



ISTQB® - Foundation Level Certified Tester Slide Handout

Wouter Vrijen
June 2024

V 4.0

Version Control

Version	Date	Remarks
1.0.1	06-Mar-2024	Initial version on the 2023 syllabus
1.0.2	18-Jun-2024	Fixing of small issues
1.0.3	19-Aug-2024	Trademark additions ISTQB®

Training Agenda

- ▶ Introduction of ISTQB® and the Foundation Exam
- ▶ Fundamentals of Testing
- ▶ Testing throughout the software life-cycle
- ▶ Static Testing
- ▶ Test Analysis and Design
- ▶ Managing the Test Activities
- ▶ Test Tools

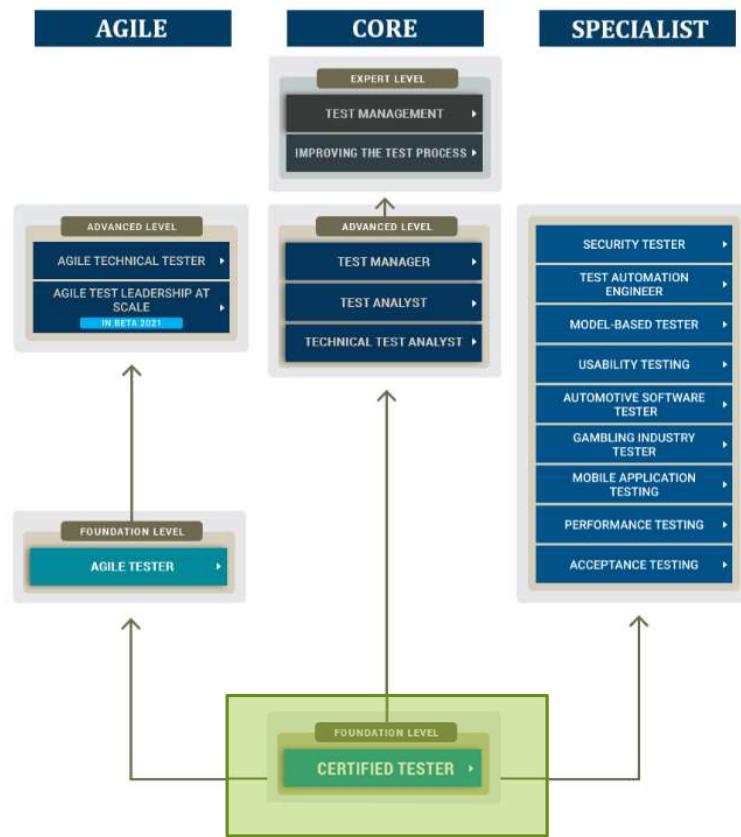


Introduction to ISTQB®

- ▶ The International Software Testing Qualifications Board (ISTQB®) is a software testing qualification certification organization that was founded in November 2002, and is a non-profit association legally registered in Belgium
- ▶ The ISTQB® Certified Tester scheme has become the world-wide leader in the certification of competences in software testing, its' certifications being internationally recognized and highly respected.

<http://www.ISTQB.org/about-ISTQB.htm>

ISTQB® Levels & Modules



Business Outcomes

- ▶ FL-BO1 Understand what testing is and why it is beneficial
- ▶ FL-BO2 Understand fundamental concepts of software testing
- ▶ FL-BO3 Identify the test approach and activities to be implemented depending on the context of testing
- ▶ FL-BO4 Assess and improve the quality of documentation
- ▶ FL-BO5 Increase the effectiveness and efficiency of testing
- ▶ FL-BO6 Align the test process with the software development lifecycle
- ▶ FL-BO7 Understand test management principles
- ▶ FL-BO8 Write and communicate clear and understandable defect reports
- ▶ FL-BO9 Understand the factors that influence the priorities and efforts related to testing
- ▶ FL-BO10 Work as part of a cross-functional team
- ▶ FL-BO11 Know risks and benefits related to test automation
- ▶ FL-BO12 Identify essential skills required for testing
- ▶ FL-BO13 Understand the impact of risk on testing
- ▶ FL-BO14 Effectively report on test progress and quality

