

INF3121

Veien til C

Fundamental concepts in testing

Why is testing necessary?

LO: Describe with examples the way in which software can cause harm to a person, environment or company
LO: Distinguish between a defect and its effects
LO: Give reasons why testing software is necessary
LO: Explain why testing depends on the level of risk, time and budget

Software systems context

Software systems are an important part of life:

- Most people had experience with software not working as expected.
- If the SW system doesn't work correctly, it can lead to problems like:
 - Loss of money
 - Loss of business reputation
 - Injury or death

Causes of software defects

Causes of software defects Human errors

- Internal
 - Fatigue
 - Lack of training
 - Lack of understanding
 - Lack of interest

- External
 - Time pressure
 - Complex code
 - Many system interactions
 - Changed technologies

Non-controllable events (i.e. environmental conditions)

- Radiation
- Magnetism
- Pollution

Both causes of errors produce defects (= faults, bugs) in the code.

- Defects, if executed, may result in failures of the SW system (the system will fail to do what it should).
- Failures can affect seriously the users of the SW system, i.e.:
 - Break pedal not working in some cars
 - Miscalculations in financial SW systems

Role of testing

Testing has an important role in all the stages of SW products life cycle:

- Development
- Maintenance
- Operations

The role of testing is:

- To reduce the risk of problems occurring during operations
- To check if the SW system meets:
 - legal requirements
 - Industry specific standards
- To learn more about the SW system

Testing and quality

Measures the quality of certain characteristics of the SW in terms of defects found

- Functional aspects
- Non-functional aspects (Reliability, Usability, Portability)

Gives confidence in the quality of the SW

- That is, if there are few defects found and properly tested

Teaches us lessons to apply in future projects

- By understanding the root causes of defects, processes can be improved. This can prevent defects from reoccurring.

How much testing is enough?

Testing should provide sufficient information for the stakeholders to take informed decisions about:

- Release of the software
- Next development steps, etc

Depends on:

- Level of risk:
 - Technical risks
 - Business risks
 - Project constraints
- Project constraints:
 - Time
 - Budget

What is testing?

LO: Recall the common objectives of testing
LO: Provide examples for focus of testing in different phases of the software life-cycle
LO: Differentiate testing from debugging

Definition

The process consisting:

- of all life cycle activities: both static and dynamic,

concerned with:

- planning, preparation and evaluation

of:

- software products and related work products

to:

- determine that they satisfy specified requirements
- to demonstrate that they are fit for purpose
- and to detect defects.
-

Depending on the objectives of the test process, testing can be focused on:

- Confirming that the SW system meets the requirements
- Causing as many failures as possible
- Check that no defects have been introduced during dev. changes

- Assess the quality of the SW (with no intention of finding bugs)

Test principles

LO: Explain the seven principles in testing

The testing principles

The testing principles offer some guidelines common for all testing:

P1 - Testing shows presence of defects

- Testing can show that defects are present, but cannot prove there are no defects.
- Testing reduces the probability of undiscovered defects remaining in the software; but even if no defects are found, this is not a proof of correctness .

P2 - Exhaustive testing is impossible

- Testing everything is not feasible. We use risks and priorities to focus test effort.

P3 - Early testing

- Testing should start as soon as possible in the development lifecycle and should be focused on defined objectives.

P4 - Defect clustering

- • A small number of modules contain most of the defects discovered during pre-release testing

P5 - Pesticide paradox

- If the same set of tests will be repeated over and over, it will no longer find new bugs.

P6 - Testing is context dependent

- I.e., safety-critical SW is tested differently from an ecommerce site.

P7 - Absence-of-errors fallacy

- Finding and fixing defects does not help if the SW system is un-usable or does not meet user's expectations.

Fundamental test process

LO: Recall the five fundamental test activities and respective tasks from planning to closure

Plan and control

Plan means: what, who, when?

- Scope, objectives and risks of testing
- Test levels and types that will be applied
- Documentation that will be produced
- Assign resources for the different test activities
- Schedule test implementation, execution,
- evaluation

Control and adjust the planning to reflect new information, new challenges of the project.

Analysis and design

Review test basis

- Requirements
- Product architecture
- Product design
- Interfaces

Analysis

General test objectives are transformed into

- Test conditions
- Test cases

Design

- Test cases
- Test environments
- Test data

Implementation and execution Implementation and execution

Implement

- Group tests into scripts
- Prioritize the scripts
- Prepare test oracles
- Write automated test scenarios

Execute

- Run the tests and compare results with oracles

- Report incidents
- Repeat test activities for each corrected discrepancy

Evaluate exit criteria and report

Evaluate

- Assess test execution against the defined objectives
- Check if
 - More tests are needed
 - Exit criteria should be changed

Report

- Write or extract a test summary report for the stakeholders.

Test closure activities

What deliverables have been delivered? Are there any remaining ones we need to re-plan?

Check the closure of incident reports

Archive the items delivered:

- SW code
- Set of tests & results
- Documentation created/updated (this is necessary for future reuse)

Analyze lessons learned for future releases.

The psychology of testing

LO: Recall the psychological factors that influence the success of testing
LO: Contrast the mindset of a tester and of a programmer

Independence test levels

- A certain degree of independence is often more effective at finding defects and failures. However, the developer can very efficiently find bugs in their own code.
- Applying a certain level of independence of the testing depends on the objective of testing.
- Independence levels:
 - Tests designed by the same person that wrote the code

- Tests designed by another person from the same team (dev. colleague), but same organization
- Tests designed by a person from a different team (QA colleague), but same organization
- Tests designed by a person from a different organization / company (outsourcing the testing)

Tips and tricks

- Looking for failures requires:
 - Curiosity
 - Professional pessimism
 - Attention to details
 - Good communication skills
 - Experience on error guessing
- Communicate failures in a constructive way: fact-focused; give factual reports and review findings
- Be clear and objective
- Confirm that:
 - You have understood the requirements
 - The person that has to fix the bug has understood the problem

Testing throughout the software life cycle

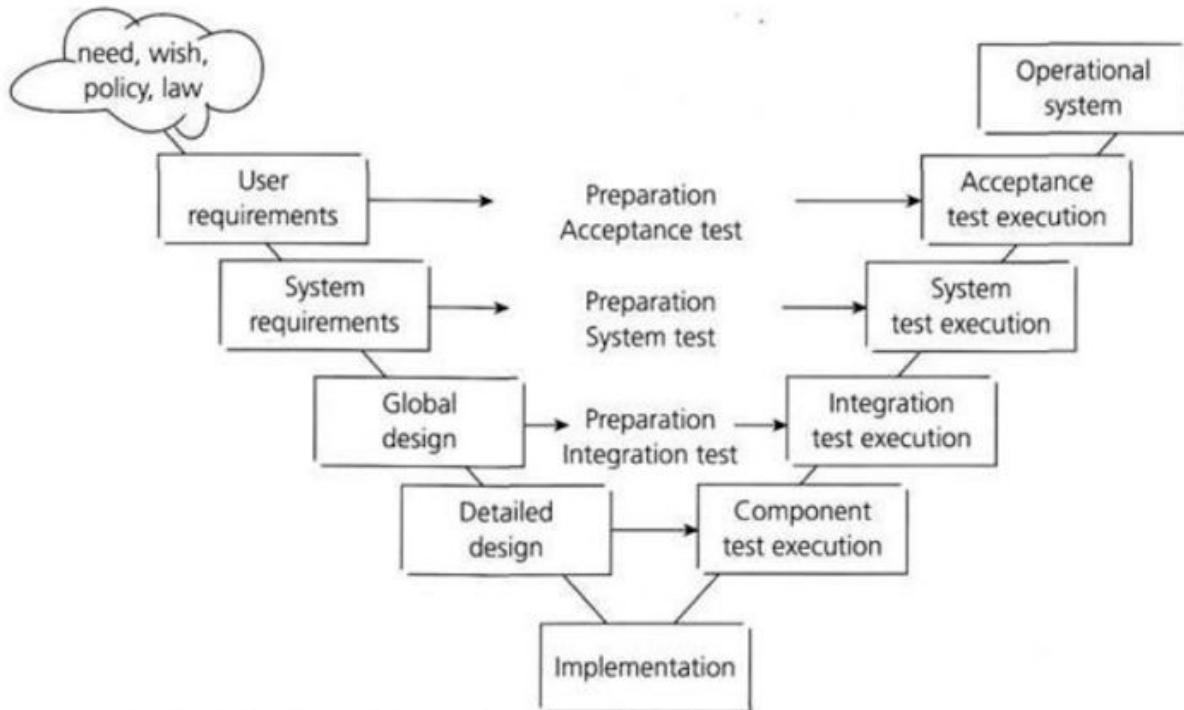
Software development models

LO: Explain the relationship between development, test activities and work products in the development life-cycle, by giving examples

LO: Recognize the adaptability of the software development model to the context of a project and product characteristics
--

LO: Recall the characteristics of good testing, applicable to any life-cycle model
--

Sequential development model (V-model)



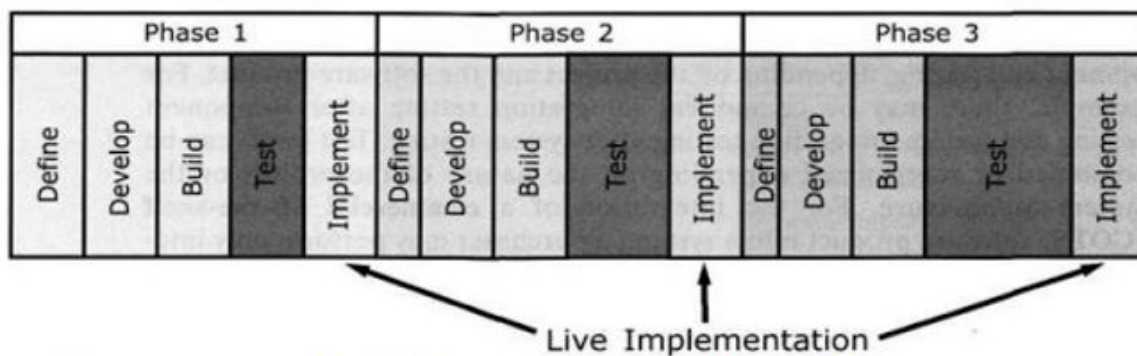
Testing needs to begin as early as possible in the life cycle.

Testing can be integrated into each phase of the product life cycle.

Within the V-model, validation testing takes place especially during

- the early stages (i.e. reviewing the user requirements),
- and late in the life cycle (i.e. during user acceptance testing).

Iterative-incremental development model



Iterative-incremental development is the process of:

- establishing requirements,

- designing,
- building
- and testing a system,

done as a series of shorter development cycles.

An increment, added to others developed previously, forms a growing partial system, which should also be tested.

Regression testing is increasingly important on all iterations after the first one.

Testing within a life cycle model

In any life cycle model, there are several characteristics of good testing:

- Every development activity has a corresponding testing activity.
- Each test level has test objectives specific to that level.
- The analysis & design of tests for a given test level should begin during the corresponding development activity.
- Testers should be involved in reviewing documents as soon as drafts are available in the development life cycle.

Test levels can be combined or reorganized depending on the nature of the project or the system architecture

Test Levels

LO: Compare the different levels of testing: major objectives, typical objects of testing, typical targets of testing and related work products, people who tests, types of defects found, types of failures to be identified

The test levels

- **Acceptance**
Is the responsibility of the customer – in general. The goal is to gain confidence in the system; especially in its nonfunctional characteristics
- **System**
The behavior of the whole product(system) as defined by the scope of the project
- **Integration**
Interface between components; interactions with other systems (OS, HW, etc)
- **Unit**
Any module, program, object separately testable

For each test level, please note

- The generic objectives
- The test basis (docs/products used to derive test cases)
- The test objects (what is being tested)
- Typical defects and failures to be found
- Specific approaches and responsibilities

Component testing

Objectives: Component testing includes testing of functionality and specific non-functional characteristics, such as:

- resource-behavior (e.g. memory leaks)
- robustness testing
- structural testing (e.g. branch coverage)

Component testing

Test basis

- specification of the component
- software design
- the data model

Tools - Stubs, drivers and simulators may be used.

Approaches

- test-driven development (prepare and automate test cases before coding)
- Defects are fixed as soon as they are found, without formal recording of incidents

Responsible - (QA or programmer in theory) Programmer in practice

Integration testing

Objectives

It tests:

- interfaces between components,
- interactions with different parts of a system, such as
 - the operating system
 - file system
 - hardware
 - interfaces between systems

Types of integration tests

1. Component integration: tests the interactions between software components
 - is done after component testing

2. System integration: tests the interactions between different systems
 - is done after system testing.

Approaches

To reduce the risk of late defect discovery, integration should normally be incremental rather than “big bang”.

Both functional and structural approaches may be used.

Responsible

Ideally, testers should understand the architecture and influence integration planning.

System testing

Objectives

- Testing the behavior of the whole system as defined by the scope of the project

Test basis

- System requirements specification
- Business processes
- Use cases
- Other high level description of the: system behavior, interactions with OS/system resources

Requirements may exist as text and/or models.

Testers also need to deal with incomplete or undocumented requirements.

Approaches

Test environment should correspond to the production environment as much as possible.

- First, the most suited black-box technique
- Then, white-box technique to assess the thoroughness of testing

Responsible

Independent test team

Acceptance testing

Types of acceptance testing

- **User acceptance testing** - verifies the fitness for use of the system by users.
- **Operational testing** (done by the system administrators, and include)

- testing of backup/restore
- disaster recovery
- user management
- maintenance tasks
- periodic checks of security vulnerabilities
- **Contract and regulation acceptance testing** - performed against a contract's acceptance criteria (i.e. governmental, legal or safety regulations)
- **Alpha and beta testing**
 - **Alpha testing** is performed at the developing organization's site.
 - **Beta testing** (field testing), is performed by people at their own locations.

Both are performed by potential customers, not the developers of the product.
- **Responsible:** customers or users of a system; other stakeholders may be involved as well

Test Types

LO: Compare the following test types: functional, non-functional, structural, change-related
Identify and describe non-functional test types based on non-functional requirements
Describe the test types based on the analysis of a software's structure
Describe the purpose of confirmation and regression testing

Test types

“What” the system does		“How” the system works	
Suitability	Functional	Non functional	Performance, Load, Stress
Interoperability			Reliability (robust, fault tolerant, recoverable)
Security			Usability (understand, learn, operate, like)
Accuracy			Efficiency (time behavior, resource utilization)
Compliance			Maintainability (analyze, change, stabilize, test)
Confirmation testing	Related to changes	Structural	Portability (adapt, install, co-exist, replace)
Regression testing			Code coverage

Functional testing (Black box testing)

Objectives

Test what a system should do and considers the external behavior of the software.

Test levels

May be performed at all test levels (e.g. tests for components may be based on a component specification).

Test basis

The expected behavior description can be found in work products such as

- requirements specification
- use cases
- functional specification
- may be undocumented.

Non functional testing

Objectives

measuring characteristics of software that can be quantified on a varying scale: – ex: response times for performance testing.

Test levels

May be performed at all test levels.

Structural testing (white box testing)

Objectives

Help measure the thoroughness of testing through assessment of coverage of a type of structure.

Test levels

May be performed at all test levels, but especially in component testing and component integration testing.

Approach

- Structural techniques are best used after specification-based techniques, in order to help measure the thoroughness of testing
- Structural testing is based on the architecture of the system(i.e. a calling hierarchy)

Tools can be used to measure the code coverage of elements, such as statements or decisions.

Testing related to changes

Confirmation testing

After a defect is detected and fixed, the software should be retested to confirm that the original defect has been successfully removed.

Regression testing

The repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the change(s).

Test levels

- may be performed at all test levels,
- applies to functional, non-functional and structural testing.

Approach

- The extent of regression testing is based on the risk of finding defects in software that was working previously.
- Regression test suites are run many times and generally evolve slowly, so regression testing is a strong candidate for automation.

Maintenance testing

LO: Explain the triggers for maintenance testing
LO: Explain how the amount of maintenance testing depends on the type of change
LO: Describe the role of regression testing in maintenance
LO: Describe the role of impact analysis in maintenance

Objectives

Maintenance testing is done on an existing operational system, and is triggered by modifications, migration, or retirement of the software or system.

Types of maintenance testing

- Modifications
 - planned enhancement changes (e.g. release-based)
 - corrective and emergency changes (patches), and changes of environment (operating system or database upgrades)
- Migration (e.g. from one platform to another)
 - operational tests of the new environment
 - tests on the changed software.
- Retirement of a system
 - the testing of data migration or archiving if long data-retention periods are required.

Maintenance testing also includes extensive regression testing to parts of the system that have not been changed

The scope of maintenance testing is related to

- the risk of the change

- the size of the existing system
- the size of the change.

Test levels

Depending on the changes, maintenance testing may be done at any or all test levels and for any or all test types.

