SOFT3103 – SOFTWARE VALIDATION AND TESTING

Course Summary

Use this notes at your own risk, I take no responsibility for the accuracy of information contained in these notes. If a mistake is found please let me know so I can correct it.

Introduction to Testing

Testing

- Testing is all about finding defects in a system
- Testing is primarily a defect detection mechanism – purpose is to find defects that are already in the system
- Testing must also be balanced with defect prevention within the development process – purpose to prevent defects being introduced in the first place

Definitions

- Error – a human action that produces an incorrect result
- Fault – A manifestation of an error in software. A fault if encountered may cause a failure
- Failure – Deviation of the software from its expected delivery or service
- Verification – The process of evaluating a system or component to determine whether the products of the given development phase satisfy the conditions imposed at the start of the phase
- Validation – Determination of the correctness of the product of software development with respect to the user needs and requirements

Exhaustive Testing is Impossible

- Can't test all inputs
- Can't test all timings
- Can't test all paths

Testing Strategy

Waterfall Model

- No way to fix bugs
- Emphasis the testing of the code and not the requirements

Requirements Defect: Come up with the wrong set of requirements. Use requirements to determine and trace risks in the testing

Static Analysis: Checking code up front, not done often but it reduces error and improves quality.

V-Model

- Every design phase has an equivalent testing phase
- Problem: not obvious how to feed back testing results into development

Spiral Model

- Development no longer sequential but form a cycle that keeps coming around
- Testing critical, results fed back into development cycle
- Develop systems incrementally

Extreme Programming

- Form of RAD
- Development cycle takes place over a single day
- Testing is central to this model and products can not be released unless all tests are passes
- Core Practices
    - On-site customer
    - Small releases
    - Planning
    - Metaphor
    - Simple design
    - Pair programming
    - Collective ownership
    - Continuous Integration
    - Coding standards
    - Testing – constant
    - Re factoring – changing code you have written
    - Forty hour week – never left along to work after hours
- Can get different testing frameworks for different languages
- Unit testing
    - Unit tests are developed prior to the actual coding of the unit
    - Debugging is undertaken by adding tests to the unit tests
- Acceptance Testing
    - Requirements integrated in each release should be acceptance tested in that release
    - Because customer is on site, this actual happens as the project is being developed
    - Often done with the customer to confirm they are getting what they want

Clean Room

- Where you mathematically prove the program is correct
- Formal methods
- Programmers are not allowed to compile and run your code
- Build code and formally verify it
- Return status on how reliable the system is
- Functional specification
    - Entire product specified in an appropriate mathematical formalism
    - Specification is subject to inspection to reduce errors
- Design and verification
    - Stepwise abstraction – prove implementation is what you specified
    - Correctness conditions
- Role of review and testing
    - Heavily dependent on inspection

        - Inspection is the primary means for the developers to verify correctness
    - Non execution based development
        - Developers not allowed to test or debug their programs
        - Testing process completely separated
        - Correctness achieved through
            - Stepwise abstraction
            - Code inspections
            - Group walkthroughs
            - Formal verification
    - Statistically based, independent testing
        - Testers simulate the operational environment with random testing
        - Process includes
            - Definition of frequency distribution of inputs to system
            - Definition of frequency distribution of system areas
            - Expanding range of developed system capabilities

Risk Identification

- Risk identification is the process of discovering and elaborating on system risks
- Must work out what risks are in the system and develop tests to check that those risks don't exist
- Work out qualities we want to test for and different test activities that can be performed to check those qualities

Quality Attributes

- Functionality
    - Suitability – Attributes of software that bear on the presence and appropriateness of a set of functions for specified tasks
    - Accuracy – Attributes of software that bear on the provision of right or agreed results and effect
    - Interoperability – Attributes of software that bear on its ability to interact with specified systems
    - Compliance – attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions
    - Security – Attributes of software that bear on its ability to prevent unauthorized access, whether accidental or deliberate to programs and data
- Reliability
    - Maturity – Attributes of software that bear on the frequency of failure by faults in the software
    - Fault Tolerance – Attribute of software that bear on its ability to maintain a specified level of performance in cases of software faults or infringements of its specified interface
    - Recoverability – Attribute software that bear on its capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort need for it
- Usability
    - Understandability – attributes of software that bear on the user's effort for recognizing the logical concept and its applicability

    - Learn ability - attributes of software that bear on the user's effort for learning its application
    - Operability – attributes of software that bear on the user's effort for operation and operation control
- Efficiency
    - Time behavior – attribute of software that bear on response and processing times and on throughput rates in performing its function
    - Resource behavior – attribute of software that bear on the amount of resources used and the duration of such use in performing its function
- Maintainability
    - Analyzability – attribute of software that bear on the effort needed for diagnosis of deficiencies of causes of failures or for identification of parts to be modified
    - Changeability – attribute of software that bear on the effort needed for modification, fault removal or for environmental change
    - Stability – attribute of software that bear on the risk of unexpected effect of modification
    - Testability – attribute of software that bear on the effort needed for validating the modified software
- Portability
    - Adaptability – attribute of software that bear on the opportunity for its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software considered
    - Install ability – attribute of software that bear on the effort needed to install the software in a specified environment
    - Conformance – attribute of software that make the software adhere to standards or conventions relating to portability
    - Replace ability – attributes of software that bear on the opportunity and effort of using it in the place of specified other software in the environment of that software.

Risk Identification

- Outcome of a risk analysis
- Testing is working out what your risks are, can't test until you know what your risks are
- Must do planning and design (design by identifying your risks)
- Technical Evaluation
    - Based on a technical understanding of the system and repeatedly questioning what could go wrong in different technical situations
    - Problem: reliant on your knowledge of the technicalities of the system, if poor people you will end up with lots of bugs as a poor analysis will be the result
    - Should evaluate the following areas
        - Vulnerabilities – weaknesses or possible failures in component
        - Threats – inputs or situations that might exploit a vulnerability and trigger failure in the component
        - Victims – who or what would be impact by failures and how severe they might be impacted.
- Quality Categories
    - A number of approaches use quality categories

        - Correctness
        - Reliability
        - Performance
    - Each category is evaluated separately to identify product risks
- Generic Risk Lists
    - Testers tend to work through the areas in which they think are important
    - Consider each generic risk separately and in detail
    - Allow risk analysis to be performed quicker
    - A convenient approach is to break down the system into main components and apply the list on each individual component
- Risk Catalogues
    - Contains a set of risks that are specific to a particular kind of software operation
    - Problem: if you can't find the right one for your system, you will still have to revert to something else as well

Risk Levels

- Decide what are the most important parts of the system (those parts which have the greatest impact if not operating correctly)
- Compare risks of system failures – the likelihood of incurring damage if a defect in the system causes the system to operate incorrectly or unexpectedly
- To priorities testing of the system based on risk
    - Break the system into smaller components
    - Determine the risk of each component
    - Those components with higher risk receive higher priority for testing, those of lower risk receive lower priority
- Testers view does not have to match the developers view
- Tests may be put together differently from how developers have arranged the code
- Good to use even number of values so you have to identify a high and low risk levels
- Must choose test activities which give the greatest coverage of your serious risks
- Can never cover everything so the strategy chosen will place limitations on what testing can be performed