K-Levels

- ▶ **K1: Remember**
 - ▶ The candidate should remember or recognize a term or a concept
- ▶ **K2: Understand**
 - ▶ The candidate should select an explanation for a statement related to the question topic
- ▶ **K3: Apply**
 - ▶ The candidate should select the correct application of a concept or technique and apply it to a given context
- ▶ **K4: Analyze**
 - ▶ The candidate can separate information related to a procedure or technique into its constituent parts for better understanding and can distinguish between facts and inferences

Information on the Foundation Exam

- ▶ 40 multiple-choice questions with 1 point for each correct answer
- ▶ A pass mark of 65% (26 or more points)
- ▶ Exam questions are distributed across K-Levels with estimated answer times:

K-Level	Nr of Questions	Question Time	Total Time per K-Level
K1	8	1 min	8 min
K2	24	1 min	24 min
K3	8	3 min	24 min
Totals	40		56 min

Information on the Exam Content

- ▶ Exam questions are distributed across Chapters & K-Levels

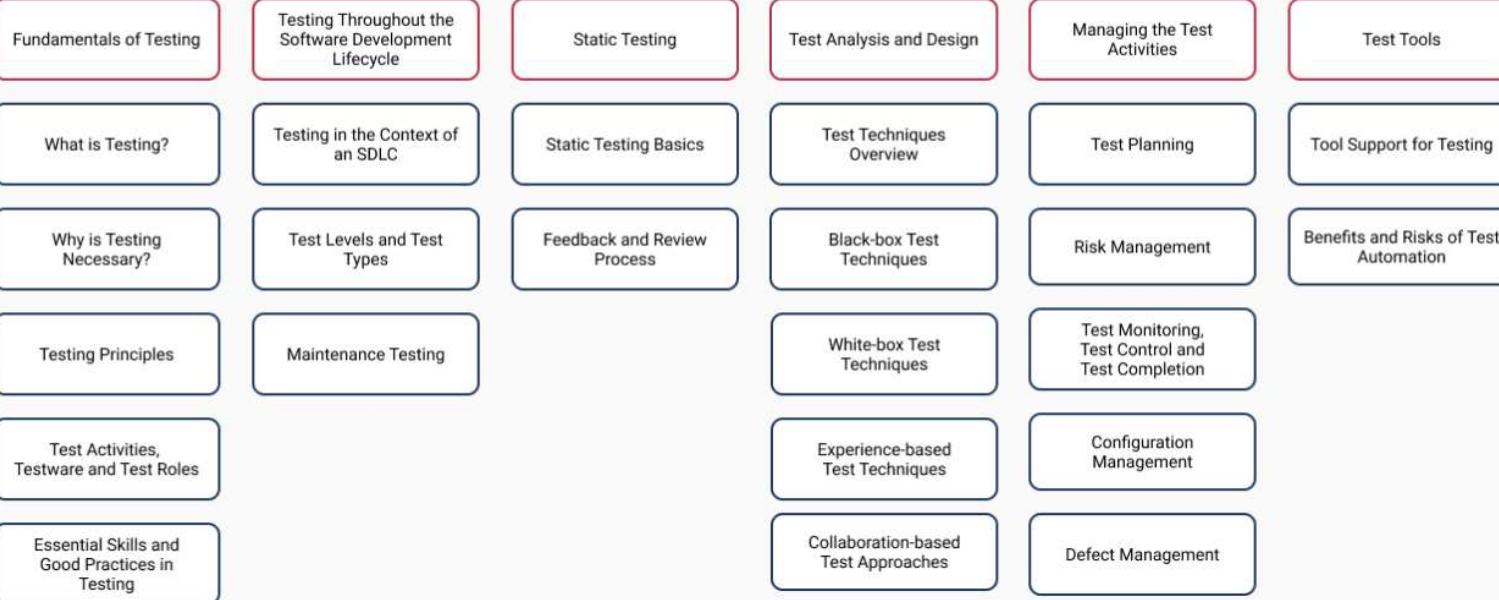
Chapter	K-Level	Nr of Questions
1	K1	2
	K2	6
2	K1	1
	K2	4
3	K1	1
	K2	3
	K3	1

Chapter	K-Level	Nr of Questions
4	K1	1
	K2	5
	K3	5
5	K1	2
	K2	5
	K3	2
6	K1	1
	K2	1



Foundation Level 2023 (V4.0.) - Content

ISTQB® Certified Tester Foundation Level (CTFL)





Fundamentals of Testing

‘What is Testing?’