Approach

Determining how the existing system may be affected by changes is called impact analysis, and is used to help decide how much regression testing to do.

Note

Maintenance testing can be difficult if specifications are out of date or missing.

Static Techniques

Static techniques and the test process

LO: Describe the importance and value of applying static test techniques for the assessment of software and related work products
LO: Explain the difference between static and dynamic test techniques, taking into consideration: the objectives of testing, types of defects to be discovered, the role of the technique in the software life-cycle

Static techniques

Reviews, static analysis and dynamic testing have the same objective – identifying defects. They are complementary. Compared to dynamic testing, static techniques find causes of failures (defects) rather than the failures themselves.

- **Dynamic testing** - requires the execution of software.
- **Static testing** - manual examination and automated analysis of the code or documentation.
- **Reviews** = a way of testing software products (including code) and can be performed well before dynamic test execution.
- **Reason to make reviews**
 - Defects detected during reviews early in the life cycle are often cheaper to remove than those detected while running tests.
 - Reviews can find omissions, for example, in requirements, which are unlikely to be found in dynamic testing.

- **Tools** (manual + tool support) The main manual activity is to examine a work product and make comments about it.
- **Object of reviews:** Any software work product can be reviewed
 - requirements specifications
 - design specifications
 - code
 - test plans, test specifications, test cases, test scripts
 - user guides
 - web pages
- **Benefits**
 - early defect detection and correction
 - development productivity improvements
 - reduced development timescales
 - reduced testing cost and time
 - lifetime cost reductions
 - fewer defects and improved communication
- **Typical defects** (easier to find in reviews than in dynamic testing)
 - deviations from standards
 - requirement defects
 - design defects – insufficient maintainability – incorrect interface specifications

Review process

LO: Recall the activities, roles and responsibilities of a typical formal review
LO: Explain the differences between the four different types of reviews: informal, technical, walkthrough and inspection
LO: Explain the factors for successful performance of reviews

Background

Different types of reviews vary from

- very informal (e.g. no written instructions for reviewers)
- to very formal (i.e. well structured and regulated)

The formality of a review process is related to factors like

- the maturity of the development process
- any legal or regulatory requirements
- the need for an audit trail

The way a review is carried out depends on the agreed objective of the review

- find defects

- gain understanding
- discussion and decision by consensus

Phases of a formal review

1. Planning	select the personnel
	allocate roles
	define the entry and exit criteria for more formal review types (e.g. inspection)
	select which parts of documents to look at
2. Kick-off	distributing documents
	explaining the objectives
	process and documents to the participants
	and checking entry criteria (for more formal review types)
3. Individual preparation	Work done by each of the participants on their own before the review meeting, noting potential defects, questions and comments.
4. Review meeting	Discussion or logging, with documented results or minutes (for more formal review types).
	The meeting participants may simply note defects, make recommendations for handling the defects, or make decisions about the defects.
5. Rework	fixing defects found, typically done by the author.
6. Follow-up	check that defects have been addressed
	gather metrics

Roles and responsibilities

1. Manager	decides on the execution of reviews
	allocates time in project schedules
	determines if the review objectives have been met
2. Moderator	leads the review of the document or set of documents
	planning the review
	running the meeting
	follow-up after the meeting
	If necessary, the moderator may mediate between the various points of view and is often the person upon whom the success of the review rests
3. Author	the writer or person with chief responsibility for the documents to be reviewed.
4. Reviewers	Individuals with specific technical or business background
	Identify and describe the findings in the product under review Note: reviewers should be chosen to represent different perspectives and roles in the review process
	Note: reviewers should take part in the review meeting
5. Scribe	Documents all the issues, problems and open points that were identified during the meeting

Types of reviews

1. Informal review	Inexpensive way to get some benefit.
	Form: pair programming or a technical lead reviewing designs and code
	no formal process
	optionally may be documented
2. Walkthrough	Learning, gaining understanding, defect finding.
	Form: meeting led by author
	scenarios, dry runs, peer group
	open-ended sessions
	may vary in practice from quite informal to very formal
3. Technical review	Discuss, make decisions, evaluate alternatives, find defects, solve technical problems and check conformance to specifications and standards.
	Form: may be performed as a peer review without management participation
	ideally led by trained moderator
	documented, defined defect-detection process; includes peers and technical experts
	pre-meeting preparation
	optionally the use of checklists, review report, list of findings and management participation
4. Inspection	Find defects.
	Form: usually peer examination led by trained moderator (not the author)
	formal process based on rules and checklists with entry and exit criteria
	pre-meeting preparation
	defined roles
	includes metrics
	inspection report, list of findings
	formal follow-up process

Success factors for reviews

Objectives

Each review has a clear predefined objective.

Roles

The right people for the review objectives are involved.

Approach

- Defects found are welcomed, and expressed objectively.
- Apply suitable review techniques for the type and level of software products.
- Use checklists or roles if appropriate to increase effectiveness of defect identification.
- Management supports a good review process (e.g. by incorporating adequate time for review activities).

Training and learning

- Training is given in review techniques, especially the more formal techniques, such as inspection.
- There is an emphasis on learning and process improvement

Static analysis by tools

LO: Recall typical defects and errors identified by static analysis
LO: Compare the type of defects found in static testing with the types of defects found in dynamic testing
LO: Describe, using examples, the typical benefits of static analysis
LO: List typical code and design defects that may be identified by static analysis tools

Static analysis by tools

Objectives of static analysis, find defects in:

- software source code
- software models
- Note! Static analysis finds defects rather than failures

Static analysis is performed without actually executing the software being examined by the tool.

Static analysis tools analyze program code (e.g. control flow and data flow), as well as generated output such as HTML and XML.

Typical defects discovered by static analysis tools include

- referencing a variable with an undefined value
- inconsistent interface between modules and components
- variables that are never used
- unreachable (dead) code
- programming standards violations
- security vulnerabilities
- syntax violations of code and software models.

Developers	Use static analysis before and during <ul style="list-style-type: none">• Component testing• Integration testing
Designers	Use static analysis during software modeling

Practical side

Static analysis tools may produce a large number of warning messages, which need to be well managed to allow the most effective use of the tool.

Why is static analysis valuable

- Early detection of defects prior to test execution
- Early warning about suspicious aspects of the code or design, by the calculation of metrics, such as a high complexity measure
- Identification of defects not easily found by dynamic testing
- Detecting dependencies and inconsistencies in software models, such as links
- Improved maintainability of code and design
- Prevention of defects, if lessons are learned in development

Test Design Techniques

The test development process

LO: Differentiate between specifications for: test design, test case and test procedure
LO: Define and compare: test condition, test case and test procedure
LO: Design test cases starting from a set of software requirements

LO: Organize test cases into a well-structured test procedure specification
LO: Evaluate the quality of test cases In terms of traceability to the requirements and expected results

Background

The test design process can be done in different ways, from very informal (little or no documentation), to very formal.

The level of formality depends on the context of the testing, including:

- The maturity of testing process
- The maturity of development process
- The organization
- Time constraints
- People involved

Test analysis

The test basis documentation is analyzed in order to determine what to test, i.e. to identify the test conditions.

Test condition (Def.) = an item or event that could be verified by one or more test cases (A function, a transaction, a quality characteristic (security, i.e.), other structural element (menus in web pages, i.e.))

Test design

During test design: Test cases and test data are created and specified.

Test case is a set of:

- Pre-conditions
- Inputs
- Expected results
- Post-conditions

Developed to cover certain test condition(s). (cf. IEEE 829)

Expected results include

- Outputs
- Changes to data and states
- Any other consequence of the test

If expected results have not been defined, then a plausible but erroneous result may be interpreted as the correct one.

Expected results should ideally be defined prior to test execution.

Test implementation

During test implementation the test cases are in the test procedure specification

1. Developed
2. Implemented
3. Prioritized
4. Organized

The test procedure (= manual test script): specifies the sequence of action for the execution of a test.

Test script (= automated test procedure) If tests are run using a test execution tool, the sequence of actions is specified in a test script.

The test execution schedule defines

- The order of execution of the test procedures & (possibly) automated test scripts
- When they are to be carried out
- By whom to be executed

The test execution schedule will take into account such factors as

- regression tests
- prioritization
- technical and logical dependencies

Categories of test design techniques

LO: Recall the purpose of both black-box testing and white-box testing. Give example of techniques for each type of technique
LO: Explain the characteristics, differences and cases in which to use blackbox, white-box and experience based testing techniques

Categories of test design techniques

Common features of black-box techniques:

- We use models (formal or informal) to specify the problem to be solved, the software or its components
- We derive systematically the test cases from these models

Common features of white-box techniques:

- The test cases are derived from information about how the software is constructed for example: code and design

- For the existing test cases, we can measure the coverage of the software
- Further test cases can be derived systematically to increase coverage

Common features of experience-based techniques:

- The test cases are derived from the knowledge and experience of people
 - Knowledge of testers, developers, users and other stakeholders about the software, its usage and its environment
 - Knowledge about likely defects and their distribution

Specification-based testing (Black-box)

LO: Write test cases from given software models using: EP, BVA, DT and ST test design techniques

LO: Explain the purpose of each black-box test design technique

LO: Explain the concept of use-case testing and its benefits

Equivalence partitioning

Typical problem:

A savings account in a bank earns a different rate of interest depending on the balance in the account.

In order to test the software that calculates the interest due, we can identify the ranges of balance values that earn the different rates of interest.

If a balance in the range \$0 up to \$100 has a 3% interest rate, a balance over \$100 and up to \$1000 has a 5% interest rate, and balances of \$1000 and over have a 7% interest rate, we would initially identify three valid equivalence partitions and one invalid partition as shown below.

-\$0.01	\$0.00	\$100.00	\$100.01		\$999.99		\$1000.00
Invalid partition		Valid (for 3% interest)			Valid (for 5% interest)		Valid (for 7% interest)

Technique:

Inputs/Outputs/Internal values of the software are divided into groups that are expected to exhibit similar behavior.

Equivalence partitions (or classes) can be found for both valid data and invalid data, i.e. values that should be rejected.

Notes:

- Tests can be designed to cover partitions.
- Equivalence partitioning is applicable at all levels of testing.
- Equivalence partitioning as a technique can be used to achieve input and output coverage.

Boundary value analysis

Example:



Boundary analysis

Analysis at the edge of each equivalence partition. Why: because there, the results are more likely to be incorrect.

The maximum and minimum values of a partition are its boundary values.

Valid and invalid boundary

- A boundary value for a valid partition is a valid boundary value
- The boundary of an invalid partition is an invalid boundary value. Tests can be designed to cover both valid and invalid boundary values.

Notes

- Boundary value analysis can be applied at all test levels
- It is relatively easy to apply and its defect finding capability is high
- Detailed specifications are helpful
- This technique is often considered as an extension of equivalence partitioning
- Boundary values may also be used for test data selection

Decision table testing

Example

Fictional wine monopoly store

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Oslo resident?	False	True	True	True
Over 18 years?	Don't care	False	True	True
Happy hour?	Don't	Don't	False	True

	care	care		
Actions				
Can buy wine?	False	False	True	True
Offer 10% discount?	False	False	False	True

Decision tables are a good way to

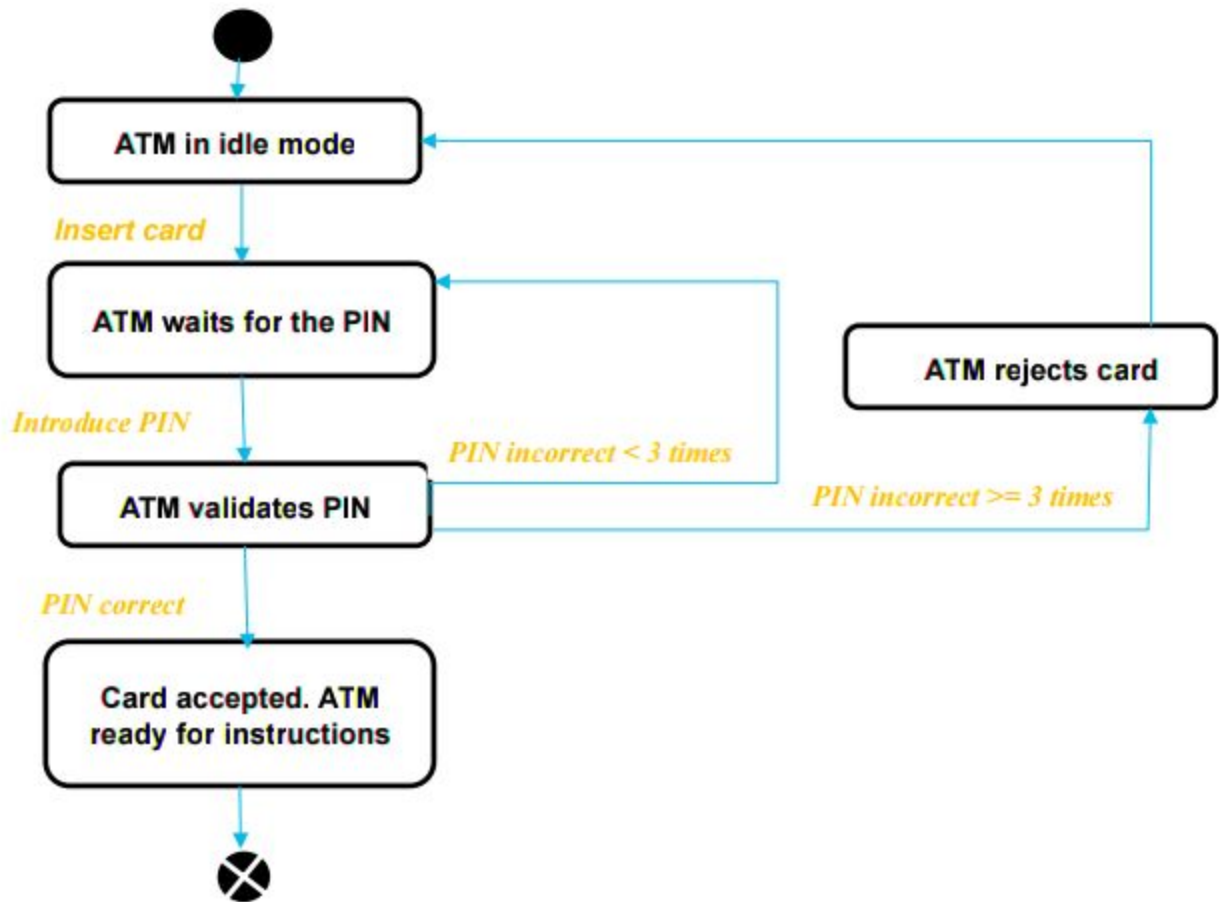
- capture system requirements that contain logical conditions
- and to document internal system design.

The input conditions and actions are most often stated in such a way that they can either be true or false (Boolean).

The strength of decision table testing is that it creates combinations of conditions that might not otherwise have been exercised during testing. It may be applied to all situations when the action of the software depends on several logical decisions.

State transition testing

Typical problem



Why? Because a system may exhibit a different response depending on current conditions or previous history (its state).

- It allows the tester to view
- the software in terms of its states
- transitions between states
- the inputs or events that trigger state changes (transitions)
- the actions which may result from those transitions

The states of the system under test are separate, identifiable and finite in number.

A state table can highlight possible transitions that are invalid.

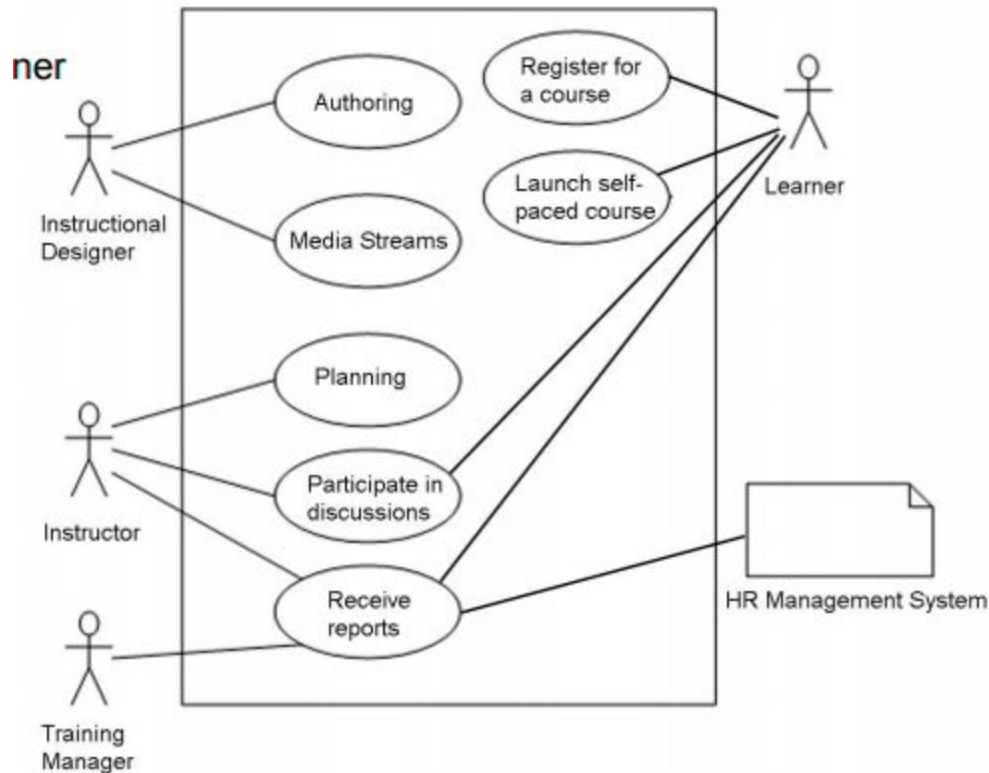
Tests can be designed:

- To cover a typical sequence of states
- To cover every state
- To exercise every transition
- To exercise specific sequences of transitions
- To test invalid transitions.