Risk

- The chance of incurring a certain cost over a given period resulting from a failure
- Risk is a product of a failure's
    - Severity of impact
    - Likelihood of occurrence

Testing Strategy

- Concerned with what the tests are required to do
- It must consider
    - What risks are most critical to your system – where in the development process defects are introduced that contribute to these risks
    - Testing activities to detect these defects, and priorities according to risk
    - Preliminary estimates of staffing, environmental and test tool requirements for budgetary approval
- Outcome – is a document outlining risks, activities and staffing estimates budgeting
- Problems of testing without a strategy

  - Difficult in getting started with test project
  - Testing all bundled into one largely unstructured activity
  - Holes in the test approach that appear later
  - Lost opportunities to address quality in development
  - Staff and resources not available when needed

Risk Assessment

- Identified risks, rated risk, associating risk with some type of activity to mitigate it
- Identified risks are recorded in a risk register
- Activities are cross referenced as mitigations in the risk register – risks may have many activities used to mitigate the risk
- Test activities will never remove the risk
  - They will simply provide greater assurance that the risk is not present in the system
  - Affects the resulting likelihood of a risk
- Testing is aimed at minimizing the risks and mitigating as many as possible
- Sometimes need interim hooks in the code so testing strategies can be applied
- Test strategy must closely fit with other development, and verification and validation strategies – common that significant changes are made to accommodate verification and validation issues within the development process
- Test strategy is not to manage how a single testing activity is implemented
  - This is handled by the test plan
  - Testing strategy proposes how different testing activities can be planned to cover all important corporate and technical test requirements
- Quite common to develop a test strategy group for the project
- Test strategy does not state how to carry out your testing but rather say what your tests are
- Create a test plan for each activity and these are mapped to objectives

Preliminary Estimates

- Need to determine number of people doing testing and to determine the roles they are to play
- Environment needs to be determined

Steps Involved – Large Project

- Offline documentation
- Risk review meetings
- Activity proposal meetings
- Document review meetings
- Ongoing revisions

Static and Dynamic Testing

- Static Testing is to examine the component or system without executing the code, specification, design and user documents
  - Review meetings – eg. Code walkthroughs
  - Peer review
  - Running syntax and type checkers as part of the compilation process

- Dynamic testing is executing the doe to test, without necessarily examining the code itself.

Functional (Black Box) Testing

- Testing based on the requirements of the system or component under test
- Not possible to look inside the system
- Derived from functional requirement documents
- Test coverage achieved by building a test matrix, never know if you have followed all paths

Structural (White Box) Testing

- Testing based on the design and implementation structures of the system or component under test
- Test coverage achieved by recoding those paths of a program that have been executed under test
- Is possible to look at the code

Module (Unit) Testing

- Is applied at the level of each module of the system under test
- Is usually done by developers themselves before the module is handed over for integration with other modules
- Usually the testing at this level is structural
- Low level testing performed by developers as they develop the software

Integration Testing

- Tests the integration and consistency of an integrated subsystem
- A partial system level test can be conduced without waiting for all the components to be available
- Usually the testing at this level is a mixture of black and white box testing

System Testing

- Represents the testing after the completed system has been assembled
- Usually done by developers but not necessarily those that wrote the code
- Is performed after all module and integration testing has been successfully applied
- Done just before you show it to the customer

Acceptance Testing

- Demonstrates to the customer that predefined acceptance criteria have been met by the system
- Done by the customer to test that the system has what they expect

Performance Testing

- Is concerned with assessing the timing and memory usage aspects of the system's operation under different workloads or configurations
- Seeing if the system behaves in a timely manner

Load Testing

- Increases the system to peak levels of operation and checks whether this has any detrimental effect on operations
- Running with maximum number of concurrent users to determine whether these is any impact on network performance
- System works when we have a lot of people using the system at the same time

Stress Testing

- Is concerned with degrading the performance of the system to determine where this has any detrimental effect on operation
- Disconnection a network server to see if remaining servers can maintain the level of operation
- Check system can still survive when underlying system is not working at peak capacity

Usability Testing

- Considers human factors in the way that users interact with the system
- Checking that the system is usable

Reliability Testing

- Establishes how reliable the system will be when it is operating
- Done by performing statistical models and profiles on the operation levels of the system

Security Testing

- Make sure authorize people can only do what they are allowed to do
- Tests access control levels
- Asses vulnerabilities

Backup and Recovery

- Check that the system still works once it has been brought back online

Installation Testing

- Check that the system can be installed

Configuration Testing

- Check that the system will run on different hardware and software environments
- Test with each type of configuration and maximum and minimum configurations the system is likely to be used under

Compatibility/Conversion Testing

- Check that old data can be used with the new system
- This may involve
    - Check that the program operates as expected using the same set of data from the previous system
    - That integrity of data is maintained during the conversion process

Serviceability Testing

- Check that the system can be maintained

Alpha Testing

- Done inside the organization
- Represents the user of the system by actual users
- Testing is informal and represents typical use of the system

Beta Testing

- Same as alpha but the users are external to the testing organization

Parallel Testing

- Run new developed system with production system to see realistic data being tested against the system
- Results are compared between the system to discover discrepancies

Regression Testing

- Automated testing
- Applied after changes have been made to the system
- Operation of new version is compared to the old version to discover any discrepancies

Documentation Testing

- User documentation associated with a system is complete and consistent
- Checking of the documentation

Mutation Testing

- The testing of the testing process, deliberately introduce a bug to see if the testing process will detect it

Testing Infrastructure and Documentation

Test Environment Characteristics

- Testability
    - How easily can the software be tested
    - Must be designed into the system by the developers
    - Testability must consider both controllability and observables
- Controllability
    - Testing is limited by the facilities available to control the system to provide the inputs and invoke conditions
        - Put the system in the state where the test can be carried out
        - Return the system to a state in which other tests can be applied
- Observability
    - When test cases are run there must be some mechanism to check the results of the tests cases are correct
- Predictability
    - The expected output of the test case can be determined in advance of execution

- Repeatability
    - Test should return the same result each time they are executed – not the same value but rather pass or fail
    - Test environment needs to be controlled so that the complete environment can be rolled back to perform a test

Test Environment Independence

- Maintain independence between test environment and production environment for the following reasons
    - To prevent testing from corrupting the production environment
    - To prevent production influencing tests – expected results may change, production events may change test procedures
    - Testing requires significant reconfiguration – changes to the database and equipment
    - Avoid testing night shifts – more expensive, low morale, not enough time to restore to production environment
    - Try to avoid use of production data – data is always changing thus changing results, large data sets makes it hard to identify if error or not

Documentation

- Need some tool support to manage documentation. Tool to handle the process not to do the work
- Need to identify who has responsibility for each document produced
- Stakeholder co-ordinate activities via the test documentation
- No limit to how much you can create
- Need records of what has to be done and what has been accomplished
- Is your protection against liability

Goals of Development Project

- Development of tests and putting them in place
- Want it to support different people in different roles
- Co-ordinate test process through development and management of test documentation