Agenda

1.1 What is Testing?

- ▶ FL-1.1.1 (K1) Identify typical test objectives
- ▶ FL-1.1.2 (K2) Differentiate testing from debugging

1.2 Why is Testing Necessary?

- ▶ FL-1.2.1 (K2) Exemplify why testing is necessary
- ▶ FL-1.2.2 (K1) Recall the relation between testing and quality assurance
- ▶ FL-1.2.3 (K2) Distinguish between root cause, error, defect, and failure

1.3 Testing Principles

- ▶ FL-1.3.1 (K2) Explain the seven testing principles

Agenda (2)

1.4 Test Activities, Testware and Test Roles

- ▶ FL-1.4.1 (K2) Summarize the different test activities and tasks
- ▶ FL-1.4.2 (K2) Explain the impact of context on the test process
- ▶ FL-1.4.3 (K2) Differentiate the testware that supports the test activities
- ▶ FL-1.4.4 (K2) Explain the value of maintaining traceability
- ▶ FL-1.4.5 (K2) Compare the different roles in testing

1.5 Essential Skills and Good Practices in Testing

- ▶ FL-1.5.1 (K2) Give examples of the generic skills required for testing
- ▶ FL-1.5.2 (K1) Recall the advantages of the whole team approach
- ▶ FL-1.5.3 (K2) Distinguish the benefits and drawbacks of independence of testing

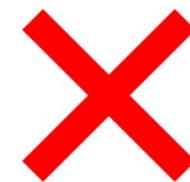
What is Software Testing?

- ▶ The process in the SDLC that evaluates the quality of a component, system or related work products:
 - ▶ Meets the business and technical requirements that guided its design and development
 - ▶ Works as expected
 - ▶ Can be implemented with the same characteristics
- ▶ Software testing:
 - ▶ Assesses the quality of the software
 - ▶ Reduces the risk of software failure in operation



Misconceptions about Software Testing

- ▶ Software testing is a set of activities to discover defects and evaluate the quality of test objects. Misconceptions about testing are that testing:
 - ▶ Only focus on executing tests
 - ▶ Also includes many other activities
 - ▶ Must be aligned with the SDLC
 - ▶ Only focus on verifying the test object
 - ▶ Also involves validation
 - ▶ Can be fully automated
 - ▶ It is an intellectual process and the tester needs:
 - ▶ Specialized knowledge
 - ▶ Use analytical skills
 - ▶ Apply critical thinking
 - ▶ Apply systems thinking
 - ▶ Proves that the software is correct and has no faults



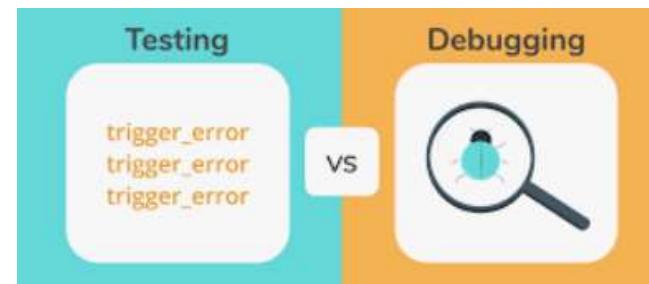
Typical Test Objectives

- ▶ Evaluate work products
- ▶ Triggering failures and finding defects
- ▶ Ensuring required coverage of a test object
- ▶ Reduce the risk of the software not being of high enough quality
- ▶ Verify conformance to requirements
- ▶ Verify compliance contractual, legal or regulatory requirements
- ▶ Provide enough information to stakeholders
- ▶ Build confidence in the quality of the test object
- ▶ Validate whether the test object is complete and works as expected