Use case testing

Typical problem 1, an on-line training website

- User 1: the learner
- User 2: the tutor (instructor)
- User 3: the training manager
- User 4: the instructional designer
- (User 5): the HR management system



Typical problem 2, a customer playing at the a store internal lottery, based on the order number of a previous purchase. The customer is interacting with an automated validator.

Customer	System
Enters order number	
	Detects that the order number matches the winning number of the month
	Register the user and order number as this month's winner
	Sends an email to the sales manager

	Congratulates the customer and gives her instructions on how to collect the prize
Exits the system	

Use case - describes interactions between actors (including users and the system), which produce a result of value to a system user.

Each use case has pre-conditions, which need to be met for a use case to work successfully.

Each use case terminates with post-conditions, which are the observable results and final state of the system after the use case has been completed.

A use case usually has a mainstream (i.e. most likely) scenario, and sometimes alternative branches.

Use cases are very useful for designing acceptance tests with customer/user participation.

Structure-based testing (White-box)

LO: Describe the concept of code coverage and the reasons why it is useful
LO: Explain the concepts of statement coverage and decision coverage. Explain why they can be used for more than component testing
LO: Write test cases for statement and decision coverage, from given control flows
LO: Assess statement and decision coverage for completeness with respect to different exit criteria

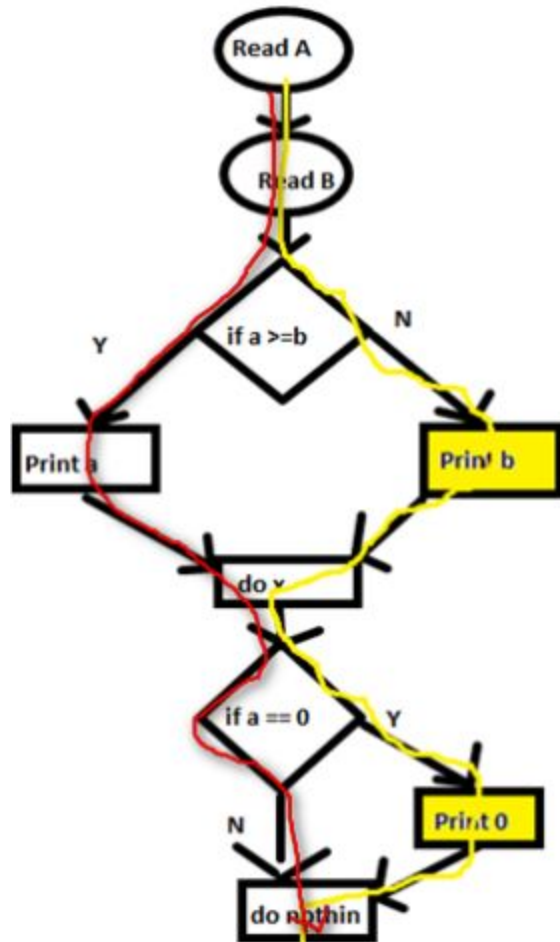
Background

Structure-based testing (white-box) is based on an identified structure of the software.

- Component level: the structure is that of the code itself: statements, decisions or branches
- Integration level: the structure may be a call tree (a diagram in which modules call other modules)
- System level: the structure may be a menu structure, business process or web page structure

Statement testing & coverage

In component testing: Statement coverage is the percentage of executable statements that have been exercised by a test case suite.



Decision coverage - is the assessment of the percentage of decision outcomes (e.g. the True and False options of an IF statement) that have been exercised by a test case suite.

Decision coverage is stronger than statement coverage: 100% decision coverage guarantees 100% statement coverage, but not vice versa.

Other structure-base techniques

There are stronger levels of structural coverage beyond decision coverage, i.e.:

- multiple condition coverage.

The concept of coverage can also be applied at other test levels, (i.e. integration level) where the percentage of

- modules, components or classes

that have been exercised by a test case suite could be expressed as

- module, component or class coverage.

Experience-based testing

LO: Recall the reasons for writing test cases based on intuition, experience and knowledge about common defects

LO: Compare experience-based techniques with specification-based techniques

Experienced-based testing

- Tests are derived from the tester's skill and intuition and their experience with similar applications and technologies.
- Especially applied after more formal approaches

1. Error guessing = a commonly used experienced-based technique.

- Generally testers anticipate defects based on experience.
- A structured approach to the error guessing technique is to enumerate a list of possible errors and to design tests that attack these errors.
- This systematic approach is called fault attack.

2. Exploratory testing = concurrent test design, test execution, test logging and learning, based on a test charter containing test objectives, and carried out within time-boxes.

It is most useful:

- where there are few or inadequate specifications
- under severe time pressure
- to complement other, more formal testing. It can serve to help ensure that the most serious defects are found.

Choosing test techniques

LO: Recall the factors that influence the choice of test design techniques
--

The choice of which test techniques to use depends on a number of factors, including:

- Type of system

- Time and budget
- Development lifecycle
- Regulatory standards
- Customer requirements
- Contractual requirements
- Test objectives
- Level of risk
- Type of risk
- Documentation available
- Use case models
- Knowledge of the testers
- Previous experience with types of defects found

Test Management

Test organization

LO: Describe the levels of independent testing
LO: Explain the benefits of having independent testing of software
LO: Enumerate which organization roles can be involved in independent testing. Explain the contribution that each role can make
LO: Enumerate the typical tasks of tester and of test leaders

Test organization and independence

The effectiveness of finding defects by testing and reviews can be improved by using independent testers

Options for independence are

1. No independent testers. Developers test their own code
2. Independent testers within the development teams
3. Independent test team or group within the organization, reporting to project management or executive management
4. Independent testers from the business organization or user community
5. Independent test specialists for specific test targets such as usability testers, security testers or certification testers (who certify a software product against standards and regulations).

For large, complex or safety critical projects, it is usually best to have multiple levels of testing, with some or all of the levels done by independent testers.

Development staff can participate in testing, especially at the lower levels

Advantages & disadvantages

Positive

- Independent testers see other and different defects, and are unbiased
- An independent tester can verify assumptions people made during specification and implementation of the system

Negative

- Isolation from the development team (if treated as totally independent)
- Independent testers may be the bottleneck as the last checkpoint
- Developers may lose a sense of responsibility for quality

Note

Testing tasks may be done by people in a specific testing role, or may be done by someone in another role, such as

- project manager
- quality manager
- developer
- business and domain expert
- infrastructure or IT operations (this can be both good and bad)

Tasks of the test leader

Test leader = test manager / test coordinator.

The test leader plans, monitors and controls the testing activities and tasks.

- **Coordination** of the test strategy and plan with project managers
- **Plan the tests**, understanding the test objectives and risks
 - selecting test approaches
 - estimating the time, effort and cost of testing
 - acquiring resources
 - defining test levels, cycles
 - planning incident management
- **Test specs**, preparation and execution Initiate the specification, preparation, implementation and execution of tests
 - monitor the test results
 - check the exit criteria
- **Adapt planning**: based on test results and progress and take any action to compensate for problems
- **Manage test configuration**: Set up adequate configuration management of testware for traceability.

- **Introduce metrics** for measuring test progress and evaluating the quality of the testing & the product.
 - **Automation of tests:** Decide what should be automated, to what degree, and how.
 - **Select test tools:** Select tools to support testing and organize trainings for tool users.
 - **Test environment:** Decide about the implementation of the test environment.
 - **Test summary reports:** Write test summary reports based on the information gathered during testing.
-
- **Test plans:** Review and contribute to test plans
 - **Requirements and specs:** Analyze, review and assess user requirements, specifications and models for testability.
 - **Test specifications:** Create test specifications.
 - **Test environment:** Set up the test environment (often coordinating with system administration and network management)
 - **Test data:** Prepare and acquire test data.
 - **Testing process:** Implement tests on all test levels, execute and log the tests, evaluate the results and document the deviations from expected results.
 - **Test tools:** Use test administration or management tools and test monitoring tools as required.
 - **Test automation:** Automate tests (may be supported by a developer or a test automation expert).
 - **Other metrics:** Measure performance of components and systems (if applicable).
 - **Help the others:** Review tests developed by others

Test planning and estimation

LO: Enumerate the different levels and objectives of test planning
LO: Explain the purpose and the content of the test plan
LO: Differentiate between the different test approaches: analytical, modelbased, methodical, process-compliant, heuristic, consultative, regression-averse
LO: Write a test execution schedule for a given set of test cases, considering prioritization and technical and logical dependencies
LO: Recall the typical factors that influence the effort put in testing
LO: Differentiate between metric-based approach and expert-based approach
Define entry criteria and exit criteria, together with their goals

Test planning

Test planning is a continuous activity and is performed in all life cycle processes and activities.

Feedback from test activities is used to recognize changing risks so that planning can be adjusted.

Planning may be documented in:

- a project or master test plan
- and in separate test plans for test levels, such as system testing and acceptance testing.

Planning is influenced by:

- the test policy of the organization
- the scope of testing
- objectives, risks, constraints
- criticality, testability and the availability of resources

Test planning activities

Scope and risk	Determining the scope and risks of testing
Objectives	Identifying the objectives of testing
Overall approach	Defining the overall approach of testing, including <ul style="list-style-type: none">• the definition of the test levels• entry and exit criteria.
Test activities	Integrating and coordinating the testing activities into the software life cycle activities <ul style="list-style-type: none">• acquisition• supply• development• operation• maintenance
Strategy	Making decisions about <ul style="list-style-type: none">• what to test• what roles will perform the test activities• how the test activities should be done• how the test results will be evaluated
Schedule	Scheduling test analysis and design activities. Scheduling test implementation, execution and evaluation.
Resources	Assigning resources for the different activities defined

Metrics	Selecting metrics for monitor
----------------	-------------------------------

Entry criteria

Entry criteria defines when to start testing.

Typically entry criteria may consist of:

- Test environment availability and readiness
- Test tool readiness in the test environment
- Testable code availability
- Test data availability

Exit criteria

The purpose of exit criteria is to define when to stop testing, such as at the end of a test level or when a set of tests has a specific goal.

Typically exit criteria may consist of:

- Thoroughness measures
 - Coverage of code
 - Functionality coverage
 - Risk
- Estimates
 - Defect density
 - Reliability measures
- Cost
- Residual risks
 - Defects not fixed
 - Lack of test coverage in some areas
- Schedule
 - Time to market

Test estimation

Two approaches for the estimation of test effort are covered in this syllabus:

- **The metrics-based approach:** Estimating the testing effort based on metrics of former or similar projects or based on typical values.
- **The expert-based approach:** Estimating the tasks by the owner of these tasks or by experts.

Once the test effort is estimated, resources can be identified and a schedule can be drawn up. The testing effort may depend on a number of factors, including:

Characteristics of the product	<ul style="list-style-type: none"> • the quality of the specification • the size of the product • the complexity of the problem domain • the requirements for reliability and security
Characteristics of the development process	<ul style="list-style-type: none"> • skills of the people involved • and time pressure
The outcome of testing	<ul style="list-style-type: none"> • the number of defects • the amount of rework required

Test approaches (test strategies)

One way to classify test approaches or strategies is based on the point in time at which the bulk of the test design work is begun:

Preventative approaches

Tests are designed as early as possible

Reactive approaches

Test design comes after the software or system has been produced

Typical approaches or strategies include:

Analytical appr.	risk-based testing - testing is directed to areas of greatest risk.
Model-based appr.	stochastic testing using statistical information about failure rates (such as reliability growth models)
Methodical appr.	failure-based (including error guessing and fault-attacks), experiencedbased, check-list based, and quality characteristic based.
Process- or standardcompliant appr.	specified by industry-specific standards or the various agile methodologies.
Dynamic and heuristic appr.	exploratory testing, execution & evaluation are concurrent tasks.
Consultative appr.	test coverage is evaluated by domain experts

	outside the test team.
Regression-averse appr.	include reuse of existing test material, extensive automation of functional regression tests.

Test progress monitoring and control

LO: Recall common metrics used for test preparation and execution
LO: Explain and compare metrics used for test reporting (e.g.: defects found & fixed, tests passed & failed)
LO: Summarize the content of the test summary report, according to IEEE-829

Test progress monitoring

The purpose of test monitoring is to give feedback and visibility about test activities.

Information to be monitored may be collected manually or automatically and may be used to measure exit criteria, such as coverage.

Metrics may also be used to assess progress against the planned schedule and budget.

Common test metrics include:

- % of work done in test case preparation
- Test case execution (e.g. number of tests run/not run)
- Test coverage of requirements, risks or code.
- Subjective confidence of testers in the product.
- Defect information (e.g. defect density, defects found and fixed).
- % of work done in test environment preparation.
- Dates of test milestones.
- Testing costs, including the cost compared to the benefit of finding the next defect or to run the next test.

Test reporting

Test reporting is concerned with summarizing information about the testing endeavor, including:

- What happened during a period of testing (ex: dates when exit criteria were met)
- Analyzed metrics to support decisions about future actions (ex: the economic benefit of continued testing)

Metrics are collected at the end of a test level in order to assess

- The adequacy of the test objectives for that test level.
- The adequacy of the test approaches taken.

- The effectiveness of the testing with respect to its objectives.

Test control

Test control describes any guiding or corrective actions taken as a result of information and metrics gathered and reported.

Examples of test control actions are:

- Making decisions based on information from test monitoring
- Re-prioritize tests when an identified risk occurs (eg. software delivered late)
- Change the test schedule due to availability of a test environment
- Set an entry criterion requiring fixes to have been retested (confirmation tested) by a developer before accepting them into a build

Configuration management

LO: Explain why configuration management is necessary in software development and testing
LO: Enumerate software artifacts that need to be under configuration management

Configuration management

The purpose of configuration management is to establish and maintain the integrity of the products of the software through the project and product life cycle.

- Components
- Data
- Documentation

For testing, configuration management may involve ensuring that:

- All items of testware are
 - identified
 - version controlled
 - tracked for changes
 so that traceability can be maintained throughout the test process.
- All identified documents and software items are referenced unambiguously in test documentation.

For the tester, configuration management helps to uniquely identify (and to reproduce)

- the tested item
- test documents
- the tests
- the test harness

Risk and testing

LO: Define and explain the concept of risk. Describe how risk is calculated
LO: Describe the differences between project risks and product risks

Risk and testing

Risk = (def.) the chance of an event, hazard, threat or situation occurring and its undesirable consequences, a potential problem.

The level of risk is determined by:

- The likelihood of an adverse event happening
- The impact (the harm resulting from that event)

Project risks

When analyzing, managing and mitigating these risks, the test manager is following well established project management principles. (see IEEE 829).

Project risks = the risks that surround the project's capability to deliver its objectives, such as:

Organizational factors

- skill and staff shortages
- personal and training issues
- political issues (i.e. problems with testers communicating their needs and test results)
- improper attitude toward testing (i.e. not appreciating the value of finding defects during testing)

Technical issues

- problems in defining the right requirements
- the extent that requirements can be met given existing constraints
- the quality of the design, code and tests

Supplier issues

- failure of a third party
- contractual issues.

Product risks

Product risks = Potential failure areas in the software.

They are a risk to the quality of the product, i.e:

- Failure-prone software delivered.
- Software/hardware could cause harm to an individual or company.

- Poor software characteristics (e.g. functionality, reliability, usability and performance).
- Software that does not perform its intended functions.

Risks are used to decide where to start testing and where to test more

Testing is used to

- reduce the risk of an adverse effect occurring
- reduce the impact of an adverse effect.

In a risk-based approach the risks identified may be used to:

- Determine the test techniques to be employed.
- Determine the extent of testing to be carried out.
- Prioritize testing in an attempt to find the critical defects as early as possible.
- Determine whether any non-testing activities could be employed to reduce risk (e.g. providing training to inexperienced designers).