Objectives of Test Documentation

- Define overall testing strategy for a project
- Plan how each level of testing is to be conducted
- Identify what tests are to be carried out
- Define how each test is to be performed
- Record execution of each test
- Track incidents that arise during testing
- Summaries outcomes of testing activities and make recommendations regarding quality and acceptability

Test Plan

- Introduction – summaries what is in the plan
- Test items – identify the components to be tested
- Features to be tested – those aspects of the system which will undergo test
- Features not to be tested – aspects of the system not to be tested

- Approach – general approach used to perform the testing
- Item Pass/Fail criteria – criteria to determine the success of each test
- Suspension Criteria and Resumption Criteria – identifies conditions which testing can be suspended and what must be done once it resumes
- Test deliverables – describe the documentation which describes the test activities to be performed
- Testing Tasks – identifies all the tasks required to complete testing, including any dependencies between tasks
- Environmental needs – what facilities are required for testing
- Responsibilities – who is responsible for what in the testing
- Schedule – when will testing be carried out
- Risks and Contingencies – identifies any high risk assumptions of the plan
- Approvals – sign off of the plan

Test Design Specifications

- Give all objectives of testing the feature
- Input/outputs and what is going to be tested
- Dependencies on other components within the system
- Includes
    - Test items
    - Input specifications
    - Output specifications
    - Environmental needs
    - Special procedural requirements
    - Inter case dependencies

Test Procedure Specification

- Description of procedures
- Many to one mapping between procedure and test case
- Procedure must contain enough detail so lowest skilled test person can perform it
- Includes
    - Purpose
    - Requirements
    - Procedure steps

Test Item Transmittal Report

- Request for testing
- Tell what you want tested
- How and when something is to be tested
- Stages of testing

Test Log

- What is returned after the test has been run
- Results of the tests, any abnormal events that could have affected the tests

Test Incident Reports

- How people identify what needs to be corrected
- Includes

  - Summary
  - Incident description
  - Impact on system

Test Process

- Describe the whole of testing, all activities, items produces, relationship between items, but not any procedures to be followed
- Describe what you need to understand when talking about testing
- Documentation will determine your test process

Tracking

- Want the documentation to be 3D, need the ability to trace through your documentation
- Requires tracking at fine grained support
  - Risk register
  - Features in test plan
  - Test objectives and test cases in test design
  - Test procedures
  - Test logs
  - Defects
- Reports can then produce summaries of different aspects
- Should keep track of how many times test have been run

Support Framework

- Requires tools to efficiently support
- Will have to be modified as you progress

Continuous Improvements

- Process improvements in on-going
- Future enhancements
  - Test outcomes to be reported at a test case level
  - Integration of time tracking
  - Enhanced management reports
  - Reports to support regression testing
  - Reports to analyze quality of products tested
  - Reports to analyze productivity
- Introduce improvements slowly, usually best to wait for the next project

Configuration Management

- Crucial to the success of a testing project
- Means for managing integrity and traceability throughout artifacts of software development and testing
- Problems
  - Inconsistent/inappropriate configurations released to customers
  - Reviewing/testing of an out of data component
  - Reviewing/testing the wrong component
  - Components changing mid way through review/testing
  - Inability to track faults in system configurations

    - Untested components used in subsequent development
- Planning Activities
    - Identification
        - identification of all components and their versions, baselines and configurations which represent the systems
        - Identify the different configurations which make up the product
        - Identify the attributes that can be assigned to components
        - Define how components can be stored in a repository
    - Control
        - Describes how changes to components of software development and testing activities are controlled
            - How to access components
            - When can a person access components
            - When components can be accessed
            - Assessment of the impact of proposed changes
    - status accounting
        - reporting functions for assessing state of development configurations
            - when development phases are complete
            - components tested
            - work pending
            - changes requested
    - audit and review
        - verifies whether
            - the configuration is complete
            - the configuration is a consistent set of parts
- Main Consideration for Testing
    - Define what components need to be placed under version and change management testing
    - Define criteria for development handling over components for testing
    - Define mechanisms for testing to be able to specifically identify versions of components or systems under test
    - Define what test documentation needs to be placed under version and change management conditions
- Problems Change Management Tackles
    - Is used to reduce problems associated with developing software systems
        - Inconsistent/inappropriate configurations released to customers
        - Reviewing/testing of an out of data component
        - Reviewing/testing the wrong component
        - Components changing mid-way through review/testing
        - Inability to track faults in system configurations

Incident Management

- A process for handling, identifying and resolving defects in the system
- How are we going to manage incidents, who is responsible for them
- Purpose
    - Mechanism for identifying and recording potential defects in systems under test
    - Ensuring that problems and defects are addressed appropriately and in a timely manner

    - Provide mechanisms to monitor defects and discover areas in which improvements could be made for a better quality product
    - Use it as a method for monitoring the development process thus reducing bugs
- What is an incident
    - See strange this, things not expected
    - Often called bugs, defects, anomalies
    - Any event or unexpected behavior of a system under test that requires further investigation
- Someone must always review and make a decision on what to do about it
- Need to have a document to record and manage incidents
- Situations must be reproducible, need to be able to isolate bugs to determine reason, need to know what was done to get the bug
- Have process for handling incidents in each stage of development
- Assessment
    - Each incident raised is assessed
        - To determine whether it is to be considered a defect that requires resolution
        - To determine the impact of expected behavior
- Resolution – the system is modified to remove the defect and resolve the incident
- Validation – retest after modification, make sure there are no side affects of the fix
- Management – overall generation and assessment of reports summarizing collective defect data. Purpose is
    - Ensuring all defects are addressed as soon as practicable
    - Estimating the progress of defect resolution
    - Estimating effectiveness of the testing process
    - Estimating the reliability of aspects of the system under test
- Analyze
    - Try and identify phase of development where bugs are being injected into the system
    - Also provide feedback on the testing process
- Incidents raised
    - When testing starts find lots of bugs because system is immature
    - Find fewer bugs as issues are resolved
    - If little defects found early – result of poorly planned development
    - Late increase
        - May find more because starting new forms of testing and they raise new forms of incidents
        - Sometimes do not raise minor issues early and raised later

Test Design (Black Box Techniques)

Purpose of Black Box Testing

- Any testing where you do not base it on any development code
- Proposing tests without detailed knowledge of the internal structure of the system or component under test
- Based on the specification of what the system's requirements
- Deployed once there are enough units together to build a portion of the system. Once a fair portion of the system has been constructed.