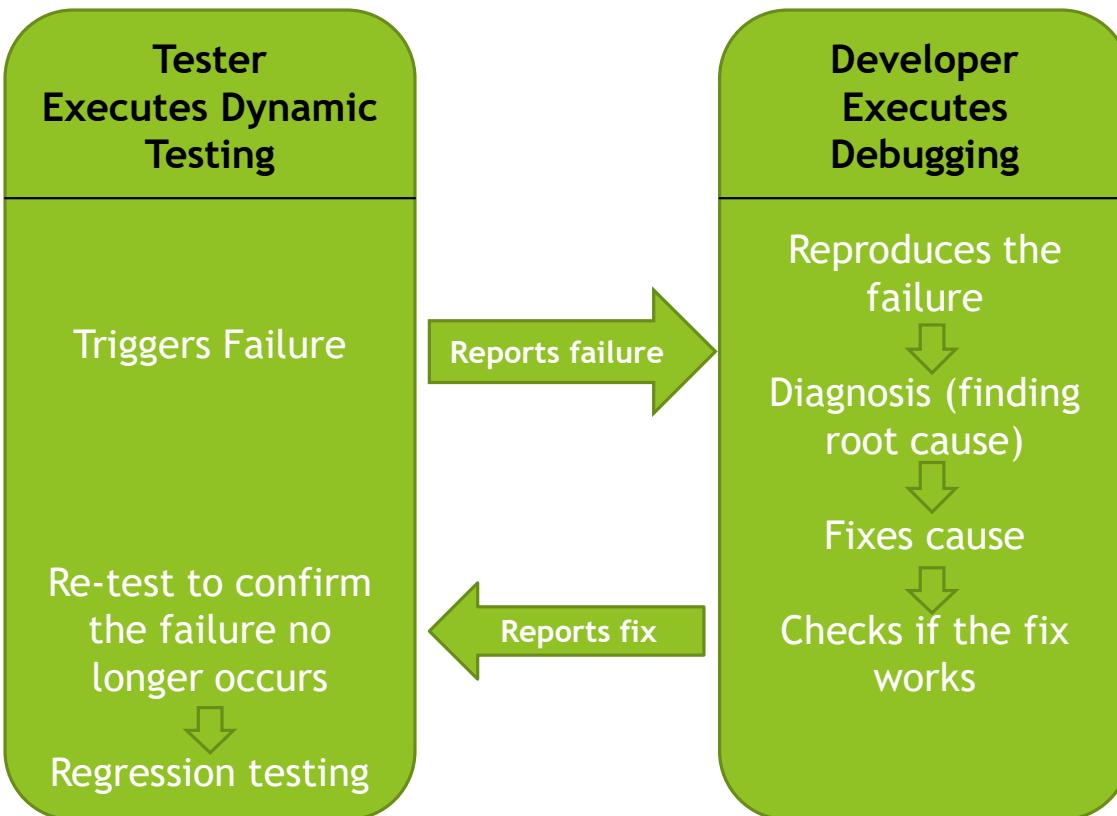


Testing and Debugging

- ▶ **Testing and debugging** are often confused with one another
 - ▶ Testing can:
 - ▶ Trigger failures that are caused by defects (dynamic testing)
 - ▶ Find defects in the test object (static testing)
 - ▶ **Debugging** is an activity that finds, analyzes and eliminates the cause of the failure (the defect) found in **dynamic testing**
 - ▶ **Debugging** a defect found in **static** testing, debugging is simply removing the defect, no reproduction or diagnosis needed.



Dynamic Testing and Debugging - visualized



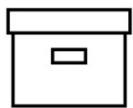


Fundamentals of Testing

‘Why is Testing Necessary?’

Why is testing necessary

- ▶ Helps in achieving agreed upon goals
- ▶ NOT limited to the Test Team
- ▶ Testing Contributes to success
 - ▶ Cost-effective way to find defects
 - ▶ Evaluation of quality leads to well-informed decisions
 - ▶ Testers represent the users in the development project
 - ▶ Sometimes required (contractually, legally, industry-specific)



Testing reduces risk

- ▶ Reduce the **risk** of failures in operation by:
 - ▶ Being involved in reviews & refinements - prevent defects
 - ▶ Working together with designers - prevent design defects and early test identification
 - ▶ Working with developers - increase understanding on code - reduce code and test defects
 - ▶ Verifying and validating before go-live - detect and fix failures and better meet the stakeholder requirements

