Session 2

Evaluation and Testing

Contents

1: Introduction to Testing

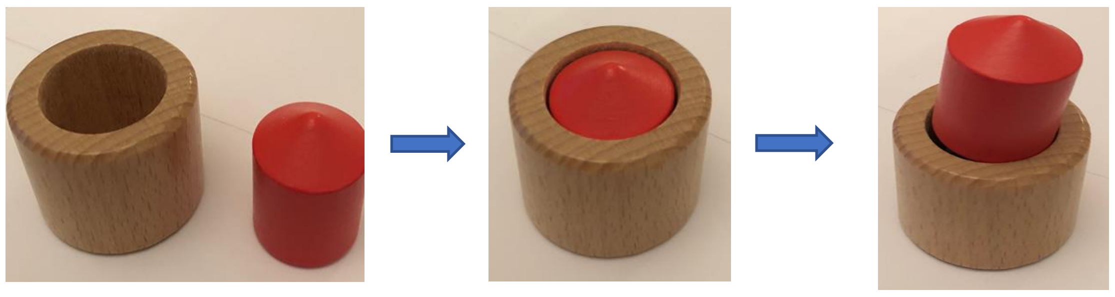
Activity 1: Reflecting on Software Testing.

Read *Evaluate testing the mini rocket* below and note any thoughts you have on how you would carry out any tests. You might consider:

- Can I test everything at the same time?
- Would I prioritise anything when testing?
- Are there any testing suggestions I would add/remove from the list?
- Is it important to have a testing strategy?

Evaluate testing the mini rocket

Imagine testing a product such as the one shown – a STEM toy, in which the red 'rocket' is supposed to fly out of its holder when a person blows across the top.



If you were to test this toy, how would you do it and how long would it take? It is tempting perhaps to answer that, as you just want to test if it works, the test might consist of blowing across the top as required and if it functions as expected, your testing is over in 5 seconds.

However, you were not given full requirements for your test, and you may want to do some or all of the following:

1. Assess the power required for the toy to function properly.
2. Who could produce such power. Possibly estimate an age where you could reasonably assume that the person could manage to do this.
3. Analyse any potential dangers in the product? How could these be tested? Is there a danger of a young child swallowing this?

4. Test if the paint on the rocket is poisonous?

5. Evaluate the size / raw materials / colour scheme / shape and model alternatives, which could be used in future versions.

6. Could the test process be done earlier / integrated into the manufacture of the toy.

Notes:

In the case of software, **testing** involves much more than just a cursory check on whether something works or not. *Consider the scores of terms in the glossary from Ad-hoc testing to White box testing, and it becomes easy to see that this is a huge field*. Whether in industry, college or school, the testing process can involve evaluating the software; making suggestions for future design; finding inefficiencies in the code; discovering unusual situations where the code doesn't work; considering the user's experience; making suggestions about the look and feel of the program or app.

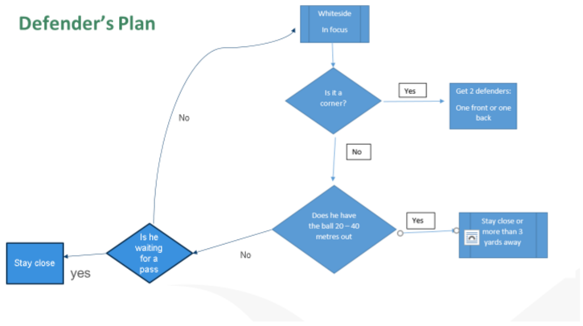
Some interesting scenarios

Aircraft manufacturer Boeing suffered losses of millions of dollars, when they tried to cut costs on their **software testing**. A problem where the nose of the aircraft went too high was overcorrected in a solution which was not properly tested. More details are in a Bloomberg article in the appendices.

The amount of bugs in large programs is enormous and lead to the tester having to make decisions of how to approach a project. In Windows XP for example, there were 65,000 bugs found before the product went to market. And these were just the ones found.

Some of the highest paid people in the IT industry are testers, as they have the skill of, not just being creative, but being able to analyse the work of software developers.