Types of Errors Detected

- Incorrect or missing functions
- Interface errors – talking about the interface between different modules and components
- Initialization and termination errors
- Information and data flow errors

Domain Driven Testing

- Divides large range of possible tests into subsets, and select best representative from each set
    - Testing ranges of possible inputs and outputs
    - Compatibility testing
- Pros
    - Easy to implement
    - Generalizes well
    - High yield for small set of tests
- Cons
    - Domains are difficult to determine
    - Errors that are not boundaries or obvious special cases may be missed

Stress Driven Testing

- Work out the limits of the software
- Explores the capabilities of the software by pushing it beyond its limits
    - High volumes of traffic
    - Number of clients and connections
    - Length of transaction chains
- Pros
    - Expose weaknesses that occur in the field
- Cons
    - Many problems are unrelated to stress

Specification Driven Testing

- Looking at the documented functions of the system. Make sure these are covered in our test cases
- Checks conformance of product to each assertion made in requirements and specification
- Pros
    - Critical defense against warranty claims etc.
- Cons
    - Missing and wrong requirements may be missed

Risk Driven Testing

- Risk analysis on each individual feature
- Thinking in terms of business manager what risk exposure happen if this part does not work
- Prioritizes testing effort in terms of the relative risk of different areas or issues that can be tested
    - Frequency of use testing

    - Stress test, error handling, security tests, test locking for predicted or feared errors
- Pros
    - Optimal prioritization of tester effort
- Cons
    - Missing risks may be missed
    - Wrong priority may force test to be overlooked

Random/Statistical Testing

- Run a large variety of tests against the system
- Consult oracle to determine if observed behavior is valid or not
- Thinking in terms of a scientist
- Have the computer randomly select, execute and evaluate a large number of tests automatically
- Pros
    - Regression tests won't rely on the exact tests being rerun each time to have higher likelihood of test coverage – can choose different data each time, providing broader coverage
    - Partial oracles find errors in code quickly
    - Detect failures as a result of long, complex chains of actions that would be hard to run as planned tests – generate for complicated test cases easier
- Cons
    - Test cases are only as good as the Oracle, does not cover everything you want to test
    - Oracles need to be able to detect pass from failure
    - May not be able to create an oracle thus results in a thin slice of coverage

Function Testing

- Looking at all the actual functions of the system, need to be able to determine functions. May have to take to expert uses and programmers
- Programmers test
- Test each function thoroughly one at a time
- Pros
    - Thorough analysis of each item tested
- Cons
    - Misses interactions, misses exploration of the benefits offered by the program
    - May overlook interaction between different functions

Regression Testing

- Manage the risk that a bug fix didn't fix the bug or the fix or other change had a side effect
- Pros
    - Reassuring confidence building and regulation friendly
- Cons
    - Misses anything not covered in the regression test series
    - Maintenance can be costly
    - If change one feature you may be required to go back and change all your regression tests

Scenario Testing

- Good for user acceptance testing, average person hat, not user hat
- Pros
  - Complex, realistic events
  - Exposes failures that occur over time
  - Useful for acceptance testing
- Cons
  - Single function failures can make this test inefficient
  - Must think carefully to achieve good coverage

User Testing

- Need to think like a user
- You will find errors which will pop up during deployment
- Identify cases that will arise at the hands of a real user
- Pros
  - Design issues more credibly exposed
  - In house tests can be captured with recorders
  - In house tests can focus on issues that could be controversial
- Cons
  - Not good at exercising all features of the system
  - Very expensive exercise (bring users in or support system if take it out)
  - Coverage is not assured
  - Test cases can be poorly designed, trivial, unlikely to detect

Exploratory Testing

- Software comes to the tester undocumented. Tester must simultaneously learn about the product and about test cases/ strategies that will reveal the product and its defects
- Used when you have very little documentation. Do not no where it came from or where to start, done in early stages of development
- Test is wearing their hacker hat
- Pros
  - Thoughtful strategy for obtaining results in the dark
  - Strategy for discovering failure to meet customer expectations
  - Less resources required to carry it out
- Cons
  - The less that we know the more we are at risk

Structured Test Design

- Need a method to develop test cases, assists in guarantee of coverage
- Break system down into smaller functions
- At the start of the testing project, testers have to decide what test cases to execute to thoroughly test the system
- Key is to develop a systematic approach
  - Break the system down into smaller manageable activities
  - Apply a systematic approach to each component
- Designing what test cases are required is a labor intensive activity

- Need some form of documentation, document all functions of the system, if don't can't determine what the system does and therefore can't judge the system if they pass or fail
- Essential that knowledge of end users of the system are included in test design
- At least a simple form of specification should be created from which the top level set of test objectives can be derived
- A significant part of the test design technique involves formulating a specification to test against

Decomposing Test Objectives

- Test plan will list all the features that need to be tested. Need to write objectives to be tested, need to give objectives for each case. Need to be able to verify we have tested those features adequately.
- Test design will focus on a set of application components to test
- High level test objectives are proposed for these specification components
- Each test objective is then systematically decomposed into either other test objectives or test cases using test design techniques
- Test design techniques are applied once you have decomposed your test objectives to single components for a given test criteria
- Documenting your test designs is crucial
    - Test designs are normally represented in a document called a test design specification
    - Provides an audit trail that traces the design from specification components through to applicable test cases
    - Records the transformation of test objectives and justifies that each test objective has been adequately decomposed at the subsequent level
    - Often required to justify to other parties that each requirements has been fulfilled.
    - Looks at very detailed procedures to ensure we have coverage
- Techniques are used to workout how to create test cases and how they relate back to tools

Functional Analysis

- Analysis of each function to determine tests that verify the function is operating as expected
- Need to identify the functions that comprise the system
    - Build a hierarchy or tree of functions
    - Branches are successive functional areas
    - Leaves are separate functions
- Good starting point is looking at the menus and user manual
- Analyze each function separately to define a set of test criteria, analyze each function's input and output processing to determine what tests are required
- Focuses on
    - Criteria
        - Talking about the functions of the function. Allows us to asses whether the test has passed or failed
        - Define criteria that determine whether the function is performing correctly
    - Outputs

        - Identify those outputs that must be produced by the system in order to support the function
        - Identify how values are captured
        - Identify how values are determined to be correct
    - Inputs
        - Identify those inputs required to generate the outputs for each function
        - Differs from outputs because they must check how the system handles invalid inputs
    - Internal Conditions
        - Identify internal conditions under which outputs are expected to be produced
        - Identify what happens when required conditions are not satisfied
        - Must check that it does not do any processes it should not
    - Internal state
        - Looking at static data structures eg. Files
        - Identify how the internal state changes after receiving each input
        - How can the internal state be observed externally to check for correct change to state

Use Cases

- A use case is a sequence of actions performed by a system, which together produce results required by users of the system
- It defines process flows through a system based on its likely use
- Deals with working out how people interact with the system
- Tests derived from use cases help uncover defects in process flows during actual use of the system – may discover that it is not possible to transition to a different part of the system when using the system as defined by the use case
- Use cases also involve interaction or different features and functions of the system – helps uncover integration errors
- Deriving Test Cases
    - Consists of choosing paths that traverse through the use case – a single path can give rise to many use cases
- Negative test Cases
    - Consider the handling of invalid data, as well as scenarios in which the precondition has not been satisfied
    - Three kinds
        - Alternative branch
        - Attempting inputs not listed in use case
        - Attempting paths not listed in the test case
- Deriving test cases will inevitably discover errors in the use cases as well
- Use cases and test cases work to together well in two ways
    - If the use case for a system are complete, accurate and clear, the process deriving test cases is straight forward
    - If the use cases are not in good shape, deriving test cases will help to debug the use cases

Equivalence Partitioning

- Assumption: as long as we have selected a value from every class we have tested everything