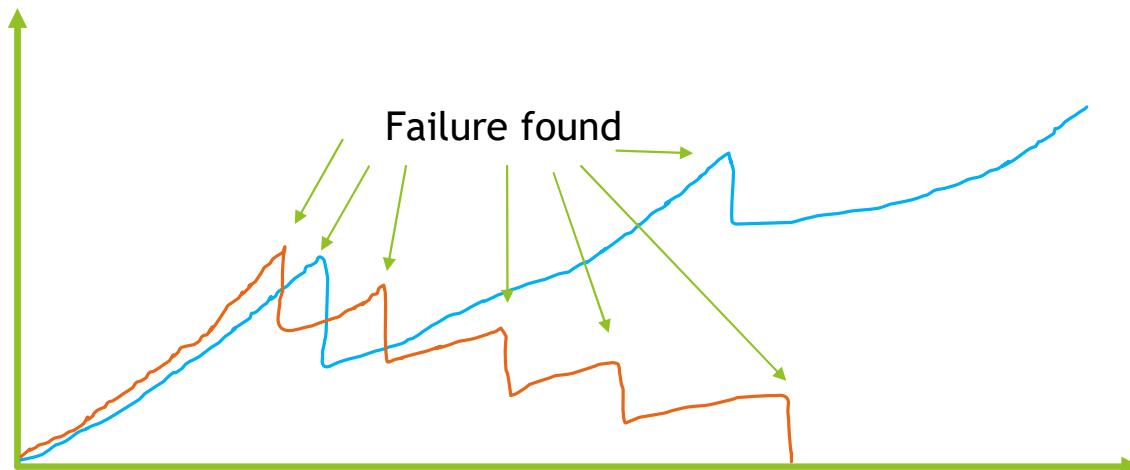


**SOFTWARE
FAILURE**

Some costly software failures

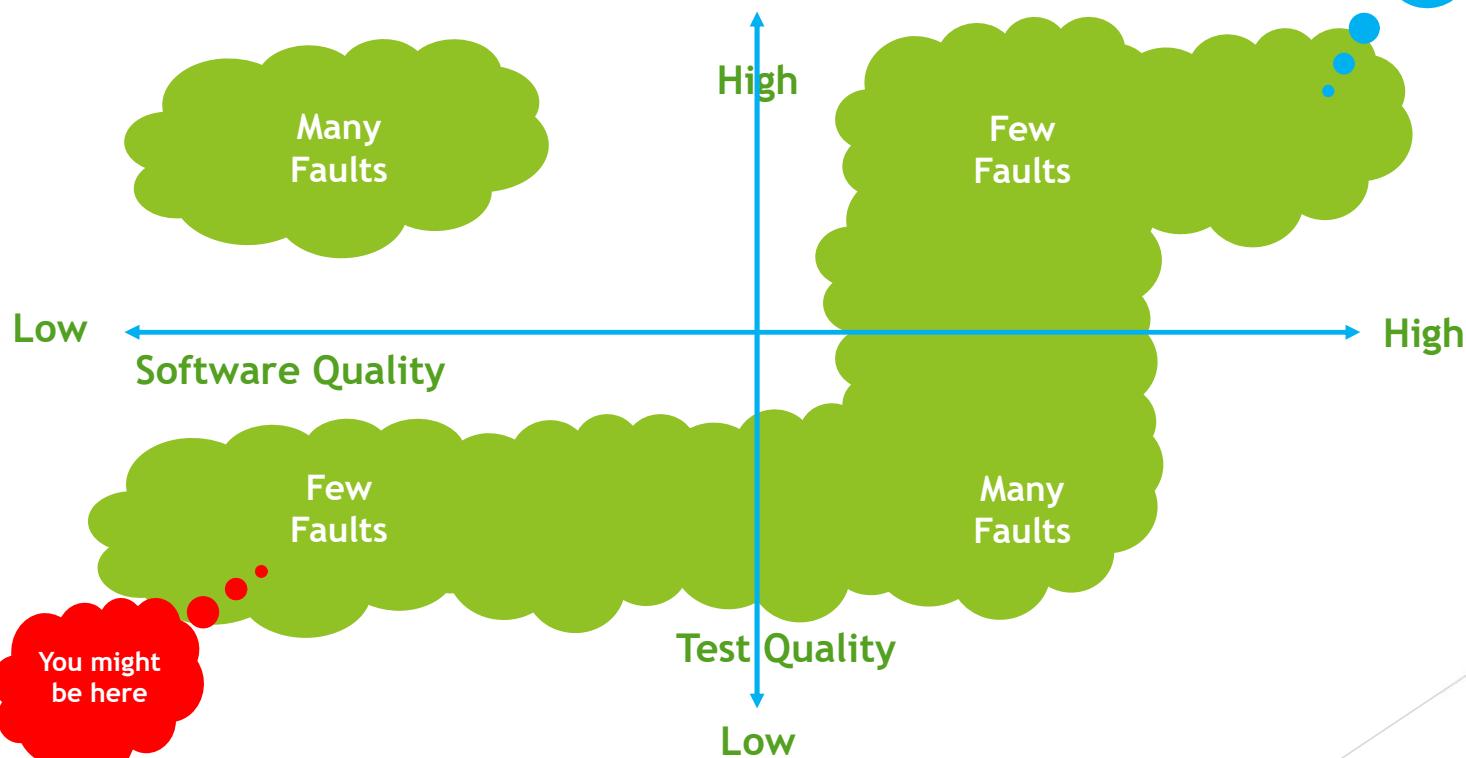
- ▶ **NASA Mars Climate Orbiter**
 - ▶ 1998 - spacecraft lost in space
 - ▶ Cost - 125 million USD
- ▶ **EDS Child Support System**
 - ▶ 2004 - new highly complex IT System
 - ▶ Cost - 1 billion USD and counting
- ▶ **Bitcoin Hack**
 - ▶ 2011 - Compromised security
 - ▶ Cost - 850k Bitcoins lost