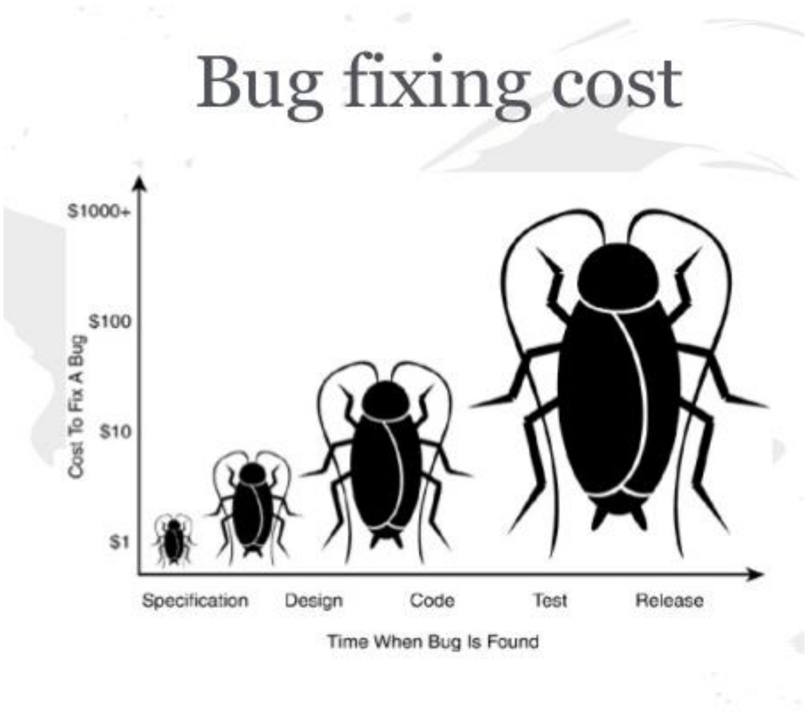
To give an analogy from sport, testers are similar to defenders – they have to do an analysis of each attacker they will be facing, and develop a plan, one of which is shown here. The attackers, on the other hand,


Defender's Plan
Whiteside In focus
Is it a corner?
Yes
Get 2 defenders: One front or one back
No
Does he have the ball 20 – 40 metres out
Yes
Stay close or more than 3 yards away
No
Is he waiting for a pass
yes
Stay close
No

doesn't need to do this level of research every game.

The **software tester** has a lot of responsibility. Consider one of the popular word processing programs. A small mistake in, for example, a short-cut key, could have disastrous consequences for thousands of end users.

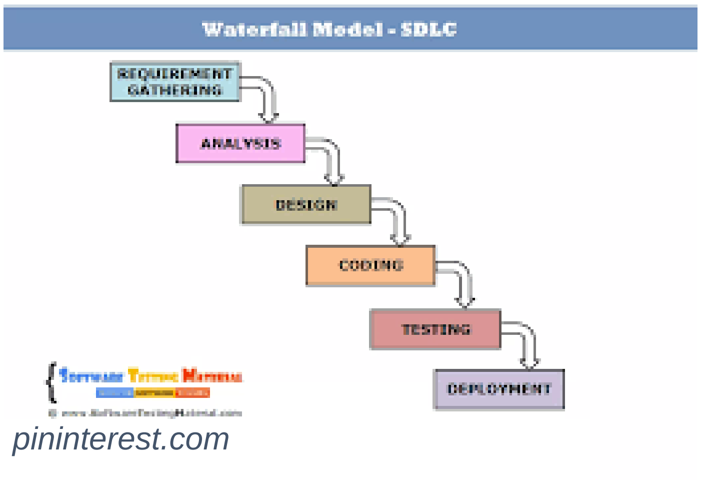
The cost of fixing **Bugs** which are not found in time is huge when compared from a bug that is found in the early stages, as can be seen in the next diagram:



www.systemsemantics.com

2: Software Testing

Recent changes to software development methodologies are reflected in a changing role for software testers.

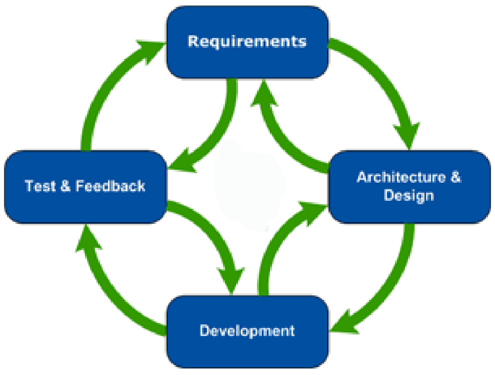


pininterest.com

In the traditional **Waterfall model** (and other staged models) the testing process takes place at a particular point in the software development cycle.