- Inputs and outputs of a component can be partitioned into classes which are treated similarly by the component
    - Similar inputs will evoke similar responses
    - A single value in an equivalence partition is assumed to be representative of all other values in the partition
    - Assume if a test passes with the representative value, it should pass with all other values in the same partition
- Classes are values that perform the same operation and produce the same result
- Design test case to test each class
- Separate test cases are generated for each class
- A test case comprises the following
    - The inputs to the component
    - The partitions exercised
    - The expected outcome of the test case
- Two approaches for developing test cases
    - Separate test cases are generated for each partition on a one-to-one basis
    - A minimal set of test cases is generated to cover all partitions

Boundary Value Analysis

- Extends equivalence partition to include values near the boundaries of classes. Extreme value testing
- We assume that sets of values are treated similarly by components
- Developers are prone to making errors in the treatment of values on the boundaries of these partitions
- Usually limits of the equivalence classes

Cause Effect Graphing

- Uses a model of the behavior of a component based on the logical relationships between causes and effects
- Each cause is expressed as a Boolean condition over inputs and each effect is expressed as a Boolean expression representing an outcome or combination of outcomes
- The model is typically represented as a Boolean graph relating cause and effect expression using Boolean operators AND, OR, NAND, NOR and NOT
- Test cases are produced to exercise each unique possible combination of inputs to the components

State Transition Testing

- Uses a model of the system comprising
    - The states the program may occupy
    - The transition between those states
    - The events which cause those transitions
    - The actions which may result
- Model typically represented as a state transition diagram
- Test cases are designed to exercise the valid transition between states
- Additional test cases may also be design to test that unspecified transitions cannot be induced
- Valid Transition tests

    - For each test case the following is specified
        - The starting state
        - The input(s)
        - Expected outputs
        - Expected final state

Syntax Testing

- Validate that the system is processing the units of language input correctly
- Problem with getting coverage of all possible inputs
- Use a model of formally defined syntax of the inputs
- Check that only proper formats of data are accepted and all improper formats are rejected
- Performing
    - Design test cases with valid and invalid syntax
    - Valid test cases should execute
        - For selection, derive a test case for each alternative
        - For iterations derive a test case for the minimum number of iterations and more than the minimum iterations
    - Test cases are defined with
        - Inputs
        - Syntax expected
        - Expected outcomes
    - Deriving invalid test cases
        - State with the uppermost level in the tree
        - Introduce one error at a time on the selected level
        - Continue for all errors at that level
        - Move to next level until entire tree has been traversed

Test Automation 1

What is automated testing

- Set of tools to help test our system
- Maximize the utilization of resources saving money and reducing risks
- Tools used to streamline the process and documentation
- Tools used to streamline the execution of tests
- Tools used to streamline the gathering of metrics
- Offers significant improvements in productivity
- Always not appropriate and may be expensive
- Tools very expensive

Types of Testing Tools

- Test Management Tools
    - Generate test cases automatically for you. Will look at combination of arguments and specification
    - Document oriented
    - Help coordinate, organize, plan, design and document testing
- Test Design Tools
    - Help to decide what to test

- Test drivers and Capture/Playback tools
  - Record tests and replay them to compare results
  - Generate scripts which can be played back
- Load and performance tools
  - Simulate a heavy load on the system, popular for web testing
- Test data generators
  - Generate data to pump into your system, testing what is going in
- Web Site link testing
  - Testing of website
- Test Evaluation tools
  - Tools that will figure paths through code and test the coverage of the code
- Static Analysis Tools
  - Things that the compiler would do
  - Type checking on strong language
  - Checking of non strongly types language
  - Look for data areas, non initialized variables, out of bounds error
  - Check that every part of the code is reachable
  - Check pointers

Capture Playback Tools

- Automation allows us to record test cases, use scripts to compare results, run them many times, record outputs and gain statistics from the outcomes recorded
- Easiest type to test are batch systems as there is no user interaction
- Test comparison part is complex, need to develop scripts so they are easily maintained. Have same issues as other systems/programs have eg. Maintenance
- Steps
  - Capture the activities carried out by the user
  - Need to identify the things which we want to test for
  - Determine how we are going to perform checking
    - What will happen if a failure occurs
    - Do other tests depend on this one
    - What post execution tests are required
    - What manual tests are required
  - Validation is not always automated as very complex
  - Can execute the captured inputs over and over again
  - Check the outputs returned after each execution with those against what we expect
  - Need to ensure all programs start in a consistent state

Data Driven Testing

- Separate the data from the code, put data in a spreadsheet and it will put it in appropriate places and execute them for you
- Hard to maintain, code more complex. Programmer needs to develop the system

Automating the Process

- Making long term investment when automating, good for when there will be multiple releases of the program
- What to automate
  - Start with simpler tasks first

    - More complex cases as skills build
    - Some tests will remain manual
    - Always be practical – do a feasibility study, don't integrate into practical components
- Look at building two sets of tests
    - Sanity tests
        - Check nothing has gone wrong
        - Run before the full testing is started to determine if the system is ready to go through full testing
        - Runs quickly and ensures no unexpected surprises
    - Full testing
        - Complete set of tests
- Make scripts as general as possible
- Can have errors in your test cases so checking needs to be built in
- Not good at handling changes to the interface/data overall environment. Need a well known position for the tests to run from
- If testing is in the normal product system you have no control over the system state. Need a separate system. Need to know why the failure is being caused by
- Long term investment
- Performance and load testing is done using automation as it is very difficult to perform manually
- Improvements in productivity are gained through
    - Duplication
        - A test can be duplicated/parameterized for different cases
        - Useful for load and stress testing to duplicate large quantity of tests
    - Repetition
        - The same test is repeated many times – where the benefit of automation is gained
        - Useful for regression testing to ensure modifications have no adversely affected the system

Advantages of Automation

- Enabling more thorough testing
    - Improved regression testing
    - Time to develop new/improved tests
- Shorten duration of testing
    - Improve time to market
    - Quicker test turn around prior to release
- Improve productivity of scarce resources
- Improves morale
- Reduces tester error and oversights
- Unattended testing supported
- Provides a detailed test log and audit trail
- Records test execution precisely for debugging process
- Incremental process supported
- Test scripts are an asset

Disadvantages of Automation

- Time and effort to setup tests initially

- Maintenance of test cases
- Learning curve
- Doesn't eliminate manual testing
- Easy to focus on tests that are simple to automate
- Investment code
- Technical limitations
- Ease of use
- Lack of stability and support

Adoption Process

- Steps towards automation
  1. assess needs and get high level management commitment
  2. contact vendors and assess high level capability of tools
  3. assess costs and budget commitment
  4. evaluate products for own environment
  5. contact reference sites
  6. negotiate investment
  7. initial purchase and pilot
  8. broader use and additional investment