Confidence



No failures encountered = confidence?

Assessing software quality



How much testing is enough?

- ▶ it's never enough
- ▶ when you have done what you planned
- ▶ when your customer/user is happy
- ▶ when you have proven that the system works correctly
- ▶ when you are confident that the system works correctly
- ▶ it depends on the **risks** for your system



How much testing is enough - Risk

- ▶ depends on **RISK**
 - ▶ risk of missing important faults
 - ▶ risk of incurring failure costs
 - ▶ risk of releasing untested or under-tested software
 - ▶ risk of losing credibility and market share
 - ▶ risk of missing a market window
 - ▶ risk of over-testing, ineffective testing



Risk based testing

- ▶ Test time will always be limited
- ▶ Use RISK to determine
 - ▶ what to test first
 - ▶ what to test most
 - ▶ how thoroughly to test each item } i.e. where to place emphasis
 - ▶ what not to test (this time)
- ▶ Use RISK to
 - ▶ allocate the time available for testing by prioritizing testing ...



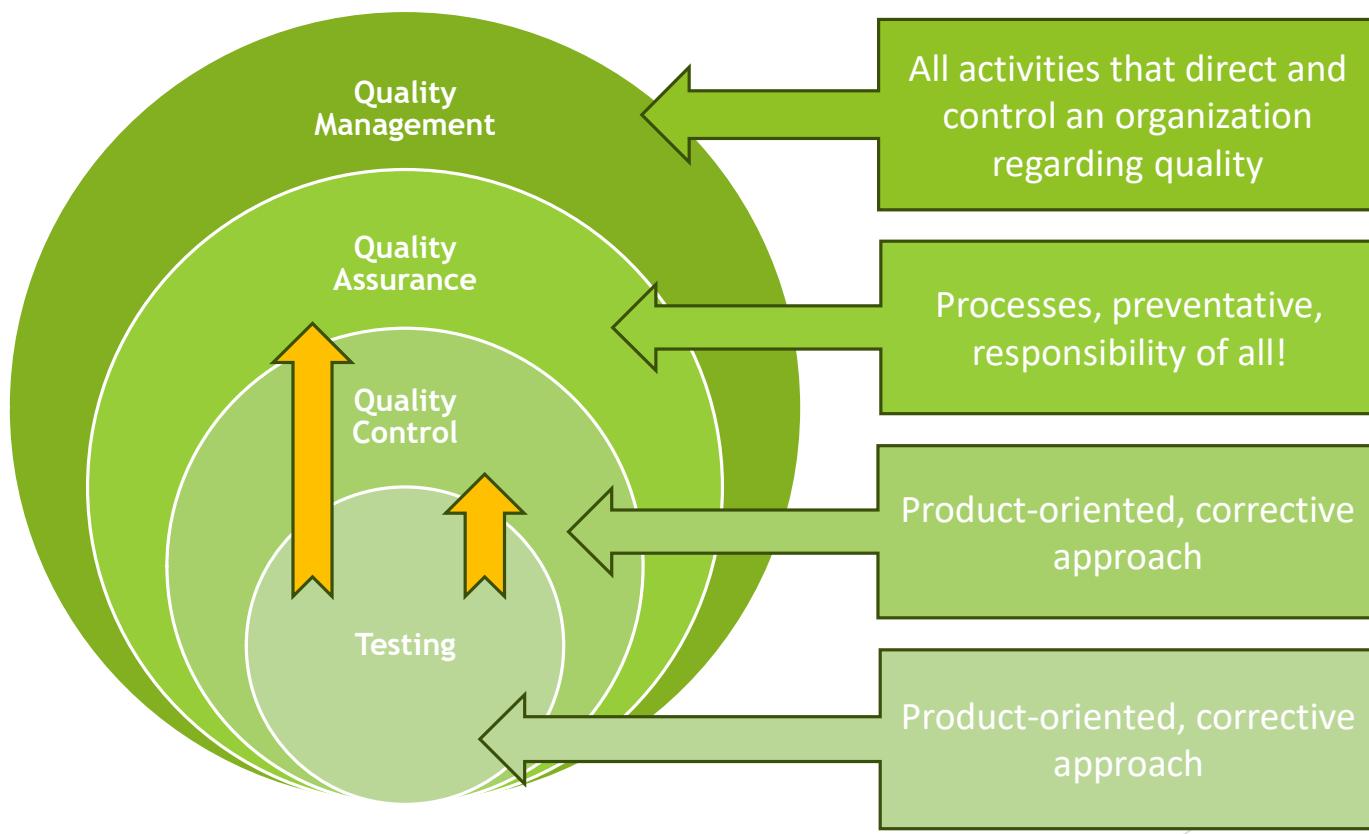
**Prioritize tests so that at any time, whenever you stop testing,
you have done the best testing in the time available.**