Incident management

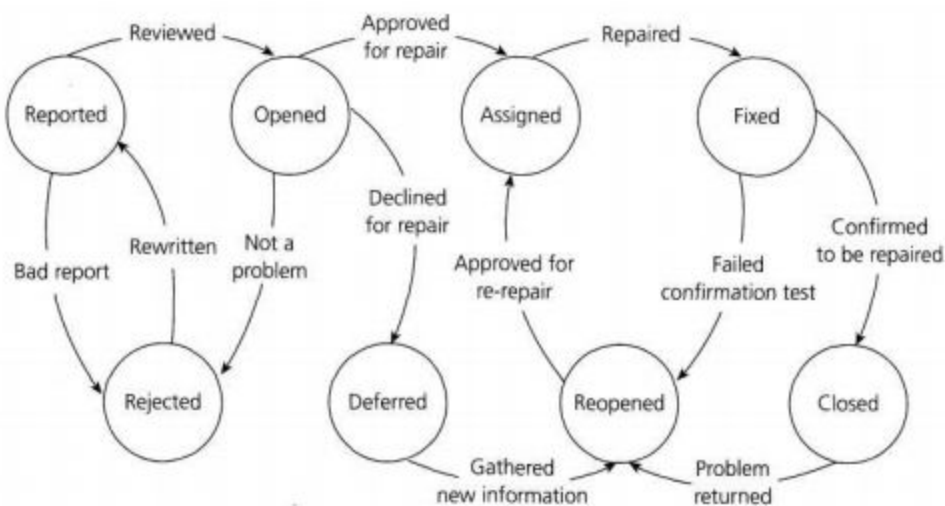
LO: Describe the content of a typical incident report

LO: Write an incident report of a bug you have discovered in a software product

Incident = (Def.) discrepancies between actual and expected test outcomes.

When to raise incidents: during development, review, testing or use of a software product.

Statuses of incident reports:



Objectives of incident reports:

- Provide developers and other parties with feedback about the problem to enable identification, isolation and correction as necessary.
- Provide test leaders a means of tracking the quality of the system under test and the progress of the testing.
- Provide ideas for test process improvement.

Incident reports

Details of the incident report may include (cf. IEEE 829):

Date: _____

Project: _____

Programmer: _____ Tester: _____

Program/Module: _____ Build/Revision/Release: _____

Software Environment: _____ Hardware Environment: _____

Status of the incident _____

Number of Occurrences: _____ Severity: _____ Impact _____ Priority _____

Detailed Description: _____ (logs, databases, screenshots)

Expected result / Actual result: _____

Change history _____

References (including the identity of the test case specification that revealed the problem) _____

Assigned To: _____

Incident Resolution: _____

Tool support for testing

Types of test tools

LO: Classify different types of test tools according to their purpose and to the activities in the test process and in the software life-cycle
LO: Explain the term “test tool”
LO: For each type of test tool, explain how it supports testing

Tool support for testing – types of tools

Test tools can be used for one or more activities that support testing

- Tools that are directly used in testing (e.g.: test execution tools, test data generation tools, result comparison tools)
- Tools that help in managing the testing process (ie: test results, requirements, incidents, defects) and for monitoring and reporting the test execution
- Tools that are used in exploration (e.g. tools that monitor the file activity for an application)
- Any tool that aids in testing

Tool support for testing - purposes

Tools support for testing can have one or more of the following purposes, depending on the context:

- improve the efficiency of the test activities (e.g.: by automating repetitive tasks)
- automate activities that require significant resources when done manually (e.g.: static testing)
- automate activities that cannot be done manually (e.g. largescale performance testing of client-server applications)
- increase reliability of testing (by automating large data comparisons or simulating complex behavior)

Test tool classification

Tools are classified in this course according to the testing activities that they support.

- one activity
- more than one activity, but classification falls under the main activity

Testing tools can improve:

Efficiency of testing by automating repetitive tasks.

Reliability of testing by automating large data comparisons or simulating behavior.

Notes

Some types of test tool can be intrusive - the tool itself can affect the outcome of the test. (i.e. timing measurements may be different depending on how you measure it with different performance tools). The consequence of intrusive tools is called the probe effect.

Some tools offer support more appropriate for developers. Such tools are marked with “(D)” in this chapter.

Tools support for management of testing & tests

Characteristics:

- Support for the management of tests and the testing activities.
- Interfaces to

- test execution tools
 - defect tracking tools
 - requirement management tools
- Support for traceability of tests, test results and incidents to source documents, such as requirements specifications
- Generation of progress reports
- Logging test results
- Offer info on metrics related to the tests

Requirements management tools:

- store requirements
- check for consistency and undefined (missing) requirements
- allow prioritization
- enable individual tests to be traceable to requirements

Incident management tools:

- store and manage incident reports
- support management of incident reports

Configuration management tools:

- are necessary to keep track of different versions and builds of the SW and tests
- are particularly useful when developing on more than one configuration of the HW/SW environment

Requirements management tools

- Enterprise architect
- nets

Incident management tools

- Jira
- Youtrack
- Bugzilla
- HP Quality Center
- ClearQuest
- Mantis
- Visual Studio

Configuration management tools

- Ansible
- ViretualBox

Tools support for static testing

Review tools

- store information about review processes
- store and communicate review comments, report on defects and effort

They can provide aid for online reviews, which is useful if the team is geographically dispersed.

Static analysis tools (D)

- Major purpose
 - The enforcement of coding standards
 - The analysis of structures and dependencies (e.g. linked webpages)
 - Aiding in understanding the code
- support developers, testers and quality assurers in finding defects before dynamic testing.

Static analysis tools can calculate metrics from the code (e.g. complexity), which can give valuable information for planning or risk analysis.

Modeling tools (D)

Validate models of the software.

The major benefit of static analysis tools and modeling tools is the cost effectiveness of finding more defects at an earlier time in the development process. As a result, the development process may accelerate and improve by having less rework.

Review tools

- VDOC
- gerrit
- Review Board

Static analysis tools (D)

- Code Sonar
- Coverity integrity manager
- Visual Studio

Modeling tools (D)

- Star UML
- Umbrello UML Modeller

Tools support for test specification

Test design tools:

- Generate test inputs or executable tests
 - from requirements
 - from a graphical user interface
 - from design models (state, data or object)
 - from code
- This type of tool may generate expected outcomes as well (i.e. may use a test oracle)
- They can save valuable time and provide increased thoroughness of testing because of the completeness of the tests that the tool can generate.

Test data preparation tools:

- Manipulate databases or files to set up test data to be used during the execution of tests
- Benefit: they ensure that live data in a test environment is made anonymous, for data protection.

Test design tools:

- ALLPAIRS

Test data preparation tools:

- DTM data generator
- Data rule

Tools support for test execution & logging

Test execution tools:

- Enable tests to be executed automatically using stored inputs & expected outcomes
- The scripting language allows to manipulate the tests with little effort(i.e. repeat the test with other data)
- Can also be used to record tests(capture & playback tools)

Test harness / unit test framework tools (D):

- Facilitate the test of components of a system - simulating the environment in which that test object will run.
- They may be called unit test tools when they have a particular focus on the component test level.

Test comparators:

- Determine differences between files, databases or test results
- Test execution tools include dynamic comparators, but post-execution comparison may be done by a separate comparison tool.
- A test comparator may use a test oracle, especially if it is automated.

Coverage measurement tools (D):

- Can be intrusive or non intrusive (depends on the measurement technique used)
- Measure the percentage of specific types of code structure that have been exercised

Security tools:

- Search for specific vulnerabilities of the system

Examples

Test execution tools:

- UltimateTestCase
- Selenium
- KeePassTestSuite
- QuickTest Professional
- Microsoft Visual Studio

Test harness/unit test framework tools (D):

- Mock

Test comparators:

- Munit

Coverage measurement tools (D):

- NCover

Security tools:

Tools support for performance & monitoring

Dynamic analysis tools (D):

- Find defects that appear only when software is executing (i.e. memory leaks)
- They are typically used in component and component integration testing

Performance testing/load testing/stress testing tools:

- Monitor and report on how a system behaves under a variety of simulated usage conditions.
- They simulate a load on
 - an application
 - a database
 - a system environment.
- They are often based on automated repetitive execution of tests, controlled by parameters.

Monitoring tools:

- Are not strictly testing tools but provide information that can be used for testing purposes.
- Analyze, verify and report on usage of specific system resources, and give warnings of possible service problems.

Examples

Dynamic analysis tools (D):

- Windows Task Manager

Performance testing/load testing/stress testing tools:

- HP LoadRunner

- Apache JMeter

Monitoring tools:

- Performance monitor

Tools support for specific application areas

There are tools specialized for use in a particular type of application.

Example

- performance testing tools specifically for web-based applications
- dynamic analysis tools specifically for testing security aspects.

Example of targeted areas

- embedded systems

Tools support using other tools

This is not a complete list of tools of this category.

Testers may use:

- word processor
- spreadsheets

as a testing tool, but they are often used to store:

- test designs
- test scripts
- test data.

Testers may also use SQL to set up and query databases containing test data.

Tools used by developers when debugging, to help localize defects and check their fixes, are also testing tools.

It is a good idea to look at any type of tool available to you for ways it could be used to help support any of the testing activities.

Effective use of test tools: potential benefits and risks

LO: Summarize the potential benefits of using test tools in the software lifecycle
LO: Summarize the potential risks of using test tools in the software life-cycle
LO: Summarize the potential risks of test automation in the software life-cycle
LO: Remember the special considerations for test execution tools, static analysis tools and

Potential benefits and risks

Simply purchasing or leasing a tool does not guarantee success with that tool.

Each type of tool may require additional effort to achieve real and lasting benefits.

Potential benefits of using tools

- Reduced repetitive work (running regression tests, re-entering the same test data. Etc)
- Greater consistency and repeatability (tests executed by a tool, tests derived from requirements).
- Objective assessment (static measures, coverage).
- Ease of access to information about tests or testing (statistics / graphs about test progress, incident rates, performance)

Potential risks of using tools

- Unrealistic expectations for the tool (functionality & ease of use).
- Underestimating time, cost and effort for the introduction of a tool (training, external expertise).
- Underestimating the time and effort needed to achieve significant and continuing benefits from the tool
- Underestimating the effort required to maintain the test assets generated by the tool. Over-reliance on the tool (replacement where manual testing would be better)

Special considerations

1. Test execution tools

- This type of tool often requires significant effort in order to achieve significant benefits.
- Capturing tests by recording the actions of a manual tester seems attractive, but this approach does not scale to large numbers of automated tests. This type of script may be unstable when unexpected events occur.
 - Data-driven approach: separates out the test inputs (the data) and uses a more generic script that can read the test data and perform the same test with different data.
- In a keyword-driven approach: the spreadsheet contains keywords with the actions to be taken (also called action words), and test data. Testers can then define tests using the keywords.

2. Performance testing tools

- These tools need tester with expertise in performance testing to design the tests and interpret results.

3. Static analysis tools

- These tools applied to source code can enforce coding standards, but if applied to existing code may generate a lot of messages
 - A gradual implementation with initial filters to exclude some messages would be an effective approach.
4. Test management tools
- They need to interface with other tools or spreadsheets in order to produce information in the best format for the current needs of the organization.
 - The reports need to be designed and monitored so that they provide benefit.

Introducing a test tool to an organization

LO: State the main considerations for introducing a new test tool to an organization
LO: State the goals of a proof-of-concept for a test tool, with the scope of evaluation and pilot
LO: Explain the success factors for the deployment of a new test tool into an organization

Introducing a tool into an organization

The main considerations in selecting a tool for an organization include

- Assess the organizational maturity, strengths and weaknesses
- Evaluate against clear requirements and objective criteria.
- A proof-of-concept to test the required functionality and determine whether the product meets its objectives.
- Evaluation of the vendor (including training, support and commercial aspects).
- Identification of internal requirements for coaching and mentoring in the use of the tool.

Introducing the selected tool into an organization starts with a pilot project, with the following objectives

- Learn more detail about the tool
- Evaluate how the tool fits with existing processes and practices, and determine what would need to change
- Decide on standard ways of using and maintaining the tool and the test
- Assess whether the benefits will be achieved at reasonable cost

Success factors for the deployment of the tool within an organization include

- Roll out the tool to the rest of the organization incrementally
- Adapt and improve processes to fit with the use of the tool
- Provide training and coaching/mentoring for new users
- Define usage guidelines
- Implement a way to learn lessons from tool use
- Monitor the tool use and benefits

Usability Testing

HCI – definition and purpose

LO: Recall the different names and the definition of HCI
LO: Describe the purpose of HCI with regards to software systems

HCI – definition and purpose

The many names and meanings of HCI:

- Human-computer interaction
- Usability
- User friendliness
- Interaction design (IxD)
- User experience (UX)

Definition:

”The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.” (ISO 9241-11)

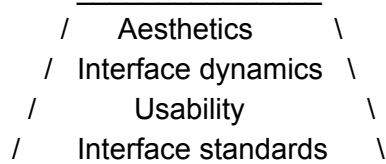
The purpose of HCI testing is to make a software system:

- Understandable
- Easy to learn
- Effective to use
- Easy to remember
- Satisfactory to use

HCI framework

LO: Enumerate and describe the elements of the HCI framework
LO: Provide examples of what they include the following HCI framework elements: interface standards, usability, interface dynamics, Aesthetics

!!!Such pyramide!!!!



Interface standards

Interface standards are constituted by:

- Best practices
- Consistent behavior and design
- Decrease work load
- Faster development

These standards have to be followed for both user interfaces and also APIs.

Usability

Usability means:

- Effectiveness
- Efficiency
- Satisfaction

Note: The key is to understand the target users and their needs and create a user-centric design.

We will detail this part in the next chapter.

Interface dynamics

An interface (whether visual or API) has to be designed in such way that it is:

- Responsive and fast
- Adaptable to the users needs and context
- Empowering for the user
- Captivating
- Dynamic

Aesthetics

- Responsible for the first impression
- Modern, fresh, appealing design
- Recognition of a company's applications
- A company's graphical profile

User-centric design process

LO: Enumerate and explain the steps in the user-centric design process
LO: Describe with examples the user-centric design process of a software system
LO: Explain the notion and purpose of low-fidelity prototyping
LO: Give reasons why we use personas in the user-centric design process
LO: Describe how to make user observations
LO: Describe how to evaluate a design concept (user testing)

Synopsis

When designing a user-centric software systems, one has to

- Understand how the users think and behave
- Gather fact and data instead of rely on opinion or speculation
- Perform studies, design and test on users before implementation
- Iterate!

There are many different methods available to help in the process.

- Understand user context
 - Interview
 - Survey
 - User observation
 - Competitive analysis
 - Expert evaluation
 - Domain knowledge / experience
 - Support statistics
- Specify demands
 - Personas
 - User scenarios and cases
 - Prioritized list of requirements
- Develop concept
 - Card sorting
 - Mind map
 - Low-fidelity prototyping
 - High-fidelity prototyping
- Evaluate concept
 - User tests

- Expert evaluation
- Comparative tests

User observation - understand user context

Most important step in the process

- Who are the users?
- In what environment/situation do they use the product?
- What goals shall the product help the user to achieve?
- What demands must the application fulfil to satisfy the user?

Purpose of user observation

It studies user behavior

Identifies 60-80% of usability problems with 3-5 users

Note: Just observe what are they doing – do not tell them what to do

How to perform user observation

It's hard to convince the users to participate in an observation.

Therefore, the user needs to feel safe, not criticized.

Prior to the observation:

- Contact the user
 - Ex: email, phone. Express the intent of the observation, what would be the user's role, what would be your role
- Send out background questions about the user
 - Ex: what are your daily tasks? Which of your tasks do you perform in the ?
 - How often do you use the ?
 - How long have you used the ?
 - Do you have background education on how to use the software?
- Prepare observation guide, focus areas
 - Login
 - Do you find everything you are looking for in the ?
 - Ask the user to find a specific item
 - Ask the user to add / remove a specific item
 - Design a workflow that you would like the user to go through (searches, filters, approvals, rollbacks)

How to perform user observation

During the observation:

- Visit user's workplace

- Use think-aloud techniques (ask the user to describe what he is trying to do)
- Document with notes and video recorders

User observation tools

Silverback(Mac)

Techsmith Morae(Windows)

- Recorder
 - Displays tasks in screen
 - Records screen interaction
 - Records user video and audio
 - Mouse clicks
- Observer
 - Connect observers
 - Possibility to make notes
 - Chat between observers
- Manager: Analysis
 - Gathers recorder and observer data.
 - Measures time, mouse clicks per task.