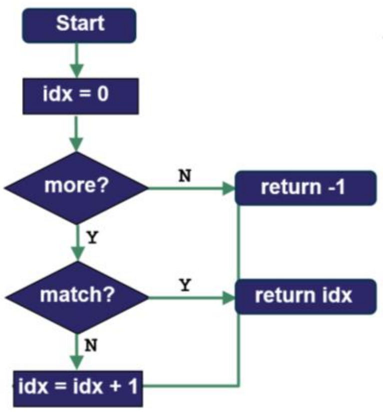
Two more recent software development methodologies are **Test-driven development (TDD)** and **Agile development**. In TDD, the test cases are written before any coding is done. This has led to a shift of roughly one tester for every eight developers to one tester for three developers. Microsoft, by the way, generally had a one-to one ratio, but that was due to the huge amount of localisation required for their products.

In **Agile Software Development**, the distinct stages have been removed and testers are involved right from the start, in a constant dialogue with clients, designers and project managers. The role of the tester has been enhanced and made more central.



karthiksangi.wordpress.com

Testing the Linear Search algorithm



idx	0	1	2	3	4	5	6
values	21	17	-1	26	22	-5	24

Is searching for 26 a sufficient test of this algorithm?

Notes:

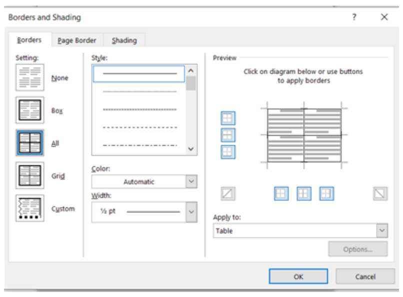
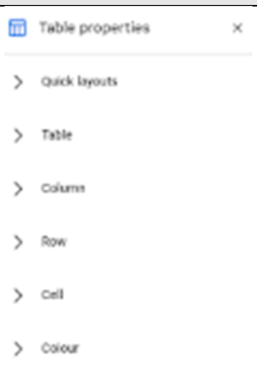
Is searching for 26 a sufficient test of this algorithm?

Explain your answer.

Activity 2: Testing the functionality of the *Shading Tab* in the *Borders and Shading* dialog box in a MS Word document/ Border Colour and Cell Background using *Table Properties* in a Google Document.

Create a table with 2 columns and a number of rows in MS Word/in a Goggle Doc. Like the table here.

Using the table properties (see below) can you investigate testing the functionality of the Shading Tab in the Borders and Shading dialog box in a MS Word document/ Border Colour and Cell Background in a Google Document by testing several possible combinations of colours, border size and types.

MS Word Test the functionality of the Shading Tab in the Borders and Shading dialog box	**Google Document** Test the functionality of the Border Colour and Cell Background using *Table Properties.*
 Borders and Shading Borders Page Border Shading Setting: None Box All Grid Custom Style: Color: Automatic Width: ½ pt Preview Click on diagram below or use buttons to apply borders Apply to: Table Options... OK Cancel	 Table properties Quick layouts Table Column Row Cell Colour

Consider the questions:

- Can I test every possible combination of colours, border size and types?
- Do I need to prioritise what I need to test and if so, how would I do that?

Note your thoughts below.

Notes

In this Breakout Activity, you are asked to test the Shading tab of the Borders and Shading dialogue box or the Border Colour and Cell Background using Table Properties in a Google Document.. With the thousands of different colours and hundreds of different styles of table, you could be left with millions of test cases, which would be impossible to carry out in a reasonable time. So, the tester will often use **heuristics** to decide what would and wouldn't be included in the **test plan**.

3: Types of Testing / Testers' roles in different software development models

Activity 3: Home Expert Activity - Investigating types of testing.

While investigating the different types of testing you could consider what the testing involves with an example and who carries out the test. Write your findings below.

