of future results.

ESTIMATION FOR SOFTWARE PROJECTS

Decomposition Techniques:

- Decomposition techniques take a divide-and-conquer approach to software project estimation.
- By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.

Empirical estimation models

It can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right. A model is based on experience and takes the form

$$d = f(vi)$$

Where d is one of a number of estimated values(e.g., effort, cost, project duration) and vi are selected independent parameters(e.g., estimated LOC or FP).

ESTIMATION FOR SOFTWARE PROJECTS

Decomposition Approach can be of two different point of view

1. Decomposition of the problem.
2. Decomposition of the process.

Estimation uses one or both forms of partitioning

ESTIMATION FOR SOFTWARE PROJECTS

Software sizing:

- The accuracy of a software project estimate is predicated on a number of things:
- The degree to which you have properly estimated the size of the product to be built
- The ability to translate the size estimate into human effort, calendar time, and dollars
- The degree to which the project plan reflects the abilities of the software team
- The stability of product requirements and the environment that supports the software engineering effort

ESTIMATION FOR SOFTWARE PROJECTS

Putnam and Myers suggest four different approaches to the sizing problem:

- "Fuzzy logic" sizing : This approach uses the approximate reasoning techniques that are the cornerstone of fuzzy logic. To apply this approach, the planner must identify the type of application, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range.

ESTIMATION FOR SOFTWARE PROJECTS

Continuation…

- Function point sizing : The planner develops estimates of the information domain characteristics.
- Standard component sizing : Software is composed of a number of different "standard components" that are generic to a particular application area.
- Change sizing : This approach is used when a project encompasses the use of existing software that must be modified in some way

ESTIMATION FOR SOFTWARE PROJECTS

Problem-Based estimation:

- LOC and FP data are used in two ways during software project estimation:
- As estimation variables to “size” each element of the software
- As baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections
- Baseline productivity metrics(e.g., LOC/pm or FP/pm^6) are then applied to the appropriate estimation variable, and cost or effort for the function is derived.
- Function estimates are combined to produce an overall estimate for the entire project.

ESTIMATION FOR SOFTWARE PROJECTS

Process-Based Estimation

- The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.
- Like the problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of framework activities must be performed for each function.

ESTIMATION FOR SOFTWARE PROJECTS

Estimation with use cases:

- Use cases are described using many different formats and styles- there is no standard form
- Use cases represent an external view(the user's view) of the software and can therefore be written at many different levels of abstraction.
- Use cases do not address the complexity of the functions and features that are described
- Use cases can describe complex behavior(e.g., interactions) that involve many functions and features.

ESTIMATION FOR SOFTWARE PROJECTS

Empirical estimation models:

- An estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP. Values for LOC or FP are estimated , the resultant values for LOC or FP are plugged into the estimation model.
- The empirical data that support most estimation models are derived from a limited sample of projects. For this reason, no estimation model is appropriate for all classes of software and in all development environments. Therefore, you should use the results obtained from such models judiciously.

ESTIMATION FOR SOFTWARE PROJECTS

- An estimation model should be calibrated to reflect local conditions. The model should be tested by applying data collected from completed projects, plugging the data into the model, and then comparing actual to predicated results. If agreement is poor, the model must be tuned and retested before it can be used.

ESTIMATION FOR SOFTWARE PROJECTS

THE COCOMO II MODEL

- In his classic book on “software engineering economics”, Barry Boehm introduced a hierarchy of software estimation models bearing the name COCOMO, for constructive cost model.
- The original COCOMO model became one of the most widely used and discussed software cost estimation models in the industry.
- It has evolved into a more comprehensive estimation model, called COCOMO II, COCOMO II is actually a hierarchy of estimation models that address the following areas:

ESTIMATION FOR SOFTWARE PROJECTS

The Application Composition Model

- Suitable for projects built with modern GUI-builder tools. Based on new Object Points.

The Early Design Model

- You can use this model to get rough estimates of a project's cost and duration before you've determined it's entire architecture. It uses a small set of new Cost Drivers, and new estimating equations. Based on Unadjusted Function Points or KLOC.

The Post-Architecture Model

- This is the most detailed COCOMO II model. You'll use it after you've developed your project's overall architecture. It has new cost drivers, new line counting rules, and new equations.

ESTIMATION FOR SOFTWARE PROJECTS

Estimation for agile development:

- Estimation for agile projects uses a decomposition approach that encompasses the following steps:
- Each user scenario(the equivalent of a mini use case created at the very start of a project by end users or other stakeholders) is considered separately for estimation purposes.
- The scenario is decomposed into the set of software engineering tasks that will be required to develop it

ESTIMATION FOR SOFTWARE PROJECTS

Continuation…

- 3a. The effort required for each task is estimated separately. Note: Estimation can be based on historical data, an empirical model, or "experienced"
- 3b. alternatively, the "volume" of the scenario can be estimated in LOC, FP, or some other volume-oriented measure
- 4a. Estimates for each task are summed to create an estimate for the scenario.
- 4b. alternatively, the volume estimate for the scenario is translated into effort using historical data.
- The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment

ESTIMATION FOR SOFTWARE PROJECTS

Estimation for webapp projects

- Webapp projects often adopt the agile process model. A modified function point measure, coupled with the steps outlined, can be used to develop an estimate for the webapp. The following approach when adapting function points for webapp estimation.
- Inputs are each input screen or form (for example, CGI or java), each maintenance screen

ESTIMATION FOR SOFTWARE PROJECTS

Continuation…

- Outputs are each static web page, each dynamic web page script(for e.g., ASP, ISAPI, or other DHTML script), and each report(whether web based or administrative in nature).
- Tables are each logical table in the database plus, if you are using XML to store data in a file, each XML object(or collection of XML attributes)
- Interfaces retain their definition as logical files(for e.g., unique record formats) into our out-of-the-system boundaries
- Queries are each externally published or use a message-oriented interface. A typical example is DCOM or COM external references.
- For more information: software estimation techniques

Conclusion:

- Changes in software engineering technology are indeed “rapid and unforgiving” but at the same time progress is often quite slow.
- By the time a decision is made to adopt a new process, method, or tool
  - Conduct the training necessary to understand its application
  - Introduce the technology into the software development culture
  - Something newer has come along, and the process begins anew

Q & A