Fundamentals of Testing

Testing and Quality Assurance

Quality Assurance and Testing



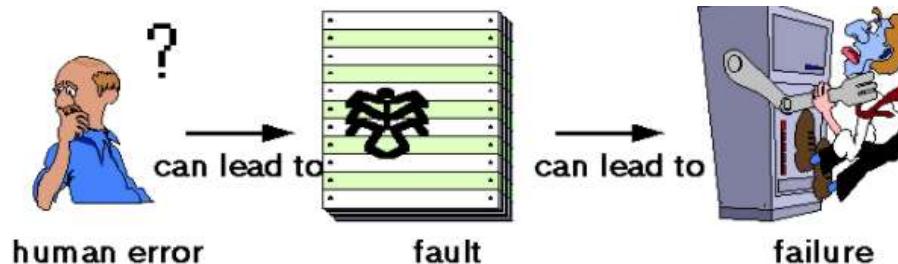


Fundamentals of Testing

Errors, defects, failures
and root causes

Terms explained

- ▶ **Error:** a human action that produces an incorrect result
- ▶ **Defect:** a manifestation of an error in a software work product
 - ▶ also known as a fault or bug
 - ▶ if executed, a defect **may** cause a failure
- ▶ **Failure:** deviation of the software from its expected delivery or service



Why do defects occur?

- ▶ Humans make mistakes
 - ▶ Some of those mistakes are unimportant, but some of them are expensive or dangerous.
 - ▶ Time pressure
 - ▶ complexity of work products, processes, infrastructure, technologies
 - ▶ Miscommunication and misunderstandings
 - ▶ Inexperienced or unskilled staff
- ▶ Environmental Conditions (radiation, pollution, etc)
- ▶ Bad assumptions and blind spots
 - ▶ We might make the same mistakes when we check our own work as we made when we did it.
So we may not notice the flaws in what we have done.
- ▶ Ideally, someone else should check our work because another person is more likely to spot the flaws.



Where do defects occur?

- ▶ Documentation
 - ▶ Requirement specifications
 - ▶ Test scripts
- ▶ Source code
- ▶ Any supporting artifacts

Basically, defects can be found in any stage of the SDLC in any work product, but the earlier they are found, the cheaper they are to fix.