1. **Functional** v non-functional testing
2. Unit v Integration testing
3. **System** v User Acceptance Testing
4. White v Black box testing
5. Alpha v Beta Testing
6. Usability v Security Testing
7. Regression v Smoke Testing
8. Accessibility v Stress Testing
9. Other

Notes:

4: Test Driven Development (TDD)

Looking at TDD / Agile / Waterfall Software Development Models

Investigate and answer the following questions:

- Does TDD fit in the Agile framework?
- Is TDD the same as Agile development?
- Is Waterfall development relevant?
- How does the role of testing change, according to these methodologies?

Notes:

5: Reflection

From what we covered today, what could you bring to your classroom?

How could you incorporate testing into future ALTS?

Notes:

End

6: Appendices:

Glossary of terms used in Software Testing

A

Ad hoc testing
Testing carried out informally without test cases or other written test instructions.

Agile development
A development method that emphasizes working in short iterations. Automated testing is often used. Requirements and solutions evolve through close collaboration between team members that represent both the client and supplier.

Alpha testing
Operational testing conducted by potential users, customers, or an independent test team at the vendor’s site. Alpha testers should not be from the group involved in the development of the system, in order to maintain their objectivity. Alpha testing is sometimes used as acceptance testing by the vendor.

Anomaly
Any condition that deviates from expectations based on requirements specifications, design documents, standards etc. A good way to find anomalies is by testing the software.

B

Beta testing
Test that comes after alpha tests, and is performed by people outside of the organization that built the system. Beta testing is especially valuable for finding usability flaws and configuration problems.

Bespoke software:
Software developed specifically for a set of users or customers. The opposite is off-the-shelf software.

Big-bang integration
An integration testing strategy in which every component of a system is assembled and tested together; contrast with other integration testing strategies in which system components are integrated one at a time.

Black box testing
Testing in which the test object is seen as a “black box” and the tester has no knowledge of its internal structure. The opposite of white box testing.

Bottom-up integration
An integration testing strategy in which you start integrating components from the lowest level of the system architecture. Compare to big-bang integration and top-down integration.

Boundary value analysis
A black box test design technique that tests input or output values that are on the edge of what is allowed or at the smallest incremental distance on either side of an edge. For example, an input field that accepts text between 1 and 10 characters has six boundary values: 0, 1, 2, 9, 10 and 11 characters.

Bug
A slang term for fault, defect, or error. Originally used to describe actual insects causing malfunctions in mechanical devices that predate computers. The International Software Testing Qualifications Board (ISTQB) glossary explains that “a human being can make an error (mistake), which produces a defect (fault, bug) in the program code, or in a document. If a defect in code is executed, the system may fail to do what it should do (or do something it shouldn’t), causing a failure. Defects in software, systems or documents may result in failures, but not all defects do so.”

C

Capture/playback tool
See record and playback tool.

CAST
A general term for automated testing tools. Acronym for computer-aided software testing.

Change request
A type of document describing a needed or desired change to the system.

Checklist
A simpler form of test case, often merely a document with short test instructions (“one-liners”). An advantage of checklists is that they are easy to develop. A disadvantage is that they are less structured than test cases. Checklists can complement test cases well. In exploratory testing, checklists are often used instead of test cases.

Client
The part of an organization that orders an IT system from the internal IT department or from an external supplier/vendor. See also supplier.

Code coverage
A generic term for analysis methods that measure the proportion of code in a system that is executed by testing. Expressed as a percentage, for example, 90 % code coverage.

Code review
See Review.

Code standard
Description of how a programming language should be used within an organization. See also naming standard.

Compilation
The activity of translating lines of code written in a human-readable programming language into machine code that can be executed by the computer.

Component
The smallest element of the system, such as class or a DLL.

Component testing
Test level that evaluates the smallest elements of the system. See also component. Also known as unit test, program test and module test.

Configuration management
Routines for version control of documents and software/program code, as well as managing multiple system release versions.

Configuration testing
A test to confirm that the system works under different configurations of hardware and software, such as testing a website using different browsers.

Context-driven testing
Testing which makes use of debugging techniques inspired by real-world usage conditions. It is a method of testing which encourages testers to develop testing opportunities based on the specific details of any given situation.

D

Daily build
A process in which the test object is compiled every day in order to allow daily testing. While it ensures that defect reports are reported early and regularly, it requires automated testing support.

Debugging
The process in which developers identify, diagnose, and fix errors found. See also bug and defect.