Module/Unit Testing

- Testing against a low level specification
- Not used to find syntax errors but rather logical errors in the code
- Testing done on the code
- Done by developers when writing code
- Very unstructured
- Was up to developers to determine the quality of the code
- Check every statement, ever condition, every combination of conditions not possible. Need to execute tests that give the widest coverage
- Not just test the condition but how we got to the condition in the first place
- Statement coverage – check each statement in the code is executed once, do not look at conditions or branches
- Branch Converge – look at what logic takes us down a particular path, check that each condition is checked
- Path coverage – check that every path is executed. Number of paths become large, how many paths before and after code needs to be taken into consideration
- Can have statement coverage without branch converge
- Can have statement and branch converge without path coverage
- 100% branch coverage gives 100% statement coverage
- 100% path coverage gives 100% branch coverage
- Look at all possible paths in the code. Draw flow graphs. May do this for particular parts of the code not the entire system.
- Complete statement and branch coverage – path coverage is not possible when while loop. Should think of what paths are possible and what ones you want to test
- Look at data segments to determine paths and conditions
- Keep path coverage simple if manual testing or leave it up to a system
- Need to understand the code and logic

- As path gets longer it gets harder small paths, string lots of little tests together to reduce complexity
- Need to have a technical background, do lots of little tests as code developed to make things easier
- These can give you functional and code testing because determine inputs and outputs
- Loop details are skipped over
- Difficult to determine what paths are feasible and test case values to satisfy paths
    - Requires solid understanding of program logic
    - Easier to have a number of smaller paths
    - Typical to first apply black box testing, measure coverage, and then use white box testing to gain further coverage

Coverage Test Design

- Analyze source code to derive flow graph
- Propose coverage test paths on flow graph
- Evaluate source code conditions to achieve each path
- Propose input and output values based on conditions
- Look at all possible paths in the code. Draw flow graphs, may do this for particular path of the code and not the entire system

Loop Testing

- Is impossible when start talking about while loops and things out of your control
- Do basics paths – checking different segments. Number of executions of path. Inductive process, assume everything else works when proved correct
- Zero Path – represents short circuit of the loop
- One Path – represent a loop in which only one iteration is preformed
- Basic paths represents atomic components of all paths – possible paths are combinations and sequence of basis paths

Basis Path Testing

- Each basis path is exercised by at least one test – both zero-path and one path basis paths
- Combinations and sequences of basis paths do not need to be exercised by a test – they are covered by combinations and sequences of already tested basis paths
- Reduce number of test cases and improve the amount of coverage weight
- Does not check when termination is met, combination of conditions checked by individual segments but not the termination conditions

Beizer's Loop Tests

- Bypass: any value that causes loop to be exited immediately
- Once: values that cause the loop to be executed exactly once
- Twice: values that cause the loop to be executed exactly twice
- Typical: a typical number of iterations
- Max: the maximum number of allowed iterations
- Max + 1: one more than the maximum allowed iterations
- Max – 1: one less than the maximum allowed iterations
- Min: the minimum number of iterations

- Min + 1: one more than the minimum number of iterations
- Min – 1: one less than the minimum required
- Null: one with a null or empty value for number of iterations
- Negative: one with a negative value for number of iterations

Branch Condition Testing

- Need to know how each element in the condition gets there value, as well as checking the execution of the statement
- Identifies individual Boolean operations within decisions
- Test exercise individual and combinations of Boolean operand values

Modified Condition Decision Coverage

- Way of reducing the number of tests but still covering all the statement, by using smart analysis in your test design
- Show that each Boolean operand can independently affect the decision outcome.

Data Flow Testing

- Looks at path coverage with a focus on data as well
- Look at what data is needed to go through a particular path
- Interactions between parts of a component
- Structural model aims to execute sub paths from where variables are defined to where they are used
- Identifies variable occurrences – definitions and use of variables
- Test cases design to cover control of flow paths
    - Inputs
    - Location of variable definitions and use pairs
    - Control flow sub path(s) executed
    - Expected outcomes

LCSAJ Testing

- Very high maintenance testing. No flow code, look at linear code
- Look at the start and end points to determine the way the code goes. Need to identify the linear sequences
- Analysis is very complex, change in the code will change the analysis completely
- Coverage more effective than other techniques
- Done on real time software and once software is complete

White Box Techniques

- Can lead to an infeasible number of test cases
- Can be used to give good coverage
- Useful for analyzing particular paths in programs to exercise them with tests
- Tends to be more objective if another developer is looking at it as well
- Techniques to generate test cases
- Advantages
    - Easier to identify decisions, and equivalence classes that are being used in the logic than with black box testing
    - More objective than black box testing, different people deriving test cases will arrive at corresponding results

    - Automated tools available to propose paths to test and to monitor coverage as the code is tested
- Disadvantages
    - Will not find wrong implementation
    - Does not find errors in design
    - Need to be a technical person
    - May find parts that are never reached
    - Does not look at interface issues
    - Incorrect code gives rise to tests that do not conform to expected operation as well
    - Need to good understanding of the code to understand the meaning of each test path through the code
    - Some paths are difficult to visit
    - Need access to the code
    - If bad code you can't do testing but you identified the bad code so testing has worked
    - Multi thread systems impossible
    - Only practical for small components on code. Developer focus
    - Impossible to test spaghetti code as there are excessive number of paths
    - Problems with multiple processes, thread, interrupts, event processing
    - Dynamically changing data and code is difficult to analyze
    - Only practical at low levels, for whole systems the number of paths through a whole program is incredibly large.

Building JUnit Tests

- Define a subclass of TestCase
- Override the setup() method to initialize objects under test
- Override the teardown() method to release objects under test
- Define one or more testXXX() methods to exercise objects under test
- Define a suite() factory method that creates all the testXXX() methods or the testCase
- Define a main() method to run the TestCase

Unit Test Organization

- Should build the tests as you are coding. Once infrastructure there, just another task. Create a package, you would normally be doing something similar
- Create the tests in the same package as the code under test
- For each java package in the application, define a TestSuite class that contains all the tests for verifying the class
- Make sure the build process includes the compilation of all test suites and test cases

Unit Test Principles

- Code a little, test a little, code a little …
- Run tests as often as possible
- Run all the tests in the system at least once a day
- Begin by writing tests for the area of code that you are most worried about breaking
- Write test that have the highest possible return on your testing
- Add test before adding new functionality
- Don't deliver code that doesn't pass all the tests

Test Harness

- Supporting code and data used to provide an environment for testing components of your system
    - Typically used during unit/module test
    - Typically created by the developer of the code

Test Stubs

- Dummy components that stand in for unfinished components during unit/module testing and integration testing
- Quite often code routines that have no or limited internal processing
    - May always return a specific value
    - Request value from user
    - Read value from test data file

Test Oracles

- Another system that will be produced with the appropriate data required for input to a system
- Will have problems all of their own
- Need a test process on the Oracle
- Very complex
- Question of did the problem come from the Oracle or the system under test
- Facilities provided to calculate the expected output for a test case or to check that the actual output for the test case is correct
- If different results are obtained for a test case, must first determine whether the system under test is incorrect or whether the oracle is incorrect
- Are commonly the tester calculating or checking the result by hand
- If the test case involves more complex calculations then programs may be written to compute or check the output

Heuristic Oracle

- Tester analyses the inputs and results of the sine function to determine if there are predictable relationships between them
    - Provide a number of test cases using the sampling oracle approach
    - The remainder of the tests would be based on consistency of inputs and results with the predicted relationships