User observation tips

- Start looking for users quite long in advance.
- Be realistic about how many observations you have time for.
- Give the user time to perform the task. Do not interrupt or show them how to do something.
- Make it a positive experience for the user: you are not testing users, you are just discovering problems in the software, with the users help.
- Collect general impressions at the end.
- Start looking for users quite long in advance.
- Be realistic about how many observations you have time for.
- Give the user time to perform the task. Do not interrupt or show them how to do something.
- Make it a positive experience for the user: you are not testing users, you are just discovering problems in the software, with the users help.
- Collect general impressions at the end.

Specify demands

Most important steps of this process:

- Document the problems
- Summarize, analyze and compare the results
- Create a basis for the conceptualizing step

Specify demands - personas

Personas:

- Are a design and specification tool.
- A description of a representative user (a pretended person who represents our user).
- Tell us who are the users, their goals, activities, motivations to use our products.
- Helps us define how can we make their work routines more effectively.

Personas =

- Informed based on research
- Check and validate your assumptions.

User profiles =

- What we believe is true about our users

Specify demands – how to identify personas

Find patterns and identify personas, based on:

- statistics
- workshops, interviews and user studies
- surveys

Specify demands – how to create personas

Background

- Name
- Age
- Job title
- Education
- Experience

Characteristics

- What type of person is he/she? Detailed, casual...

Skills

- Technical, analytical, worked online...

Goals/ Motivations

- Why does he/she need this product?
Savings, accurate accounting...

Pains

- What are the key things the usability engineer needs to consider?


Tasks

- What are he/she main work tasks?






Specify demands – example of personas

Shen – “The follower”

PERSONA 1.




MARY JACOBS



It's sentimental, the data matters more than the monetary value. If I lost my diary and phone numbers that would huge.

Job Title: MSc Student
Age Group: 16-24
Phone: Nokia 2600

Mary loves taking photos of friends and family and is the kind of person who brings a camera along to the pub. Mary's requirements of her phone are changing. She is just about to finish her MSc and is looking for a phone that offers everything on the Nokia 2600 but internet and email as well. She would describe herself as clumsy and loves her old Nokia as it bounces! She's good with technology but worries that she will drop and break it. Mary loves keeping fit and runs up to five times a week and enjoys yoga too. Keeping in contact with friends and family back in Africa is very important.



User Requirements:

- Durability
- Organised
- Alarms
- Calendar
- Easy Access to Train Times
- Email
- Large Keys
- Reliable Data Storage
- Good Communication
- Image Quality

e' phone, I'm always one or two

ione primarily to keep in contact with
joys using it to take photos of his
gh he would like to own a smart phone
or emails and the internet, they are
ensive for his budget.

p his phone until he's eligible for a
ide, although, if given the option, he
ore often for the novelty of having a

sed at how quickly technology is
lways interested to see new the newest
arket.

ics

d before renewing his phone
cious; thinks twice before buying
/ new phones
fer upgrading if a life event called for

d reason to spend money on a phone
egapixels on a camera
nt to be embarrassed to pull out his
lic

d of his contract so that he can get a
rade
deal that he can, taking into account the
l the handset
igh-end phone when he can justify the
t



Specify demands – how to use the personas

Make them visible throughout the development and marketing teams.

- Ask stakeholders to communicate the user's needs through personas.
- Refer to the personas when creating and testing requirements, specifications and stating assumptions. (How would Hanna use it?)
- When creating a new functionality, test if it's in line with the persona's needs.

Develop concept

This is the creative step of the process:

- Explore new ideas
- Two types of prototyping

- Low-fidelity
- High-fidelity

Develop concept – Low-fidelity prototyping – why?

Low-fi prototyping is:

- Fast (quick progress on requirements specification and testing)
- Easy (anyone can do this)
- Cheap (it takes a pen and a paper)
- Sketchy
- Relevant

Note: Low-fidelity prototypes are also called wireframes or mock-ups

Develop concept – Low-fidelity prototyping –how to apply it

- Always start on paper – paper prototyping
- Focus on structure and function, no details
- Do not apply any design
- Create multiple concepts
- Evaluate (to get feedback), refine and narrow down

Note: it's not supposed to be pretty!

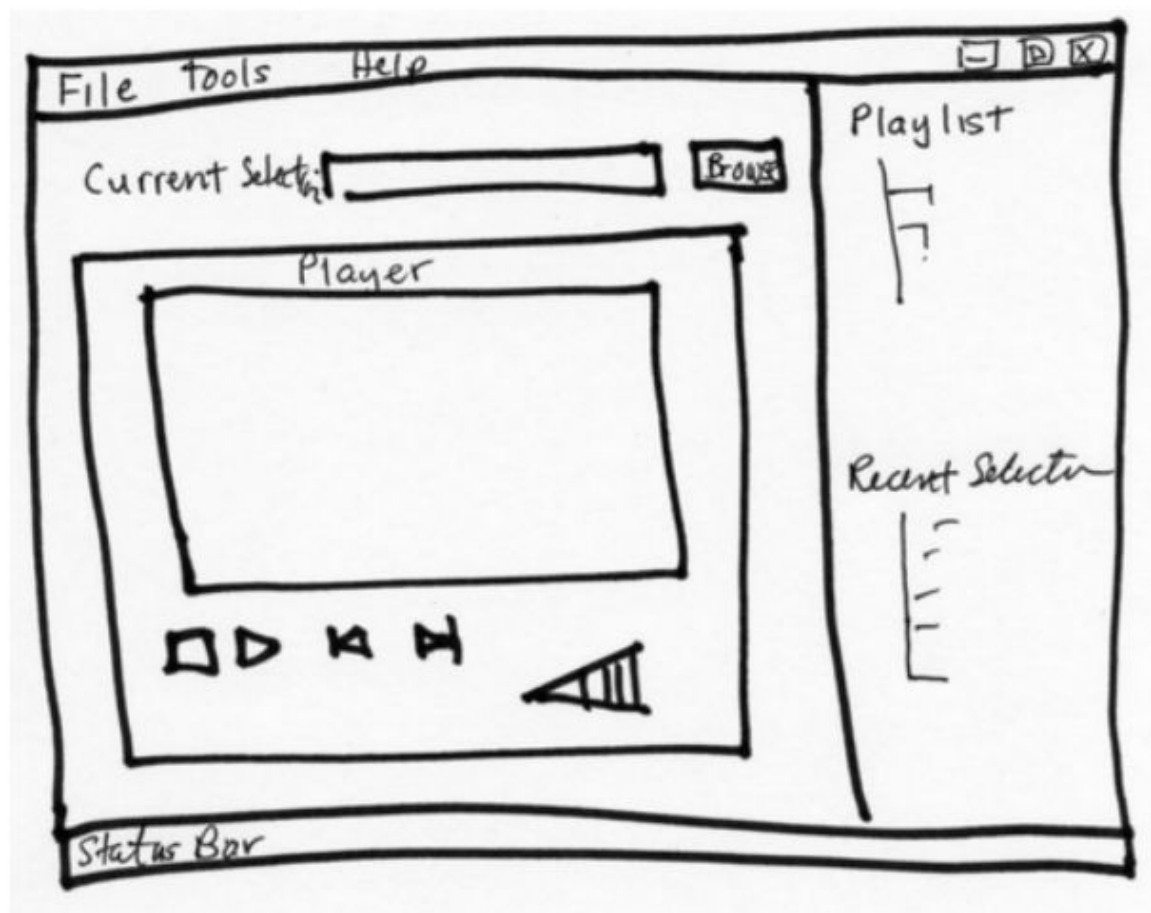
Develop concept – Low-fidelity prototyping – tools

Examples are:

- Photoshop
- PowerPoint
- Visio
- Balsamiq

Note: use the tool you are comfortable with

Develop concept – Low-fidelity prototyping – example



Evaluate concept

- Evaluate against demands
- Find areas of improvement
- Iterate!

Evaluate concept – user testing

The user testing is the evaluation of a product by the users.

You will present the product to real users with interactive prototypes.

The goal is to discover errors and areas of improvement.

Evaluate concept – types of user testing

- Face to face
- Remote
- Test labs

Evaluate concept – prepare for user testing

- Decide what will you use in the testing:
 - Paper prototypes
 - Software prototype
 - A beta version of software
- Set goals for the user testing: what do I want users to accomplish? (learn, remember the software, discover new features etc).
- Plan the test tasks based on real problems to be solved or workflows to be followed. - Select a user group to test at a time.
- Choose between alfa-testing or beta-testing (at your company's site or at user's site).

Note: it's a good idea that 1 person leads the test and 1 person documents the users' answers.

Evaluate concept – conduct the user test

- Welcome and introduce user to the test
- Provide one task at the time
- Use think-aloud technique
- Document with notes and record the screen
- Do not help the user during the test
 - Ex: Q: "Can I do it like this?" A: "You can do all you want. Try to find out."

HCI guidelines

LO: Recall the purpose of the HCI guidelines
LO: List and explain the basic usability elements, as defined by the HCI guidelines
LO: Explain and compare the guidelines provided for the different types of system messages (info, warning, error, question)

Purpose of guidelines

Gathered interface standards - Windows, OS X and web guidelines such as W3C

They give the developers and testers a set of best practices, which in their turn provide:

- Consistent behavior and design of software
- Decrease people's work load
- Faster development cycles

Guidelines - Usability elements

The following are the basic usability elements that the guidelines refer to. The tester has to compare the following elements against the specifications of the provided guidelines:

- Workflows
- Navigation
- Search and filter
- Grids
- Flow on page (top-left to bottom right, in the western cultures) -
- Placement of important buttons, like Save and Cancel (Windows / OX standard)
- Tab order (has to follow the reading order; columns over rows; within each group box)
- Alignment
- Grouping (level of boxes, use and overuse)
- Active / inactive elements (the inactive elements must always be disabled)
- Destructive actions (never put Save and Delete next to each others; have confirmation for destructive actions)
- Placement of checkboxes and radio buttons (Win / OX standard)

System messages

System messages, the way they behave and the way they must be designed and used are the second category included in the HCI guidelines.

Error and warning messages must explain the (potential) problem and provide a solution

Never use error codes, jargon or technical terms – speak the users language

Never use capital letters or exclamation mark – you scream at the user

Choosing the right type of message:

Error – alerts the user of a problem that has occurred

Warning – to make the user aware of a potential problem

Information – inform the user

Question – requesting a response from the user

Note:

- Keep the message short, people do not read!
- Use the right action buttons:
 - Errors and warnings are never OK, use Close
 - To have Yes, No and Cancel for a question is confusing

Accessibility Testing

The context of accessibility

LO: Define the notion of accessibility
LO: Explain what are the barriers in using software that the accessible design tries to solve
LO: Contrast between HCI and Accessibility
LO: .List the reasons why Accessibility isn't more included in the HCI guidelines

Definition of accessibility

Disability:

The outcome of the interaction between a person and the environmental and attitudinal barriers they may face. (World Health Organization, International Classification of Functioning (ICF))

Usability:

The effectiveness, efficiency and satisfaction with which a specified set of users can achieve a specified set of tasks in a particular environment (ISO 9241-11)

Accessibility:

The usability of a product, service environment or facility by the people with the widest range of capabilities. (ISO 9241-20) (Accessibility is the degree to which a product, device, service, or environment is available to as many people as possible.)

Barriers

What problems will stop someone from being able to use a software product?

Barrier priority	What it covers
Critical	Barriers that stop someone from using a software product or some of its features successfully
Serious	Problems that cause frustration, slow someone down or require work-arounds
Annoying (moderate)	Things that are frustrating, but won't stop someone from using the site
Noisy (minor)	Minor issues that might cause someone a problem, but which mainly damage credibility

Why is accessibility not considered more in the UX work

- Invisible
- Hidden
- Misunderstood

International accessibility legislation

The purpose is to offer equal access to social, political, and economic life which includes not only physical access but access to the same tools, services, organizations and facilities for which everyone pays (e.g., museums).

UN: Article 9 of the United Nations Convention on the Rights of Persons with Disabilities commits signatories to provide for full accessibility in their countries (all 192 member-countries).

EU: The European Union which has signed the United Nations' Convention on the Rights of Persons with Disabilities, also has adopted a European Disability Strategy for 2010-20. The Strategy includes the following goals, among others:

- ensuring the European Platform Against Poverty includes a special focus on people with disabilities
- working towards the recognition of disability cards throughout the EU to ensure equal treatment when working, living or travelling in the bloc
- developing accessibility standards for voting premises and campaign material
- taking the rights of people with disabilities into account in external development programs and for EU candidate countries

Accessibility personas

LO: Explain the role of the personas in the study of accessibility
LO: List the main types of personas used in the accessibility studies

The role of personas in the accessibility study

The personas can help address big challenges in approaching the usability issues:

- give a realistic view of the people we design for
- help taking different users into account (will tell a story we can relate to)
- help organizing increasing amounts of data; will document our assumptions
- build consensus around a clear, consistent view on accessibility needs to be solved

Autism spectrum disorder



Trevor, 18

"I like consistent, familiar places on the web"

- Lives with family
- Goes to secondary school
- Computers at school; laptop at home, basic mobile phone with SMS

Characteristic	Uses larger text and a program who hides everything but the text, so he doesn't get distracted
Aptitude	Uses the computer well for games, but doesn't learn new sites easily
Attitude	Prefers familiar sites in an established routine
Assistive technology	Text preference settings, power keyboard user

Cerebral palsy



Emily, 24

"I want to do everything for myself"

- High school graduate, working on a college degree
- Lives independently in a small apartment
- Works part-time at a local community center

Characteristic	Difficult to use hands and has some difficulty speaking clearly; uses a motorized wheel chair
Aptitude	Uses the computer well, with the right input device, good at finding efficient search terms
Attitude	Wants to do everything for herself; can be impatient
Assistive technology	Communicator with speech generator, iPad, power wheelchair

Blindness with some light perception



Jacob, 32

“The right technology lets me do anything”

- College graduate, legal training courses
- Paralegal, reviews and writes cases
- Shares apartment with a friend
- Laptop, braille display, smartphone

Characteristic	Blind since birth with some light perception
Aptitude	Skilled technology user
Attitude	Digital native, early adopter, persists until he gets it
Assistive technology	Screen reader, audio note-taker, Braille display

Fibromyalgia (fatigue)



Lea, 35

“No one gets that this really is a disability”

- Masters degree
- Writes for a trade publication
- Works from home
- Computer, tablet

Characteristic	Fatigue, more pronounced when using trackballs and keyboards
Aptitude	Average user
Attitude	Wishes people would understand how hard it can be for her to make it through the day
Assistive technology	Split keyboard, speech recognition software

Deaf-mute



Steven, 38

“My only disability is that not everyone can sign”

- Art school
- Graphic designer in a small ad agency
- Computers, laptops, tablets, smartphones

Characteristic	Native in ASL (American sign language)
Aptitude	Good with graphic tools
Attitude	Can be annoyed about accessibility, like lack of captions
Assistive technology	Sign language, CART (communication access real-time translation), captions, video-chat

Visual impairment

(6) Visual impairment



Vishnu, 48

"I want to be on the same level as everyone else"

- Born in India, finished school in Malaysia, lives in Singapore
- Engineering degree, speaks 3 languages
- Works for a medical software company on international projects
- High-tech dedicated devices at work, two mobile phones and a laptop

Characteristic	Uses contrast adjustment to see the screen clearly
Aptitude	Expert user of technical tools
Attitude	Sees himself as a world citizen and wants to be able to use any site
Assistive technology	Contrast adjustment, screen magnification, personalized style sheets

ARMD (age-related macular degeneration)




Carol, 74

“My grandkids are dragging me in the world of technology”

- Retired, worked as a bookkeeper
- Lives alone in an apartment
- Older computer at home, basic mobile phone

Characteristic	Cannot see so well Hearing aid Doesn't have any special AT on the computer
Aptitude	Used computers when she worked ad a bookkeeper, but now her grandkids keep her old home-computer updated
Attitude	Willing to learn
Assistive technology	Enlarges text

Non-English speaker

 <p>Maria</p>	
Maria, 49 “I love the internet. It’s all here... when I can find it”	
<ul style="list-style-type: none">- High-school graduate- Married, grown children- Community health worker- Smartphone from her work, home computer – mainly used for her husband's work	
Characteristic	Needs to use sites in Spanish (when she can find them) Needs computer instructions written clearly
Aptitude	Curious but not very proficient Husband and daughter set-up bookmarks for her.
Attitude	Thinks it’s nice to be able to have her favorite websites with her at all times
Assistive technology	Skype, online translation sites

Accessible design

LO: List and describe the characteristics of accessible design of software

Accessible design

Has to comply with the following:

1. People first: design for differences
2. Solid structure: built to standards
3. Easy interaction: everything works
4. Helpful way-finding: guide the users
5. Clear presentation: supports meaning
6. Plain language: easy to understand
7. Accessible media: supports all senses

Web-content accessibility guidelines

LO: List and explain the four principles in the web accessibility guidelines
LO: Enumerate and explain the characteristics that make web-content perceivable
LO: Enumerate and explain the characteristics that make web-content operable
LO: Enumerate and explain the characteristics that make web-content understandable
LO: Enumerate and explain the characteristics that make web-content robust

Introduction

Web Content Accessibility Guidelines (WCAG) 2.0 covers a wide range of recommendations for making Web content more accessible.