Decision table
A test design and requirements specification technique. A decision table describes the logical conditions and rules for a system. Testers use the table as the basis for creating test cases.

Defect
A flaw in a component or system that can cause the component or system to fail to perform its required function. A defect, if encountered during execution, may cause a failure of the component or system.

Defect report
A document used to report a defect in a component, system, or document. Also known as an incident report.

Deliverable
Any product that must be delivered to someone other than the author of the product. Examples of deliverables are documentation, code and the system.

Desk checking
A static testing technique in which the tester reads code or a specification and “executes” it in his mind.

Document review
See review.

Driver
See test driver.

Dynamic testing
Testing performed while the system is running. Execution of test cases is one example.

E

End-to-end testing
Testing used to test whether the performance of an application from start to finish conforms with the behaviour that is expected from it. This technique can be used to identify system dependencies and confirm the integrity of data transfer across different system components remains.

Entry criteria
Criteria that must be met before you can initiate testing, such as that the test cases and test plans are complete.

Error
A human action that produces an incorrect result.

Error description
The section of a defect report where the tester describes the test steps he/she performed, what the outcome was, what result he/she expected, and any additional information that will assist in troubleshooting.

Error guessing
Experience-based test design technique where the tester develops test cases based on his/her skill and intuition, and experience with similar systems and technologies.

Execute
Run, conduct. When a program is executing, it means that the program is running. When you execute or conduct a test case, you can also say that you are running the test case.

Exhaustive testing
A test approach in which you test all possible inputs and outputs.

Exit criteria
Criteria that must be fulfilled for testing to be considered complete, such as that all high-

priority test cases are executed, and that no open high-priority defect remains. Also known as completion criteria.

Expected result
A description of the test object's expected status or behaviour after the test steps are completed. Part of the test case.

Exploratory testing
A test design technique based on the tester's experience; the tester creates the tests while he/she gets to know the system and executes the tests.

External supplier
A supplier/vendor that doesn't belong to the same organization as the client/buyer. See also internal supplier.

Extreme programming
An agile development methodology that emphasizes the importance of pair programming, where two developers write program code together. The methodology also implies frequent deliveries and automated testing.

F

Factory acceptance test
Acceptance testing carried out at the supplier's facility, as opposed to a site acceptance test, which is conducted at the client's site.

Failure
Deviation of the component or system under test from its expected result.

Fault Injection
A technique used to improve test coverage by deliberately inserting faults to test different code paths, especially those that handle errors and which would otherwise be impossible to observe.

Formal review
A review that proceeds according to a documented review process that may include, for example, review meetings, formal roles, required preparation steps, and goals. Inspection is an example of a formal review.

Functional integration
An integration testing strategy in which the system is integrated one function at a time. For example, all the components needed for the "search customer" function are put together and tested one by one.

Functional testing
Testing of the system's functionality and behaviour; the opposite of non-functional testing.

G

Gray-box testing
Testing which uses a combination of white box and black box testing techniques to carry out software debugging on a system whose code the tester has limited knowledge of.

I

IEEE 829
An international standard for test documentation published by the IEEE organization. The full name of the standard is IEEE Standard for Software Test Documentation. It includes templates for the test plan, various test reports, and handover documents.

Impact analysis
Techniques that help assess the impact of a change. Used to determine the choice and extent of regression tests needed.

Incident
A condition that is different from what is expected, such a deviation from requirements or test cases.

Independent testing
A type of testing in which testers' responsibilities are divided up in order to maintain their objectivity. One way to do this is by giving different roles the responsibility for various tests. You can use different sets of test cases to test the system from different points of view.

Informal review
A review that isn't based on a formal procedure.

Installation test
A type of test meant to assess whether the system meets the requirements for installation and uninstallation. This could include verifying that the correct files are copied to the machine and that a shortcut is created in the application menu.

Integration testing
A test level meant to show that the system's components work with one another. The goal is to find problems in interfaces and communication between components.

Internal supplier
Developer that belongs to the same organization as the client. The IT department is usually the internal supplier. See also external supplier.

ISTQB
International Software Testing Qualifications Board. ISTQB is responsible for international programs for testing certification.

Iteration
A development cycle consisting of a number of phases, from formulation of requirements to delivery of part of an IT system. Common phases are analysis, design, development, and testing. The practice of working in iterations is called iterative development.