Test Automation 2

- When thinking how to test, you need to think about your architecture
- Architecture of product and different ways of testing the product
- Choosing different testing levels
- Problems of testing before done – not complete, design tests at a lower level, dependencies in the code, people don't want to because they will have to do it again
- If leave it to the end to test – hard to find where the error is, to many variables, to complex, may have a design fault (re-engineering),
- It is good to do things in parallel to get feedback as you go

Integration Testing

- Allows partial system test without waiting for all the components to be available
  - Finds bugs earlier
  - Problems can be fixed before affecting downstream development on other components
- Sometimes performed as some tests cannot be carried out effectively on a fully integrated system – helps to focus on a particular area and isolate problems
- Different ways of building them, must match the way the application is being designed and built
- Bottom Up – start from lowest level, assembles up from the lowest level components and replaces higher level components with test harness to drive the unit tests
- Top Down – start from interface, assembles down from the highest level components replacing lower level components with test stubs to simulate interfaces
- Inside-out – pick out components surround by stubs and harnesses to test it. Just keep integrating and keep adding and evolving the tests and application

Action Words

- A scenario you want to test, see keyword and call the correct routine
- Benefit: don't have to spend all the time building low level code, have tester populating the code, can use the functions of a spreadsheet to help build the test
- Can get non technical people to populate tests
- Need to consider incrementing and maintaining change between executions
- Need to have scripts that provide a mapping between logical and physical names
- Using Perl
  - Test scripts are defined in the test language and are executed by an interpreter
  - The interpreter communicate directly with the server application using the same protocol as the client
  - The client is removed from the scope of automated testing
  - Identify transactions that the server supports and check actions. Make an action word for each one
  - Testers create scripts by sequencing various transactions and checks together in a script

Model Based Test Automations

- Don't define test data, must define states of your system (what states the system goes through)
- State transition models are critical
- Takes the model, and generate test data from the model automatically rather than generating the data manually
- Automatically derive tests from the specification
- Tests automatically executed by the test tool
- To change the tests you would adapt the model
- Must think of when you want to stop the tests, and how navigation happens, also need to see algorithm to get maximum coverage

Alternatives for selecting Test cases to retest

- The entire test set
- Specifically for the change

- Entire component containing the change
- Entire components related to the change
- Pre-planned regression test suite such as the acceptance tests
- Random selection of tests
- Tests that previously revealed bugs in the changed component
- Tests associated with the highest risks
- Selected kinds of tests
- Test for bugs reported by customer
- Tests that haven't been executed for some time
- Complex tests that exercise a number of aspects.

Performance, Load and Stress Testing

Main Causes of Poor Performance

- Increases number of architectural layers and suppliers – many more layers in multi-tiered systems
- Distributed processes – many network request, each of which may have delays of seconds
- Intelligent clients
    - Clients might process a large amount of data passed over the network
    - Many agents may be started to communicate with the server

Measuring Performance

- Need tools to measure response from the system
- Transaction throughput – how many transactions can be processes per time unit
- Transaction latency – how long before system responds to each transaction
- Round trip time – represents a time to cycle a complete test scenario for a single user

Load Testing Planning

- The volume and characteristics of the anticipated traffic are simulated as realistically as possible
- Commonly design and develop load tests that don't even come close to matching real load
- Difficult to simulate realistic load without automation

Anticipate Load

- Time duration of sessions
- Length of sessions
- What components were executed
- Additional Statistical information – how many session per month, peak numbers, anticipated growth
- Understand the wide range of user actions
- Understand load levels of user actions

Target Load Levels

- How the overall amount of traffic/users is expected to grow
- The peak load levels which might occur

- How quickly normal load might ramp-up to peak load
- How long the peak load are sustained
- Consider special events/promotions/product launch where peak levels are unusually high

Test Case Design

- Load test objective – able to handle n sessions per hour
- Pass/fail criteria – max defined response time
- Script description – should have the number of components to be executed

Bottlenecks

- CPU load – no idle time means it is a bottleneck
- Memory use – amount of free memory before caching
- Swapping rate – memory caching to disk slow performance
- Thread pool size – number of processes supported
- Network capacity and latency – bandwidth capacity and delays
- DBMS load – how heavy is the database engine handling transactions
- Blocks read/written per second – how much disk accessing is taking place

Stability Testing

- Uptime and availability – is the application free of memory leaks, how long can the system remain up for
- Need to simulate accelerated level of use over a sustained period

Non Functional Testing

Content Checking

- Check pages for
    - Accuracy
    - Completeness
    - Consistency
    - Spelling
    - Accessibility

HTML Testing

- Invalid HTML is usually ignored by the browser
- Automated tools can validate syntax conforms to HTML standard
- Web page checkers can go further
    - Spell check
    - Link check
    - HTML syntax
    - HTML browser compatibility
    - Load Times

Browser Compatibility

- Microsoft and Netscape adopt different strategies – most incompatibilities caused by non-standard HTML
- Page authoring tools have support for specific browser conformance

- Visual inspection recommended

Browser Differences

- Colours of standard objects eg. Background, lines, borders
- Centering and scaling of objects in windows
- Table layouts – automatic sizing gives different results

Production Risks

- Performance prediction – does the system sound alarms when activity approaches performance degradation points identified by performance/load testing
- Utilization – can the system servers be monitored to predict upgrade requirements
- Availability – can limitations of access by different kinds of user be monitored and failure detected
- Production tests – can functional tests be run on the production environment to corroborate test environment results

Production Failures

- Productivity loses due to slow performance
- Loss of customers who lose patience with slow site performance
- Temporary inability to access the site. Most will try back later, many won't
- Customers can't place orders once they're on the site
- Computer crashes, customers data is damaged, orders are lost
- Extended downtime, disruption of processes

Monitoring Issues

- May want to schedule tests from different locations
- Notification of failure – may set thresholds that trigger notification support
- Kinds of failure – server offline, poor server response

See Book for Translatable Components and Cultural Data Issues – Too long to type out

Security Testing see Top 10 Article

Usability Testing

What is Usability?