- Principles - At the top are four principles that provide the foundation for web accessibility: perceivable, operable, understandable, and robust.
- Guidelines - Under the principles are guidelines. The 12 guidelines provide the basic goals to make content more accessible to users with different disabilities.
- The guidelines are not testable, but provide the framework and overall objectives to help understand the success criteria and better implement the techniques.
- Success Criteria - For each guideline, testable success criteria are provided to allow WCAG 2.0 to be used.
- Sufficient and Advisory Techniques - For each of the guidelines and success criteria there is a list of test techniques. The techniques are informative and fall into two categories: those that are sufficient for meeting the success criteria and those that are advisory.

Principle 1 - Perceivable

Perceivable - Information and user interface components must be presentable to users in ways they can perceive it.

Guidelines:

- Text Alternatives
Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.
- Time-based Media
Provide alternatives for time-based media.

- **Adaptable**
Create content that can be presented in different ways (for example simpler layout) without losing information or structure.
- **Distinguishable**
Make it easier for users to see and hear content including separating foreground from background.

Advisory techniques for development and testing:

- Identify informative non-text content
- Describe images that include text
- Link to textual information that provides comparable information (e.g., for a traffic webcam, a municipality could provide a link to the text traffic report)
- Provide more than two modalities of CAPTCHAs
- Provide a transcript of a live audio-only presentation
- Provide a note saying "No sound is used in this clip" for video-only clips
- Use readable fonts
- Make sure any text in images of text is at least 14 points and has good contrast
- Provide a highly visible highlighting mechanism for links or controls when they receive keyboard focus
- Convey information redundantly using color
- The visual presentation of text and images of text has to have a contrast ratio of at least 4,5:1 (21:1 for black: white)

Principle 2 - Operable

Operable - User interface components and navigation must be operable.

Guidelines:

- **Keyboard Accessible**
Make all functionality available from a keyboard.
- **Enough Time**
Provide users enough time to read and use content.
- **Seizures**
Do not design content in a way that is known to cause seizures.
- **Navigable**
Provide ways to help users navigate, find content, and determine where they are.

Advisory techniques for development and testing:

- Provide keyboard shortcuts to important links and form controls
- Use unique letter combinations to begin each item of a list
- Avoid use of common user-agent keyboard commands for other purposes
- Provide a mechanism to stop all content that blinks within a web page
- Provide the user with a means to stop moving content even if it stops automatically within 5 seconds • Limit the number of links per page
- Provide mechanisms to navigate to different sections of the content of a Web page

- Make links visually distinct
- Highlight search terms
- Provide keyboard access to important links and form controls
- Provide skip links to enhance page navigation
- Provide access keys
- Using the 'live' property to mark live regions

Principle 3 - Understandable

Understandable - Information and the operation of user interface must be understandable.

Guidelines:

- Readable
Make text content readable and understandable.
- Predictable
Make Web pages appear and operate in predictable ways.
- Input Assistance
Help users avoid and correct mistakes.

Advisory techniques for development and testing:

- Make text that is not in the default human language of the web page visually distinct, giving the names of any languages used in foreign passages or phrases
- Provide language markup on proper names to facilitate correct pronunciation by screen readers
- Provide a mechanism for finding definitions for all words in text content
- Provide a mechanism to determine the meaning of each word or phrase in text content
Avoiding unusual foreign words
- Use unique abbreviations in a web page
- Including content summaries in metadata
- Using the clearest and simplest language appropriate for the content
- Using sentences that contain no redundant words, that is, words that do not change the meaning of the sentence
- Using sentences that contain no more than two conjunctions

Principle 4 - Robust

Robust - Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.

Guidelines:

- Compatible
Maximize compatibility with current and future user agents, including assistive technologies.

Principle 4 – Robust (cont)

Advisory techniques for development and testing:

- Provide labels for all form controls that do not have implicit labels
- Avoid deprecated features of W3C technologies
- Do not display content that relies on technologies that are not accessibility-supported when the technology is turned off or not supported.

Assistive technologies and tools

LO: Explain how the assistive technologies can help the people using them

Impairments that affect the access to IT

The following impairments are some of the disabilities that affect communications and technology access, as well as many other life activities:

- communication disorders
- hearing impairments
- visual impairments
- mobility impairments
- a learning disability or impairment in mental functioning

Assistive technologies

Impairment	Assistive technology
Communication impairment	<ul style="list-style-type: none">• Electronic speech synthesizer
Hearing impairment	<ul style="list-style-type: none">• Earphones, headphones, headsets• Real-time closed captioning• Teletypewriter
Mobility impairment	<ul style="list-style-type: none">• Page-turning device• Adaptive keyboards and computer mice (pointing devices such as trackballs, vertical mouse, foot mouse, or programmable pedal)
Physical or mental impairment, learning disability	<ul style="list-style-type: none">• Voice recognition software• Talking textbooks
Visual impairment, learning disability	<ul style="list-style-type: none">• Modified monitor interface,

	magnification devices <ul style="list-style-type: none"> • Reading service, E-text • Braille note-taker • Braille printer • Screen magnifiers • Optical scanner
--	--

Example of testing tools for accessibility

<http://wave.webaim.org/>

<http://validator.w3.org/>

<http://www.paciellogroup.com/resources/wat>

Exploratory Testing

Understanding the system to be tested

LO: Contrast the two definitions of computer programs and explain the purpose of each
LO: Recall and explain the alternative definition of quality. Give examples of quality. Give examples of software quality

Alt. definition of computer programs

A Computer Program = A set of instructions for a computer.

A House = A set of building materials, arranged in the “House” design pattern.

A House = Something for people to live in.

Kaner’s Definition of a Computer Program

A computer program is

- a communication
- among several people
- and computers
- separated over distance and time
- that contains instructions that can be run on a computer.

The purpose of a computer program is to provide value to people

Alt. definition of quality

A computer program is far more than its code

A software product is far more than the instructions for the device

Quality is far more than the absence of errors in the code (how many of you bought a wrong product just because it's bug-free?)

Testing is far more than writing code to assert that other code returns some "correct" result

Testing is an investigation of code, systems, people, and the relationships between them.

Quality is value to some person(s)

If you don't have an understanding and an agreement on what is the mission of your testing, then just applying procedural testing would be pointless.

So know your mission when practicing quality assurance and testing!

Build quality, not only check it

LO: Explain and compare testing vs. checking
LO: Explain why is testing a sapient activity
LO: Evaluate if a test is relevant or not with regards to the customer's expectations
LO: What are the questions to answer in order to build quality in software systems?

Testing vs. checking

Testing is a sapient activity; checking is not. Testing encompasses checking, not the other way round. Checking, in principle, can be done by a machine

Testing against customer's expectations

Make sure:

- You build the right product What's the use of building the perfect vacuum cleaner, when the customer wanted a microwave oven?

- You build it the right way, with the agreed level of quality What's the use of building the perfect microwave oven, when the customer has money just for a simple classic microwave oven?

Testing against customer's expectations

From requirements: "Build me a wrist-watch"

- It has to provide the time of the day, giving the hour, minute and second
- Maybe: display & a way to charge it

How to build quality

Questions to build quality:

- Who are my customers?
- Is the customer the same as the user?
- How would I describe the users?
- What do they use my product for?
- For how long have they been users of my product?
- How important is this feature for my customers?
- Can they live without this other feature?
- What is the time to market?
- How much can I invest in the development of the product?
- How much can my customers can afford to pay for my product?
- What are the user's expectations? What are the customer's expectations?

Testing is boring context-dependent

LO: Provide examples of in different contexts in which software operates
LO: Explain what is means context-dependent testing

Different software products

Because software systems are different: They ask for different focus in testing, they are used in different ways, they pose different risks.

Context-dependent testing

Testing depends on the following:

Supporting technologies:

- New technology
- Old technology
- Someone else's technology

Product interfaces:

- Batch system
- GUI system
- API-only system

Type of project:

- New project
- Ongoing project

Project size:

- Small project
- Huge project

Development lifecycle:

- "Agile" project
- "Traditional" project

Project management:

- Well managed project
- Poorly managed

Type of product:

- Experimental product
- Mission-critical product
- Life-critical product
- Regulated product

Other product characteristic:

- Embedded system
- Real-time system

Even for the same product:

- It's different if it's the first release (more positive testing)
- Or subsequent releases (more performance testing)
- Or a patch (more regression testing & negative testing)

Testing as a sapient activity

LO: Write a series of exploratory tests for a given example

Testing always happens under conditions of uncertainty.

Good testing is done in a way that stands up to scrutiny.

- Without scrutiny, this would be easy to do. You could test badly, but no one would ever know or care.
- What makes testing hard is that someone, perhaps your employer, perhaps the users, or perhaps you yourself, will be judging your work.

Example

How would you test this:

“The system shall operate at an input voltage range of nominal 100 - 250 VAC.”

Poor answer:

“Try it with an input voltage in the range of 100-250.”

Good answer:

- One fundamental question here is what happens when the system receives voltage outside its rated range? Does it shut down? Blow up? Give erroneous results? Post an error message? If it's designed to pause and then recover when the power returns to normal range, what actually happens?
- Is there special gear that can supply voltage of different levels? If so, we could probably use it. What does it cost? What's the risk?
- Brownouts from the mains could be a risk. Run the system with varying output voltages, varying slowly, varying rapidly, as we run other tests.
- What happens if there's a power spike? Does the product need isolation from the mains?
- What happens if the input power drops suddenly and then recovers? Does the product need an uninterruptable power supply? A line conditioner?
- How long can the product run above or below the rated levels?
- What does it mean “to operate” in this context?
- Run the product for a long time at the upper and lower ranges of voltage.
- What happens when we turn the product on or off? Does the product need time to reach some state before it can be used? How long is that?
- What can we observe in the output or in other aspects of the system that might be correlated to variances in the input voltage.

Heuristics

LO: Explain how do heuristics help software testing
LO: Give examples of guidewords in heuristics for software testing
LO: List triggers for heuristics in software testing

Definition of heuristics

Heuristics is about structuring your problem-solving skills

Heuriskein (greek) -> Heuristic

Heuristics (Def.) refers to experience-based techniques for problem solving, learning, and discovery that give a solution which is not guaranteed to be optimal.

Where the exhaustive search is impractical, heuristic methods are used to speed up the process of finding a satisfactory solution via mental shortcuts to ease the cognitive load of making a decision.

Examples of this method include using:

- a rule of thumb
- an educated guess
- an intuitive judgment
- stereotyping
- or common sense

Guideword heuristics

Guideword Heuristics are words or labels that help you access the full spectrum of your knowledge and experience as you analyze something. They're often provided in the form of a taxonomy, a classification system in (typically) a hierarchical structure; or as a checklist.

Try it and see if it works!

• Get it	• Choose where to look	• Read the spec
• Set it up	• See what is in there	• See if the product matches the spec
• Run it	• See what it's not there	• Find problems
• Run it again		• Find relevant problems
Procedures	Coverage	Oracles

1. Model the test space.
2. Determine coverage.
3. Determine oracles.

4. Determine test procedures.
5. Configure the system.
6. Operate the system.
7. Observe the system.
8. Evaluate the test results.
9. Apply a stopping heuristic.
10. Report test results.

Example (Usability elements)

- Navigation
- Search and filter
- Grids
- Flow on page
 - top-left to bottom right - in the western cultures
- Placement of important buttons, like Save and Cancel (Windows / OX standard)
- Tab order
- Alignment
- Grouping
 - level of boxes, use and overuse
- Active / inactive elements
 - the inactive elements must always be disabled
- Destructive actions
 - never put Save and Delete next to each others
 - have confirmation for destructive actions
- Placement of checkboxes and radio buttons (Win / OX standard)

Trigger heuristics

Ideas associated with an event or condition that help you recognize when it may be time to take an action or think a particular way. Like an alarm clock for your mind.

Emotions and feelings can be powerful sources of trigger heuristics.

Impatience → an intolerable delay?

Frustration → a poorly-conceived workflow?

Amusement → a threat to someone's image?

Surprise → inconsistency with expectations?

Confusion → unclear interface? poor testability?

Annoyance → a missing feature?

Boredom → an uninteresting test?

Tiredness → time for a break?

Improve your exploratory testing

LO: Explain what is the main mission of exploratory testing
LO: Recall why is important to have mental models when practicing exploratory testing
LO: Create a check-list that you would use when practicing exploratory testing
LO: Describe what are the main traits of practicing exploratory testing. What does it resemble with?

Mission of testing

The mission of testing is learning

Testers help to defend the value of the product by learning on behalf of the users.

Tester do this by using:

1. Exploration methods (guided heuristics)
2. Creating mental models to represent and test the system
3. Checklists
4. Scripted test procedures

All these require continuous reviews and improvements

Exploration

Testers will explore:

- The features: Move through the application and get familiar with all the controls and features you come across.
- The complexity : Find the five most complex things about the application.
- The configurations find the ways you can change settings in a way that the application retains those settings.
- The user : Imagine five users for the product and the information they would want from the product or the major features they would be interested in.
- The scenarios: Imagine five realistic scenarios for how the users identified in the user tour would use this product.
- The testability : Find all the features you can use as testability features and/ or identify tools you have available that you can use to help in your testing.
- The variability : Look for things you can change in the application - and then you try to change them.
- The interoperability : What does this application interact with?
- The data : Identify the major data elements of the application.

- The structure : Find everything you can about what comprises the physical product (code, interfaces, hardware, files, etc...)

Mental models

Create mental models of the system you are testing. Try to see more than one point of view

The mental models will be focused on one or a combination of the following:

- Program structure
- Data
- Program functions
- Platform
- Business risk
- Technical risk
- Operations

Checklists

Create checklists:

- On requirements
- On design specs / technical specs
- On the test strategy
- On the areas targeted by testing
- On the test methodologies

Test scripting

Script the most important test scenarios.

Define your:

- Basic flow
- Alternative flows
- Exception flows

Represent them as a discrete unit of interaction between a user (human or machine) and the system.

This interaction has to be a single unit of meaningful work, such as Buy a set of e-books or View Account Details (assuming you test Amazon.com).

Practicing exploratory testing

Do You Test Under Time Pressure?

- You have less time than you'd like.
- Faster is better.

- Your plans are often interrupted.
- You are not fully in control of your time.

Test Automation and Test Automation Framework

Introduction and objectives for test automation

LO: Recall the definition of test automation
LO: List the different interfaces for automated testing. Explain what can be tested through each
LO: Explain and compare the advantages and limitations of test automation
LO: List the major success factors in test automation

Test automation - definition

In software testing, test automation is the use of special software to control the setting up of test preconditions, the execution of tests and the comparison of actual outcomes to predicted outcomes.

- A key aspect to notice is that the software used for testing needs to be separate from the system under test (SUT) itself.
- Test automation is expected to help run many test cases consistently and repeatedly on different versions of the SUT.

But test automation is more than a mechanism for running a test suite without human interaction.

Test automation is a process of designing the testware, including the following:

- Software (test tools)
- Documentation (for the tools, for running tests, etc)
- Test cases
- Test environments
- Data necessary for the test activities:
 - implementing automated test cases
 - monitoring and controlling the execution of tests
 - interpreting, reporting and logging the test results

Interfaces for automated tests

The test automation has two general ways for testing a SUT

- **API testing**
Testing through the public interfaces to classes, modules or libraries of the SUT

- **GUI testing**

Testing through the graphical interface of the SUT

Advantages and limitations of test automation

Advantages

- More tests are run
- Tests that cannot be done manually are enabled (realtime, remote, parallel tests)
- Tests can be more complex
- Tests run faster
- Tests are less subject to operator error
- More effective and efficient use of testers
- Improved system reliability
- Improved quality of test

Limitations

- One cannot automate all manual tests
- The automation can only check machineinterpretable results
- The automation can only check results that can be predicted or verified

Major success factors for test automation

The system under test must be designed for testing.

Invariably, the SUT needs some changes to support automated testing or to allow new types of tests to be executed against it. This is an essential part of success with test automation.

- UI testing: the SUT should decouple as much as possible the GUI interaction data from the appearance of the graphical interface.
- API testing: more classes or modules may need to be exposed as public so that they can be tested.

Good Test Automation Architecture (TA-A): The architecture of the test automation is just as the architecture of a software work product, and should be treated as such.

Target the testable parts of the SUT first. Generally, a key factor in the success of the test automation lies in the ease of implementing automated test scenarios.

Creation and maintenance of a test automation framework (TA-F): The framework should be easy to use, well documented, and maintainable.