L

Load testing
A type of performance testing conducted to evaluate the behaviour of a component or system with increasing load, e.g. numbers of concurrent users and/or numbers of transactions. Used to determine what load can be handled by the component or system. See also performance testing and stress testing.

M

Maintainability
A measure of how easy a given piece of software code is to modify in order to correct defects, improve or add functionality.

Maintenance
Activities for managing a system after it has been released in order to correct defects or to improve or add functionality. Maintenance activities include requirements management, testing, development amongst others.

Module testing
See component testing.

N

Naming standard
The standard for creating names for variables, functions, and other parts of a program. For example, strName, sName and Name are all technically valid names for a variable, but if you don't adhere to one structure as the standard, maintenance will be very difficult.

Negative testing
A type of testing intended to show that the system works well even if it is not used correctly. For example, if a user enters text in a numeric field, the system should not crash.

Non-functional testing
Testing of non-functional aspects of the system, such as usability, reliability, maintainability, and performance.

NUnit
An open source framework for automated testing of components in Microsoft .Net applications.

O

Open source
A form of licensing in which software is offered free of charge. Open source software is frequently available via download from the internet, from www.sourceforge.net for example.

Operational testing
Tests carried out when the system has been installed in the operational environment (or simulated operational environment) and is otherwise ready to go live. Intended to test

operational aspects of the system, e.g. recoverability, co-existence with other systems and resource consumption.

P

Pair programming
A software development approach where two developers sit together at one computer while programming a new system. While one developer codes, the other makes comments and observations, and acts as a sounding board. The technique has been shown to lead to higher quality thanks to the de facto continuous code review – bugs and errors are avoided because the team catches them as the code is written.

Pair testing
Test approach where two persons, e.g. two testers, a developer and a tester, or an end-user and a tester, work together to find defects. Typically, they share one computer and trade control of it while testing. One tester can act as observer when the other performs tests.

Performance testing
A test to evaluate whether the system meets performance requirements such as response time or transaction frequency.

Positive testing
A test aimed to show that the test object works correctly in normal situations. For example, a test to show that the process of registering a new customer functions correctly when using valid test data.

Postconditions
Environmental and state conditions that must be fulfilled after a test case or test run has been executed.

Preconditions
Environmental and state conditions that must be fulfilled before the component or system can be tested. May relate to the technical environment or the status of the test object. Also known as prerequisites or preparations.

Prerequisites
See preconditions.

Priority
The level of importance assigned to e.g. a defect.

Professional tester
A person whose sole job is testing.

Program testing
See component testing.

Q

Quality
The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

Quality assurance (QA)
Systematic monitoring and evaluation of various aspects of a component or system to maximize the probability that minimum standards of quality are being attained.

R

Record and playback tool
Test execution tool for recording and playback of test cases often used to support automation of regression testing. Also known as capture/playback.

Regression testing
A test activity generally conducted in conjunction with each new release of the system, in order to detect defects that were introduced (or discovered) when prior defects were fixed. Compare to Re-testing.

Release
A new version of the system under test. The release can be either an internal release from developers to testers, or release of the system to the client. See also release management.

Release testing
A type of non-exhaustive test performed when the system is installed in a new target environment, using a small set of test cases to validate critical functions without going into depth on any one of them. Also called smoke testing – a funny way to say that, as long as the system does not actually catch on fire and start smoking, it has passed the test.

Requirements management
A set of activities covering gathering, elicitation, documentation, prioritization, quality assurance and management of requirements for an IT system.

Re-testing
A test to verify that a previously-reported defect has been corrected.

Retrospective meeting
A meeting at the end of a project/a sprint during which the team members evaluate the work and learn lessons that can be applied to the next project or sprint.

Review
A static test technique in which the reviewer reads a text in a structured way in order to find defects and suggest improvements. Reviews may cover requirements documents, test documents, code, and other materials, and can range from informal to formal.

Reviewer
A person involved in the review process that identifies and documents discrepancies in the item being reviewed. Reviewers are selected in order to represent different areas of expertise, stakeholder groups and types of analysis.

Risk
A factor that could result in future negative consequences. Is usually expressed in terms of impact and likelihood.

Risk-based testing
A structured approach in which test cases are chosen based on risks. Test design techniques like boundary value analysis and equivalence partitioning are risk-based. All testing ought to be risk-based.