- The capability of the software product to be understood, learned and liked by the user, when used under specified conditions
- Goal is to create products which are
    - Easy to learn
    - Satisfying to use
    - Highly valued among the target population

Design for Usability

- Test with real users early in product development and frequently thereafter

- Usability requirements should be part of initial product requirements definition
- Key stakeholders should be involved in the design process early on
- Ongoing assessment of usability status ensures timely feedback

Common Usability Principles

- Provide feedback to the user
- Always show system status
- Provide an Escape back to the previous state
- Minimal work
- Provide sensible defaults
- Provide help from every state
- Allow users to undo at any time
- Be consistent

User Centred Design Techniques

- Participatory design
- Focus group research – get group of potential users
- Surveys – find out from target audience
- Design walkthroughs
- Paper and pencil evaluations – draw on page screen designs
- Expert evaluations – get expert users to do tests
- Beta testing – install in target companies and get them to test system using real data
- Formal usability testing – take application into lab, take notes on what users are doing and analyse what they do

Testing Methodology

- Develop problem statements – work out the design issues you want to investigate
- Choose a representative sample of users – who are the people that are going to be using the application
- Model the actual work environment – recreate the physical context the application will be used under
- Observer users' interaction with the model
- Collect quantitative and quality data
- Formulate recommendations for improvement – identify what is good and bad

Exploratory Testing

- Carried out early in development
- Aimed at evaluating preliminary design concepts
- Makes use of prototypes and paper based mock ups
- Focuses on qualitative data gathering

Assessment Testing

- Used early to midway through the development cycle
- Aimed at assessing how effectively the design is being implemented
- Makes use of prototypes and actual products
- Users perform actual tasks rather than just commenting on mock-ups
- Feedback is obtained

- Verify the implementation of design with respect to usability

Validation Testing

- Performed with the full product, get users in interact with the system with as little support as possible
- Conducted late in the development cycle, close to release of product
- Users perform actual tasks on the product with little or no support

Comparison Testing

- Can happen in conjunction with the other tests
- Compare usability with earlier versions or with competing products
- Must be similar in order for comparison to occur
- Aimed at establishing which two alternatives is preferred or most effective, or to discover advantages/disadvantages of each

Characterising Users

- Develop user profiles – highest level of detail about the basic demographic information
- Need to assign different characteristics to domain specific characteristics eg. experience with different products
- Consider computer centric issues such as learning style, experiences, type of interaction
- You can get user information from
    - Product specifications – will normally include the intended users
    - Marketing/design studies
    - Product manager
    - Help desk staff – where are users having the most difficulty.
    - Direct from the users

Test Monitor

- Sits next to user, guides user through test, say what to do, assist user if the system fails. Must have good technology and people skills, needs to have a good attention span
- Required Qualities
    - Understanding of usability engineering
    - Quick learner
    - Good rapport with participants
    - Excellent memory
    - Good listener
    - Comfortable with ambiguity
    - Flexible
    - Long attention span
    - Able to focus on the ‘big picture’
    - Good communicator
    - Good organizer
- Common problems for test monitors
    - Leading rather than enabling
    - Too involved in data collection

- Too knowledgeable
- Too rigid with the test plan
- Inhospitable to participants
- Jumping to conclusions

Developing a Test Plan

- Is a blueprint for the testing activity
- Essential process for clarifying the reasons for conducting the test
- Defines the methods of conducting the test, the measures to be used, and what will be reported
- Should include a schedule and costing for the test

Test Plan Format

- Purpose
  - High level objectives
  - Goals and desirables
  - Avoid inappropriate goals
- Problem statement/ test objectives
  - Beware of vague test objectives
  - Test objectives should produce measurable questions to which clear answers can be found eg. can the software be installed easily
- User profile
  - Characterise the target population
  - Identify each user category in terms of size of population and importance
  - Assesses the availability of representative users
  - Divide users into categories, balance the number of users in each category by the importance of the category
  - Ensure there is consistency between the groups and the tests performed by the groups
  - Keep tests simple
  - Note any problems during testing
- Method
- Task List
  - Give an initial outline of the testing activity
    - A brief description of the task
    - Required materials
    - Machine/system state
    - Specification of successful completion criteria
    - Maximum time limits for each task
  - Aim to expose usability problems indirectly
    - Do not draw attention directly to the feature under test
    - Setup a task which involves these features in their normal context use
    - Incorporate critical usability factors into the successful completion criteria
  - Risk analysis is essential to this process
  - Risk factors include
    - Frequency

        - Critically
        - vulnerability
- Test Monitor role
    - Clearly identify actions or areas of special interest, the test monitor will need to know what to be looking for
    - Specify the conditions under which the test monitor should intervene
    - What will be test monitor be recording
- Test environment/equipment
    - All equipment required to perform the tests
- Evaluation measures
    - Measures must be valid and reliable
    - A valid measure correlates strongly with the objective being tested
    - A reliable measure will give the same results repeatedly under the same conditions
    - Measures may be subjective or objective
    - Performance measures
        - Time to complete a tasks
        - Number of tasks completed correctly
        - Count of all the errors
        - Time required to access help
    - Preference measures
        - Ratings
        - Usefulness rating
        - Ease of use
        - Ease of learning
- Report contents and presentation
- Schedule
- Resource allocation and costing

Test Materials

- Screening questionaries
- Orientation script – read to each test subject to ensure everyone gets the same information
- Background questionnaire
- Data collection instrument
- Non-disclosure agreement, consent forms
- Pre-test questionnaire
- Task scenarios – need to be structured so there is a natural flow leading from one test to another, need to make sure it is idiot proof
- Prerequisite training materials
- Post test questionnaire – good for capturing data that is easily forgotten, get information on stuff easily forgotten
- Debriefing guide

Testing Environment

Need somewhere to test

Conducting Tests

- Monitor the session impartially – do not react to mistakes or correct behaviour
- Be aware of your voice and body language
- Treat each participant as an individual
- Don't rescue the participant
- Ensure participants are finished before moving to the next task
- Keep the atmosphere light
- Use the think aloud technique
    - Pros
        - You discover their preferences immediately rather than later
        - Can help participant to focus
        - Helps understand the source of incorrect behaviours
    - Cons
        - Some participants find it unnatural and distracting
        - Slows thinking and increases awareness
        - exhausting
- Interact appropriately
- Assist the participants only as a last resort
    - Never blame the participant
    - Help the participant clarify a problem

Summary of User Cantered Testing

- Involve users during development and testing is central to producing truly user-centred software
- Need a framework for that involvement
- May be applied at varying stages with varying degrees of formality
- Is costly and requires a great deal of expertise to administer correctly

Heuristic Evaluations

- Pros
    - Very cost effective
    - Can be done early in development
    - Less effort/expertise by internal staff
- Cons
    - Difficult finding suitable expert evaluators
    - Not as comprehensive as formal testing

Neilsen's Usability Heuristics

- Visibility of system status – system keeps users informed on what is going on, through appropriate feedback in reasonable time
- Match between system and real world – system should speak the users language, follow real world conventions
- User control and freedom – when action selected by mistake need some exit function to return to the previous state
- Consistency and standards – users should not have to wonder whether different words, actions mean the same thing
- Error prevention – good error messages to prevent the problem from happening

- Recognition rather than recall – make options visible don't make users remember information
- Flexibility and efficient use – allow users to tailor frequent actions
- Aesthetic and minimalist design – dialogues should not contain information that is irrelevant or rarely needed
- Help users recognise, diagnose and recover from errors
- Help and documentation – provide help to users

Formulating Recommendations

- Compile and summarise data
  - Do this at the end of each day
  - Provide statistical information mean, median, distribution graphs
- Analyse data
  - Identify hot spots – where do the most errors occur
  - Identify user difficulties
  - Identify source of errors
- Develop recommendations
  - Focus on high impact issues and solutions
  - Ignore political considerations
  - Provide short and long term solutions
- Produce the final report
  - Executive summary – major findings and recommendations
  - Method section – setup, user profiles, data collection methods
  - Results section – results summarise, raw data put in appendix
  - Findings and recommendations
  - Appendices – including raw data and user profiles do not include participants names