Approaches for automating test cases

LO: List the main approaches for creating automated test cases
LO: Explain the principles of capture/replay type of testing

LO: Explain the principles of data driven approach in automated testing
LO: Explain the principles of data driven approach in automated testing

Create automated test cases

A number of approaches can be used to create the sequence of actions of an automated test:

- A tool generates automated test procedures from models
- A tool translates an automated test procedure into an automated test script
- The Test Automation Engineer transforms automated test procedures into automated test scripts
- The TA-E implements test cases directly into automated test scripts

Explanation of concepts:

Test cases translate into sequences of actions which are executed against an SUT. That sequence of action can be documented in an automated test procedure and/or can be implemented in an automated test script. **The test script, besides the test procedure, also contains verification steps in it.**

Types of approaches for automating test cases

Well-established approaches for automating test cases include:

- Capture replay
- Linear scripting
- Structured scripting
- Data-driven approach
- Keyword-driven approach
- Process-driven approach
- Model-based approach

Capture/replay

Principle concept:

In capture/replay approaches, we use tools to capture interactions with the SUT. We capture inputs; outputs may also be recorded for later checks.

During the replay of events, there are various manual and automated output checking possibilities:

- Complete: all system outputs that were recorded during capture must be reproduced during replay
- Exact: all system outputs that were recorded during capture must be reproduced during replay to the level of detail of the recording
- Checkpoints: only selected system outputs are checked at certain points for specified values

Pros

- The capture/replay approach can be used for SUTs on GUI and/or API level. Initially, it is easy to setup and use.

Cons

- Capture/replay scripts are hard to maintain and evolve. For example, when recording at the GUI level: a change in the positioning of a GUI element. The test cases based on capture/replay are very fragile.
- Implementation of the test cases (scripts) can only start then the SUT is available.
- When the capture is done via the GUI, a GUI is needed. Not all software implementations have a GUI (e.g., embedded software).

Data-driven approach

Principle concept

The inputs are extracted from the scripts and put it into one or more separate files (typically called data files).

- This means we can reuse the main test script to implement a number of tests (rather than just a single test).
- Typically the 'reusable' main test script is called a control script.
- The control script contains the sequence of instructions necessary to perform the tests but reads the input data from a data file.
- One control test may be used for many tests but it is usually insufficient to automate a wide range of tests.
- Thus, a number of control scripts will be required but that is only a fraction of the number of tests that are automated.

Pros

- The cost of adding new automated tests can be significantly reduced by this scripting technique. This technique is used to automate many variations of a test, giving deeper testing in a specific area rather than increasing coverage generally.

Cons

- A lot of irrelevant test scripts can be generated – it's hard to make those test scripts significant.

Keyword-driven approach

Principle concept

- The keyword-driven scripting technique builds on the data-driven scripting technique. This methodology uses keywords (or action words) to symbolize a functionality to be tested.
- The keywords are separated from data and the execution part of the script.
- Each keyword should represent a sequence of detailed actions that produce some meaningful result. For example, 'create account', 'place order', 'check order status' are

all possible actions for an online shopping application that each involve a number of detailed steps.

Pros

- This method gives the non-technical testers more freedom to specify automated tests without as much dependency on the technical test analysts.
- Once the controlling script and supporting scripts for the keywords have been written, the cost of adding new automated tests will be much reduced.
- These test cases are easier to maintain, read and write as the complexity can be hidden in the keywords.

Cons

- Implementing the keywords remains a big task for Technical Test Analysts, particularly if using a tool that offers no support for this scripting technique.
- For small systems it may be too much of overhead to implement. The costs would outweigh the benefits.

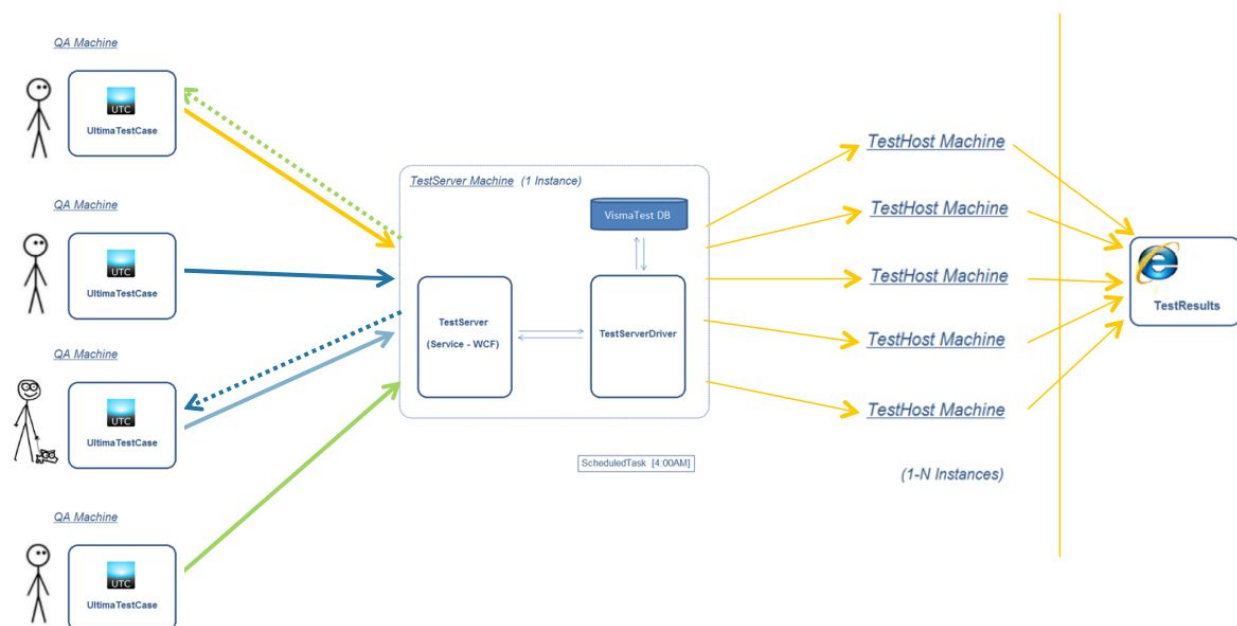
Architecture of a test automation framework

LO: Explain why there is a need to have test automation frameworks instead of simple tools

LO: Design the general architecture of a test automation framework. Explain its layers

LO: Enumerate the characteristics of the following layers in test automation frameworks: test generation, test definition, test execution, test adaptation

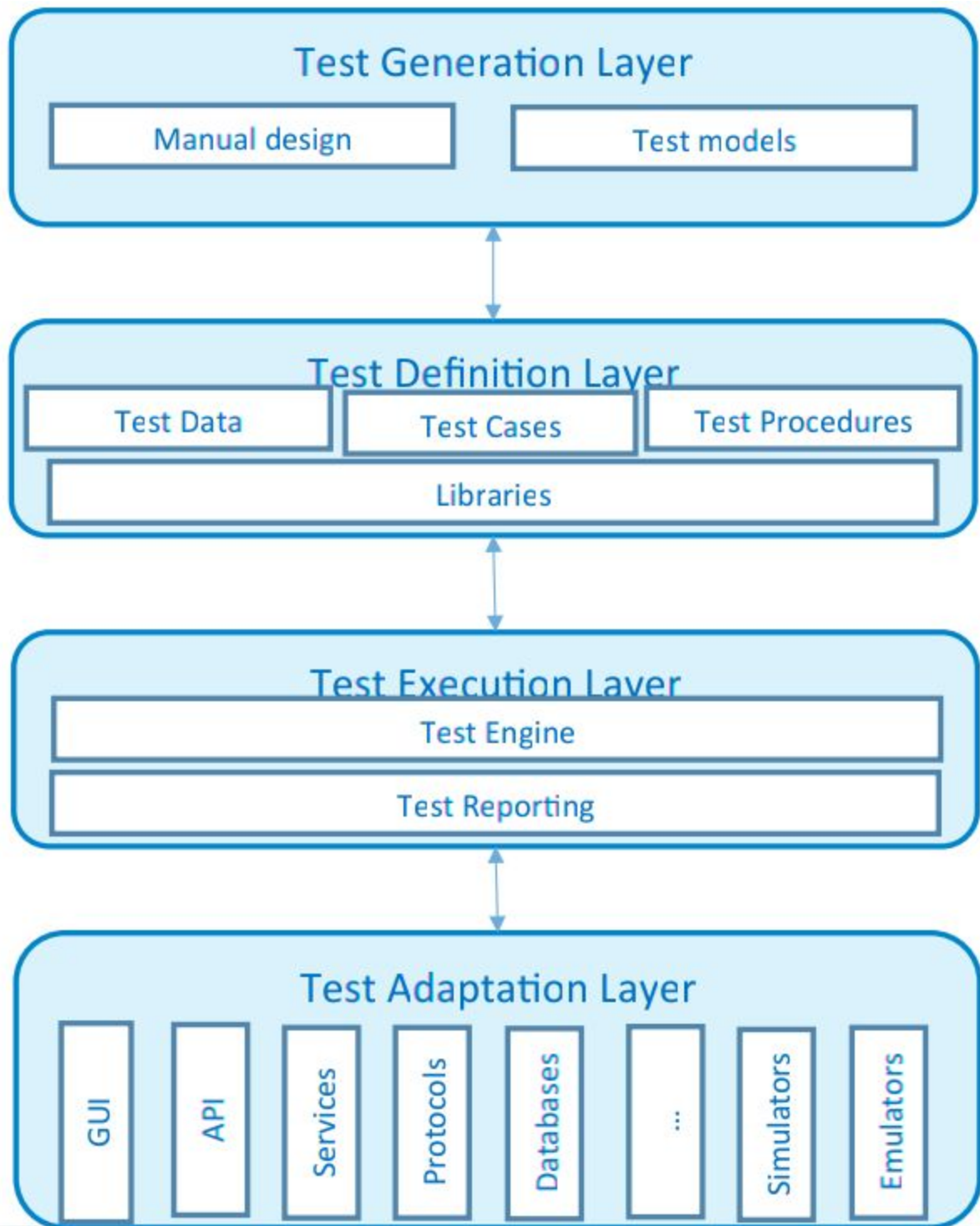
Workflow with an automated test framework



- tests are written by many testers

- they are managed by a test server
- tests are executed on different test environments
- test execution is logged
- an execution report is available

Typical architecture of an automated test framework



Test generation layer

The test generation layer consists of tool support for the following:

- Developing manual test cases
- Automatically generating test cases from models that define the SUT and/or its environment

Test definition layer

The test definition layer consists of tool support for the following:

- Defining test cases (at the abstract and/or concrete level)
- Defining test data for concrete test cases
- Defining test procedures for the execution of the test cases
- Access to test libraries as needed (for example in keyword-driven approaches)

Test execution layer

The test execution layer consists of tool support for the following:

- Executing test cases automatically
- Logging the test case execution

Test adaptation layer

The test adaptation layer consists of tool support for the following:

- Controlling the test harness
- Interacting with the SUT
- Monitoring the SUT
- Simulating or emulating the SUT environment

Transitioning manual to automated testing

LO: Enumerate and evaluate the importance of characteristics of the testing effort that one needs to take into account before transitioning to automated testing
LO: Enumerate the considerations for the test framework and for the management process that need to be considered, when preparing for use the automated test framework

Automated vs. manual tests

Prior to commencing an automated testing effort, one needs to consider the applicability and viability of creating automated vs. manual tests. The suitability criteria may include, but is not limited to:

- Frequency of use

- Complexity to automate
- Compatibility of tool support
- Maturity of test process
- Suitability of automation
- Sustainability of the automated environment
- Controllability of the SUT

Preparing the automated test environment

To adequately prepare for transitioning to an automated environment, the following need to be addressed:

Test framework:

- Availability of the automated tools in the test environment
- Test execution triggering
- Test execution time
- Test data and services (shared data or not)
- Automated tests logging and reporting

Test management:

- Scope of the test automation effort
- Education of test team to the paradigm shift
- Roles and responsibilities
- Cooperation between developers and automators

Mobile Software Testing

Mobile categories and input devices

LO: List the main types of devices in the world of mobile software
LO: Distinguish between the characteristics of input devices of computers and for mobile devices

Mobile device categories

Usability varies by mobile device category

- Regular cellphones – with a tiny screen
- Smartphones – mid-size screen and a full A-Z keypad
- Full-screen phones (iPhone, Android, Windows phones)
- Large tablets (10 inch) • Small tablets (7 inch)
- Smart watches

Mobile device categories

Test:

- Each category of device has its own strategy and goals. Take this into account when planning your testing.
- Test first for the devices that your application is committed to support.

a) Mobile phones specific:

- Test answering calls and behavior during calls.
- Test switching between mobile phone native apps (sms service, contacts, phone) and your application.

b) Tablet specific:

- Tablets are most likely shared devices – so the design has to be for multiuser devices.
- What does this translates into? Ex: the users might be reluctant to stay permanently signed into an app.
- Also, it's important to design recognizable application icons so they will stand out in the crowded listing of several user's apps.

Input devices for computers and mobile

There are more input devices for both desktop computers and mobiles, but in the following example we analyzed the standard ones.

Comparison between input devices:

Action	Mouse	Fingers
Precision	High	Low
Number of contact points	1	1 or 2-3 for multi-touch
Number of controls	3 (left, right, scroll)	1
Homing time	Yes	No
Signal states	Hover, mouse down, mouse-up	Finger-down, finger-up
Accelerated movements	Yes	No
Suitable for use with desktop monitors	Yes	No (arm fatigue)
Visible pointer / cursor	Yes	No
Obscures view of the screen	No (therefore allowing for continuous visual feedback)	Yes
Suitable for mobile	No	Yes
Direct engagement with screen	No	Yes
Accessibility support	Yes	No
Ease of learning	Fairly easy	Fairly easy



Input devices for computers and mobile

Test:

- The application you are developing for desktop computers and mobile devices has to have a dedicated design for each, to allow interaction with these two different input devices (mouse and fingertips).
- The mobile app has to allow more space for commands than the desktop app, to compensate for the lack of precision of fingertips compared to mice.
- The mobile app has to simplify the interaction with the users, because there is no such thing as left and right-click with fingers.
- The mobile app has to offer a way for the user to interact with one control at a time and minimize accidental interaction (due to lack of visibility and space, the user is more prone or accidentally activating a certain feature).

Mobile tasks and user behavior

LO: List the main tasks performed by users on mobile phones
LO: List the main tasks performed by users on tablets
LO: Contrast between the typical user behavior on mobile phones and on tablets

Mobile phones tasks

“Killing-time” apps

- Games
- Weather
- Social networking
- Maps & navigation
- Search
- Music
- News

The other “more productive” apps

- Shopping / retail
- Sport
- Productivity
- Communication
- Travel
- Health
- Education / learning
- Household / personal, etc

Tablets tasks

- Playing games
- Checking email
- Social networking sites
- Watching movies and videos
- Reading news
- Shopping (but not so much)

Typical behavior for mobile phone users

- They go to the mobiles when they want to “kill time” while waiting.
- They are rushed and impatient.
- The “killing time” activity is rather device-driven than user-driven. The users may have a very general goal (read something or play something). Then, they are happy to consume whatever content the device is offering.

Typical behavior for tablet users

- A lot of users say that the tablet has replaced their laptop. This doesn't mean they start making PowerPoint presentations on it, it means that the information-finding tasks are migrating from the laptop to the tablet.
- Tablets have dominated the media consumption.
- The information production is mostly dominated by responding to the emails.

- Users feel it's easier to shop on the desktop computers.
- Some users are worried about the security of e-commerce purchases on the tablet.

Comparison between mobile and tablet tasks

Generally, users perform more complex tasks on an tablet than on a phone.

- Ex: on an iPad the plan their vacation (destination, plane tickets) – but still they won't buy on the spot. On phones, it's just simple contextual needs – like finding the nearest hotel.

Difference in usage sessions:

- mobile phones: 3-5 minutes, very fragmented.
- tablet: from several minutes, up to one hour (on the train, at home). The sessions are not rushed and not more fragmented than on a desktop computer.

Test for each device type you are developing for:

- The typical session types for that specific device
- The user profile and behavior for that specific device

Designing software for mobile

LO: List the design issues of software running on mobile devices
LO: Explain the main problems with download times on mobile devices. List what kind of testing is necessary to for this possible issue
LO: Explain the main problems with scrolling on mobile devices. List what kind of testing is necessary to for this possible issue
LO: Explain the main problems with zooming on mobile devices. List what kind of testing is necessary to for this possible issue
LO: Give reasons why reading on mobile devices is harder. How is it possible to fix this issue?
LO: List the characteristics of software optimized for mobile use (sites or apps)

Design issues for mobile devices

Low discoverability: The interactive part is mostly hidden within the design.

Low memorability: Gestures are not consistent across applications and are hard to remember. They can be confusing for both the users and the device (as are all recognition-based user interfaces).

Accidental activation: This occurs when users touch by mistake some elements or make a gesture which unexpectedly initiates a feature.