S

Scalability testing
A component of non-functional testing, used to measure the capability of software to scale up or down in terms of its non-functional characteristics.

Scenario
A sequence of activities performed in a system, such as logging in, signing up a customer, ordering products, and printing an invoice. You can combine test cases to form a scenario especially at higher test levels.

Scrum
An iterative, incremental framework for project management commonly used with agile software development.

Session-based testing
An approach to testing in which test activities are planned as uninterrupted, quite short, sessions of test design and execution, often used in conjunction with exploratory testing.

Severity
The degree of impact that a defect has on the development or operation of a component or system.

Site acceptance testing (SAT)
Acceptance testing carried out onsite at the client's location, as opposed to the developer's location. Testing at the developer's site is called factory acceptance testing (FAT).

Smoke testing
See release testing.

State transition testing
A test design technique in which a system is viewed as a series of states, valid and invalid transitions between those states, and inputs and events that cause changes in state.

Static testing
Testing performed without running the system. Document review is an example of a static test.

Stress testing
Testing meant to assess how the system reacts to workloads (network, processing, data

volume) that exceed the system's specified requirements. Stress testing shows which system resource (e.g. memory or bandwidth) is first to fail.

Supplier
The organization that supplies an IT system to a client. Can be internal or external. Also called vendor. Contrast with Client.

System
The integrated combination of hardware, software, and documentation.

System integration testing
A test level designed to evaluate whether a system can be successfully integrated with other systems (e.g. that the tested system works well with the finance system). May be included as part of system-level testing, or be conducted as its own test level in between system testing and acceptance testing.

System testing
Test level aimed at testing the complete integrated system. Both functional and non-functional tests are conducted.

T

Test automation
The process of writing programs that perform test steps and verify the result.

Test basis
The documentation on which test cases are based.

Test case
A structured test script that describes how a function or feature should be tested, including test steps, expected results preconditions and postconditions.

Test data
Information that completes the test steps in a test case with e.g. what values to input. In a test case where you add a customer to the system the test data might be customer name and address. Test data might exist in a separate test data file or in a database.

Test driven development
A development approach in which developers writes test cases before writing any code.

Test driver
A software component (driver) used during integration testing in order to emulate (i.e. to stand in for) higher-level components of the architecture. For example, a test driver can emulate the user interface during tests.

Test environment
The technical environment in which the tests are conducted, including hardware, software, and test tools. Documented in the test plan and/or test strategy.

Test execution
The process of running test cases on the test object.

Test level
A group of test activities organized and carried out together in order to meet stated goals. Examples of levels of testing are component, integration, system, and acceptance test.

Test log
A document that describes testing activities in chronological order.

Test manager
The person responsible for planning the test activities at a specific test level. Usually responsible for writing the test plan and test report. Often involved in writing test cases.

Test object
The part or aspects of the system to be tested. Might be a component, subsystem, or the system as a whole.

Test plan
A document describing what should be tested by whom, when, how, and why. The test plan is bounded in time, describing system testing for a particular version of a system, for example. The test plan is to the test leader what the project plan is to the project manager.

Test policy
A document that describes how an organization runs its testing processes at a high level. It may contain a description of test levels according to the chosen life cycle model, roles and responsibilities, required/expected documents, etc.

Test process
The complete set of testing activities, from planning through to completion. The test process is usually described in the test policy.

Test report
A document that summarizes the process and outcome of testing activities at the conclusion of a test period. Contains the test manager's recommendations, which in turn are based on the degree to which the test activities attained its objectives. Also called test summary report.

Test run
A group of test cases e.g. all the test cases for system testing with owner and end-date.

Tests on one test level are often grouped into a series of tests, i.e. two-week cycles consisting of testing, re-testing, and regression testing. Each series can be a test run.

Test script
Automated test case that the team creates with the help of a test automation tool. Sometimes also used to refer to a manual test case, or to a series of interlinked test cases.

Test specification
A document containing a number of test cases that include steps for preparing and resetting the system. In a larger system you might have one test specification for each subsystem.

Test strategy
Document describing how a system is usually tested.

Test suite
A group of test cases e.g. all the test cases for system testing.

Testing
A set of activities intended to evaluate software and other deliverables to determine if that they meet requirements, to demonstrate that they are fit for purpose and to find defects.

Third-party component
A part of an IT system that is purchased as a packaged/complete product instead of being developed by the supplier/vendor.

Top-down integration
An integration test strategy, in which the team starts to integrate components at the top level of the system architecture.

TPI
Test Process Improvement. A method of measuring and improving the organization's maturity with regard to testing.

Traceability
Analysis of a prior chain of events, as well as the ability to follow an object such as a document or a program through various versions. Traceability enables you to determine the impact of a change in requirements, assuming you also develop a traceability matrix.

U

UML
Unified Modeling Language. A technique for describing the system in the form of use cases. See also use case.

Unit test
See component test.

Unit test framework
Software or class libraries that enable developers to write test code in their regular programming language. Used to automate component and integration testing.

Usability
The capability of the software to be understood, learned, used and attractive to the user.

Usability testing
A test technique for evaluating a system's usability. Frequently conducted by users performing tasks in the system while they describe their thought process out loud.

Use case
A type of requirements document in which the requirements are written in the form of sequences that describe how various actors in the system interact with the system.

V

V-model
A software development lifecycle model that describes requirements management, development, and testing on a number of different levels.

Validation
Tests designed to demonstrate that the developers have built the correct system. Contrast with verification, which means testing that the system has been built correctly. A large number of validation activities take place during acceptance testing.

Verification
Tests designed to demonstrate that the developers have built the system correctly. Contrast with validation, which means testing that the correct system has been built. A large number of verification activities take place during component testing.

Versioning
Various methods for uniquely identifying documents and source files, e.g. with a unique version number. Each time the object changes, it should receive a new version number. See also release management.

W

Waterfall model
A sequential development approach consisting of a series of phases carried out one by one. This approach is not recommended due to a number of inherent problems.

White box testing
A type of testing in which the tester has knowledge of the internal structure of the test object. White box testers may familiarize themselves with the system by reading the program code, studying the database model, or going through the technical specifications. Contrast with black box testing.

(adapted from reqtest.com/blog/glossary-of-testing-terms and softwaretestinghelp.com)

Boeing's 737 Max Software Outsourced to $9-an-Hour Engineers

By Peter Robison, June 28, 2019

Plane maker and suppliers used lower-paid temporary workers. Engineers feared the practice meant code wasn't done right.

It remains the mystery at the heart of Boeing Co.'s 737 Max crisis: how a company renowned for meticulous design made seemingly basic software mistakes leading to a pair of deadly crashes. Longtime Boeing engineers say the effort was complicated by a push to outsource work to lower-paid contractors.

The cockpit of a grounded 737 Max 8 aircraft.

Photographer: Dimas Ardian/Bloomberg

The Max software -- plagued by issues that could keep the planes grounded months longer after U.S. regulators this week revealed a new flaw -- was developed at a time Boeing was laying off experienced engineers and pressing suppliers to cut costs.

Increasingly, the iconic American plane maker and its subcontractors have relied on temporary workers making as little as $9 an hour to develop and test software, often from countries lacking a deep background in aerospace -- notably India.

Boeing 737 Max prepares for take-off during testing in 2016.

Photographer: Mike Kane/Bloomberg

Related: Pilots Flagged Software Problems on Boeing Jets Besides Max
In offices across from Seattle’s Boeing Field, recent college graduates employed by the Indian software developer HCL Technologies Ltd. occupied several rows of desks, said Mark Rabin, a former Boeing software engineer who worked in a flight-test group that supported the Max.

The coders from HCL were typically designing to specifications set by Boeing. Still, “it was controversial because it was far less efficient than Boeing engineers just writing the code,” Rabin said. Frequently, he recalled, “it took many rounds going back and forth because the code was not done correctly.”

Boeing’s cultivation of Indian companies appeared to pay other dividends. In recent years, it has won several orders for Indian military and commercial aircraft, such as a $22 billion one in January 2017 to supply SpiceJet Ltd. That order included 100 737-Max 8 jets and represented Boeing’s largest order ever from an Indian airline, a coup in a country dominated by Airbus.
Based on resumes posted on social media, HCL engineers helped develop and test the Max’s flight-display software, while employees from another Indian company, Cyient Ltd., handled software for flight-test equipment.

(Source: Bloomberg.com)