Test:

- Make it very clear which elements are interactive in your applications: have a clear separation between content and commands.
- Don't complicate the gestures that a user has to make to access your application. Stick to standard gestures as much as possible.
 - Ex of standard gestures: tapping, dragging, swiping through the pages.
 - Ex of non-standard gestures: double-tapping, touching and holding, pinching in and out (for zooming), scrolling with two fingers, three-finger tap or drag (to zoom and move within the zoomed area).
- Make sure you give the users the space to click only the element that they need to activate. Make sure you provide a Back option or a way to undo the previous action.

Download times

Download times dominates the user experience.

- Most pages take far too long to load.
- Even the high-end phones deliver much slower browsing than desktop computers.
- As result: users are reluctant to request additional pages and they give up easily.

Test:

- Check the interaction cost. This cost depends on the type of device and the type of application. (interaction cost = number of atomic actions like clicks, scrolls, switching context – that a person needs to perform to fulfill a task on a computer or mobile device)
- Streamline the interaction so it involves as few page downloads as possible.
- Give users feedback about the state of the download (ex: by showing them a progress bar).

Scrolling

Scrolling causes major usability problems, especially in the sites not optimized for mobiles

- The main problem is that the pages scroll too much.
- As result: people lose track of where they were and what info is in the page. Also people scroll right past something, without noticing it.

Test:

- That users don't take too much time with the scrolling, thus degrading the short-term memory.
- That it's easy to reacquire the previous location on the page.
- That the user attention is not diverted from reading to the secondary task – which is scrolling.

Zooming

Zooming into a page allows users to read more easily.

Problem: users lose context. It is harder to know where they are on the page and what else is available

Bloated pages

They make users lose context and feel lost.

On computers these pages normally don't feel bloated, but rendered on a mobile they bloat.

Problem: unnecessarily big images and long texts take a long time to load. Also, they bury the items the users wanted to see.

Test

- Include only needed information that is likely useful to your users and add value to the page.
- Don't abuse images on a page. Many images usually translate into slow download times. You have to choose smaller size images and edit them for mobile (zoom, focus on one detail etc).

Why mobile reading is harder

Users can see less at any given time. They have to rely on their short-term memory when they are trying to understand anything that is not fully explained within the viewable reading space.

Users have to move around the page more, scrolling to refer to other parts of content, instead of simply glancing at the text.

Mobile users are in a hurry and get visibly angry at sites that waste their time.

Bullet points make information more scannable on a small screen.

Test:

- Cut the unnecessary paragraphs in the beginning of pages. Check if the text still makes sense without them.
- Stick to answering 2 questions: what and why.
- Old words are the best – check that you don't use too many neologisms or specialized terms that the readers might not understand.
- Defer secondary information to secondary screens (ex: progressive disclosure, like in Wikipedia; ex: summaries of the headlines in a list with articles)

Mobile optimized apps and sites

- Cut features to eliminate functionalities that are not core to the mobile use case.
- Cut content to reduce word count.
- Defer secondary information to secondary pages.
- Enlarge interface elements to accommodate the “fat finger” problem.
- Here the challenge is to eliminate features and word count without limiting the selection of products. A mobile site should have less information about each product and support fewer tasks that users can do with the products, but the range of items should remain the same as on the full site.

If the users can't find a product on a mobile site, they assume the company doesn't sell it and go elsewhere.

Do's and don'ts in mobile design and testing

LO: List the do's and don'ts in designing software running on mobile devices

Do's

- Have a good startup screen: in a mobile world with little commitment or patience, a single bad screen can cost millions in lost revenue and brand value. Startup screens are the ones who make a good or bad user impression, this is why they are crucial.
- Focus the user's attention on the essential content. This is the main usability guideline for mobile content. The comprehension score on mobile devices is twice as low as on Desktops (Cloze test).
- Have Back buttons, reliable search, homepages.
- Have direct access to content by touching headlines on the front page.
- Support standard navigation and gestures for users interacting with the mobile application you are developing.
- • If you have to choose between scrolling and chop the content into linear sequence of pages, choose the first option (so that users don't have to wait for a page load every 20-30 seconds or so).

Also, using the scroll makes it easier to go back to the parent-page, instead of tapping the “back” button many times.

The only exception here are the books, which are not designed to be read in a single session. But be aware that book readers make moving between pages very easy and without time consumption.

- Get rid of the swiping ambiguity: using swipe gestures (for scroll, page navigation, carousel browsing) together with tapping gestures in the same page may lead to the user either starting or ending the swiping gesture on an element that is interactive. This is called swipe overload.

Check that the interactive elements that can be tapped do not interfere with the swiping gesture. In case they do, minimize this interference.

- Avoid offering the users too much navigation possibilities at the same time: scroll, carousel, thumbnail tapping, hyperlinks, tables of contents, etc.

Don'ts

- Do not use early registration: don't make users pass through a registration screen as the first step.
- Do not have in the beginning of the app/webpage messages asking for the user's permission to push notifications or to access their current location.
- Do not present the users with lengthy instructions or users manuals. Most users are not motivated to read the instructions before getting any value from the app.
- Don't follow the "print metaphor" to the same degree as the desktop websites (print metaphor refers to designing a web page as close as possible to a printed magazine).

Intellectual property and licensing

Forms of intellectual property

LO: List and explain the two forms of intellectual property
LO: Explain why it is important to give IP protection

Categories of IP

Industrial property

- Patents (inventions)
- Trademarks
- Industrial design
- Geographic indications of source

Copyright

- Literary and artistic works
- Architectural designs

Rights related to copyright include those of performing artists in their performances and those of broadcasters in their radio and television programs

IP as intangible property

- Tangible properties: – Land, houses, cars
- Intangible properties: – intellectual property
- Intangible wealth is easily appropriated and reproduced.
- Once created, the marginal cost of reproduction is negligible.

Why give IP protection

- Capital expenditure for new products
- Research and Development
- Marketing and advertisement
- Maintaining loyal followers
- Profit

What can be done with IP

- Can be sold
- Can be bought
- Can be lease or rent
- Can pass under a will
- Can be assigned

IP laws and regulations

The US Laws For Intellectual Property Protection

- Copyright Act (1790; 2014)
- Trademarks Act (1870; 2006)
- Patent Act (1790, 2013)
- Industrial Design Act 1996
- Geographical Indications Act 2000
- Law of Tort (passing-off confidential information)

Technology Innovation Legislation

- Stevenson-Wylder Technology Innovation Act of 1980
- Bayh-Dole Act of 1980
- Small Business Innovation Development Act of 1982
- Cooperative Research Act of 1984
- Federal Technology Act of 1986

Industrial property

LO: Enumerate and describe the different forms of industrial property
LO: Explain what you need to test in a software system with regards to the industrial property rights

Industrial property

- Trade Secrets

- Trademarks
 - Trademark
 - Service Marks
 - Trade Name
- Patents
 - United States
 - Provisional
 - Full application
 - International
- Industrial design
- Geographical indication
- Confidential information

Trade Secrets

- Definition
 - Confidential, unpatented information that is protected by keeping the information secret
- Factors in determining if it is a trade secret
 - Extent information is known in industry
 - Extent measures taken to safeguard secrecy
 - Value of information to owner and competitors
 - Ease/difficulty in independent development
- Advantages
 - Easy to control
 - Easy and inexpensive (relatively) to protect
 - Indefinite term (determined by degree of protection)
 - No patentability requirements to qualify
- Disadvantages
 - Limited ability to exploit information
 - Competition can do reverse engineering
 - No protection against independent development
 - No statute law protecting trade secrets
- Remedy
 - Injunction or damages
 - Onus is on owner to establish case

Trademarks

- Definition
 - Identifying mark, word, logo or symbol used by someone in commerce to identify or distinguish their goods and services from all others
 - Sometimes confused with “trade name” which is the company name under which business is conducted.

- “mark” includes a device, brand, heading, label, ticket, name, signature, word, letter, numeral or any combination.
- Does not include sound or smell
- Can either be registered or not registered
- Advantages of registered trade marks:
 - Application can be made for goods and services
 - Perform certain function such as indication of quality, identifying a trade connection

Patents

- What is patentable?
 - Any new and useful art, process, machine, manufacture or composition of matter or any new and useful improvement
 - Includes software and mask works
- Patent can be applied for a product or a process. Patentable invention must be new, involves an inventive step and industrially applicable
- Key elements:
 - Novelty
 - Utility
 - Non-obviousness
- First to invent vs. first to file
- Term is 20 years from date of application (USA)
- One year grace period for prior disclosure
- Application published 18 months after filing
- Software Patents are software-based inventions.
 - Patentability criteria evolving and include:
 - Produces new/useful data interpretation
 - Controls device performing new/useful function
 - Solves computer-related problem
 - Improves existing computerized process

Industrial design

- Protection for industrial designs that are new or original
- Design are feature of shape, configuration, pattern or ornament
- The design must be applied to an article
- The design must be applied by an industrial process.
- It has to appeal to the eye

Geographical indications

- Geographical indication = an indication which identifies any goods as originating in a country or territory, or a region or locality where a given quality, reputation or other characteristic of the goods is essentially attributable to their geographical origin.
- Product must come from a particular geographical territory
- Uses a name link to the particular geographical nature of the territory

Ex:

- Swiss made
- Swiss chocolates
- Champagne
- Sweet tamarind

Confidential information

- Protection for confidential information under contract:
 - employer-employee
 - husband and wife
- To prove passing of confidential information, one needs to show:
 - that the information are confidential
 - recipient who obtained the information uses it
 - damages suffered by the owner

Ex:

- Customers list
- Secret recipes

What you need to test regarding the industrial property

- Trade Secrets
 - You mark and distribute the documentation related to sensitive information appropriately: requirements, specifications, architecture overview, design, test, help files and other documentation
- Trademarks
 - Check that you don't use other company's trademarks
 - Check if you must use your company's trademarks
- Patents, Industrial design
 - Check that you don't use other company's patents or industrial design without prior agreement
 - Check if there is any part of the software product or process that needs to be patented.
- Geographical indications
 - Check that you don't misuse geographical indicators in your application or its outcomes

- Check if your application should contain information to be marked as confidential (as specified by the license type, contracts with customers, service level agreements)
- Check if the company's procedures to protect confidential information are respected and applied in the tested software product
 - Check if that the confidential information of your application is encrypted or stored or deleted according to those specifications
- Test against unauthorized access to the confidential information contained by your application.

Copyright

LO: Describe the notion of copyright
LO: List the main tests needed to be done in software system with regards to US copyright laws
LO: Describe the notions of: public domain, fair use, misuse of sources

Definition and description

- Copyright is a legal device that provides the creator of a work of art or literature, or a work that conveys information or ideas, the right to control how the work is used
- Copyright is a law with the intent to advance the progress of knowledge by giving authors an economic incentive to keep on creating. It is a set of legal rights that the author has over his work for a limited period of time.
- Copyright insures that the person who created something is reimbursed for his intellectual work.
- These legal rights control everything from images to sound files and even whether or not you can photocopy - that protects the creator – gives creators the right to control how their works are used.
- There is no requirement for novelty or uniqueness as there is in patent law
- Protects manner of expression; not the idea, process or concept
- Exists automatically on creation of work
 - Legal registration enhances protection
- Term
 - author's life + 70 years (USA)
 - lesser of 95 years from first publication or 120 years from creation for works for hire (USA)
- Creations and works of art
 - drawings/prints musical works text
 - architectural plans motion pictures software
 - multimedia works internet-distributed content

Duration:

- Depends on whether it is pre or post 1 Jan. 1978 (USA)
- Pre - Depends on whether published?
 - Registered, first term, renewal etc.
- Post
 - Life of author + 50 years (USA)
 - Work-for-hire 75 years from publication, 100 years from creation which ever is first (USA)

Works for hire

Employer is considered the author when:

- work prepared by an employee within the scope of his/her employment
- work specially ordered or commissioned for use as a contribution to a collective work

Transfer of title vs Work-for-Hire

- under a work for hire, employer is considered the owner.
- Duration 75 years from pub or 100 from creation (USA) – Transfer of title – can be done after 35 years (USA)

Fair use

- Fair use allows use of a copyrighted work without seeking permission from the author. This is a law.
- Fair use is considered: criticism, comment, parody, news reporting, teaching, scholarship or research
- We look at these four factors to determine if a use is fair or not :
 1. purpose/character of the use – this is where we fit in, educational use. Has to be non-profit and educational
 2. the nature of the copyrighted work Has to be published and factual
 3. the portion of the work used Has to be relatively small and non-essential
 4. the effect of the use on the market Won't prevent the owner from making money

Allows

- you to quote passages from a book or a journal article as long as it is credited/cited.
- You to reproduce short segments of songs and movies as long as it is cited/credited to the author.
- You to use an entire work depends on the circumstances.

Examples

- Professor giving you a copy of an article he wants you to read
- Photocopying selections of materials in the library for your own use
- Library Electronic Reserves

Remember

- just because you don't need permission, you still need to cite.

Misuse of sources

Misusing sources may result in unintentional plagiarism due to:

- Wrongly paraphrasing
- Paraphrasing without citing
- Copying & Pasting

When in doubt, just search (internet or library) for the answer

What you need to test regarding copyrights

- Check what external source of information you have used in developing your software product:
 - other code (libraries, routines, pieces of code)
 - images, videos, sounds, trademarks, text

Check for each of the sources if they are in the public domain or not.

For the information not in the public domain AND depending on the purpose of your software (educational, commercial, etc), check that you have the permission of the author to use it.

Cite and reference the author of any external piece of information (in the public domain or copyrighted).

Licenses

LO: Explain what is a software license
LO: Give reasons why the need of software licenses
LO: Explain and compare the need for software licenses from the vendor's perspective and the customer's perspective
LO: List and explain the main types of software licenses
LO: Describe what one needs to test regarding software, sub-systems and software licensing

What is a License?

A license = a contract between licensor and licensee.

Licensor grants to licensee the right to practice the technology claimed in the licensed patent.

Licensor agrees not to sue licensee for infringing licensor's patent.

Licensing = In the broad sense, is a discipline that makes it possible to transfer technology from a proprietor (the licensor) to an interested purchaser (the licensee) in a well defined and effective manner.

Why the need to have licenses

Idea >> concept >> reduction to practice >> prototype >> scale-up >> production

Direct application by IP owner

Sale of IP to another entity who then productizes

License of IP to licensee(s)

Public disclosure of IP

Rights under a license

- To make
- To use
- To disclose to others
 - by publication
 - in a marketed product
 - in a service manual
 - as a sub-license to a third party
 - to lease
- To sell

Types of licenses

- Exclusive - there can only be one licensee; the licensor has no rights to exploit the technology/product
- Sole - exclusive but for the licensor; i.e., the licensor has rights to exploit the technology/product
- Non-exclusive - there is no limit to the number of potential licensees
- Irrevocable/Revocable for cause
- Territory: World-wide/Limited to named geographic regions
- Field of use: Market-specific
- Royalty-bearing/Royalty-free

Strategies for choosing a license

- Position in technology life-cycle
- Presence/absence of competition

- Technical, financial and marketing strengths of the parties
- Legal, political, and cultural environments
- Protection of proprietary information
- Remuneration hoped to be realized over the term of the license

Why license? – Licensor's reasons

- to make money
- to help sell products, services, equipment
- to secure a market
- to settle a patent dispute
- cross-licensing of other, existing technology
- rights to future technology
- to adapt a product to a local market
- to fulfill local laws
- to avoid antitrust/anti-competition or trade regulation problems

Why license? – Licensee's reasons

- to obtain rights to technology
- to supplement the licensee's R&D products
- to obtain continuing access to technical help
- to benefit from the good reputation of the licensor by gaining ability to use its trademark
- to obtain quick entry without cost and technology risks
- to obtain access to the licensor's facilities

What you need to test regarding licenses

1. Check that you are using the right license for the type of software that you are developing: – Check it for each local market in which you distribute your software
2. For the type of license that you are using
 - Check your duties according to the statement in the license:
 - Technical support
 - Support for new and legacy software
 - Backup / Availability (this can be comprised in a Service Level Agreement document too)
3. Check that you don't breach the license rules of other software (systems, libraries, routines or algorithms) you are using to develop your software – you need to check if you don't go over your rights to use other software or products (sell, use, disclose to others).

GPL (general public license)

- Q: Can any open source library be used in a commercial software or any specific licensed open source library?

- Yes, but you need to distinguish between commercial software used in-house/privately and commercial software bundled for sale or distribution.
- The GPL does not require you to release your modified version, or any part of it. You are free to make modifications and use them privately, without ever releasing them. This applies to organizations (including companies), too; an organization can make a modified version and use it internally without ever releasing it outside the organization.
- But if you release the modified version to the public in some way, the GPL requires you to make the modified source code available to the program's users, under the GPL.
- Thus, the GPL gives permission to release the modified program in certain ways, and not in other ways; but the decision of whether to release it is up to you.