Reliability vs faults

- ▶ **Reliability:** the probability that software will not cause failure of the system for a specified time under specified conditions
 - ▶ Can a system be ‘fault’ free? (Zero faults, right first time) X
 - ▶ Can a software system be reliable with faults? ✓
 - ▶ Is a ‘fault-free’ application always reliable? X
- ▶ Not each unexpected test result is a failure - the test itself can result in:
 - ▶ False negatives
 - ▶ False positives

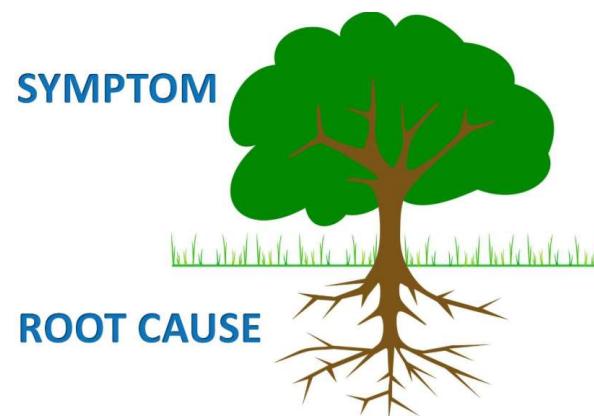
Root cause of defects and their effects

► Root Cause

- ▶ A fundamental reason for the occurrence of a problem
- ▶ Earliest action / condition that created a defect
- ▶ Should be identified to avoid similar defects
- ▶ Root Cause Analysis can lead to process improvements to prevent defects

► Identifying the Root Cause:

- ▶ 5 why's





Fundamentals of Testing

Seven Testing Principles

Seven Testing Principles

- ▶ **Testing shows presence of defects, not their absence**
 - ▶ 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.
- ▶ **Exhaustive testing is impossible**
 - ▶ Testing everything (all combinations of inputs and preconditions) is not feasible. Instead, risk analysis and priorities should be used to focus testing efforts

Exhaustive testing - Example

- ▶ Exhaustive testing is exercising **all** combinations of inputs and preconditions. This will take an impractical amount of time.

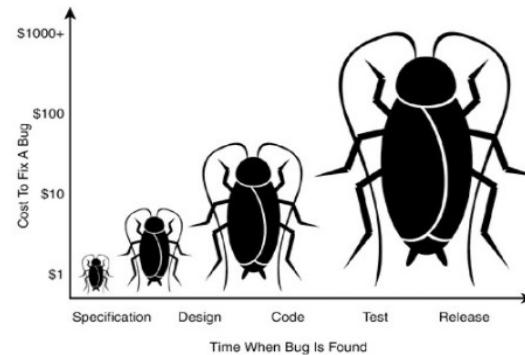
Example:

- ▶ a simple 20 screen application with 3 menus and 3 options per menu
- ▶ There are 12 fields per screen, 2 types of input (number as integer or decimal) and about 120 possible values.
- ▶ How many tests would be needed for exhaustive testing?
- ▶ Total tests for exhaustive testing: $20 \times 3 \times 3 \times 12 \times 2 \times 120 = 518.400$ tests
- ▶ If each test takes 1 second, exhaustive testing takes 8.640 minutes, 144 hours, 18 working days

Seven Testing Principles (2)

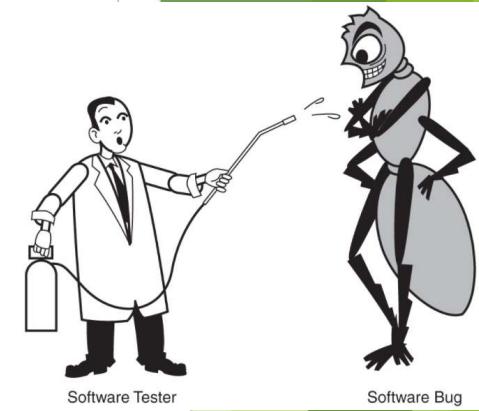
- ▶ **Early testing saves time and money**
 - ▶ To find defects early, testing activities should be started as early as possible in the software or system development life cycle, and should be focused on defined objectives

- ▶ **Defect cluster together**
 - ▶ Testing effort shall be focused proportionally to the expected and later observed defect density of modules



Seven Testing Principles (3)

- ▶ **Pesticide Paradox - Tests wear out**
 - ▶ If the same tests are repeated over and over again, they will no longer find any new defects. To overcome this test cases need to be reviewed and revised, to exercise different parts of the software
- ▶ **Testing is context dependent**
 - ▶ Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site and Agile testing differs from sequential testing
- ▶ **Absence-of-errors fallacy**
 - ▶ Finding and fixing defects does not help if the system built is unusable and does not fulfill the users needs and expectations. Besides verification, validation should also be done.





Fundamentals of Testing

Test Activities, Testware and Test Roles

Test Process

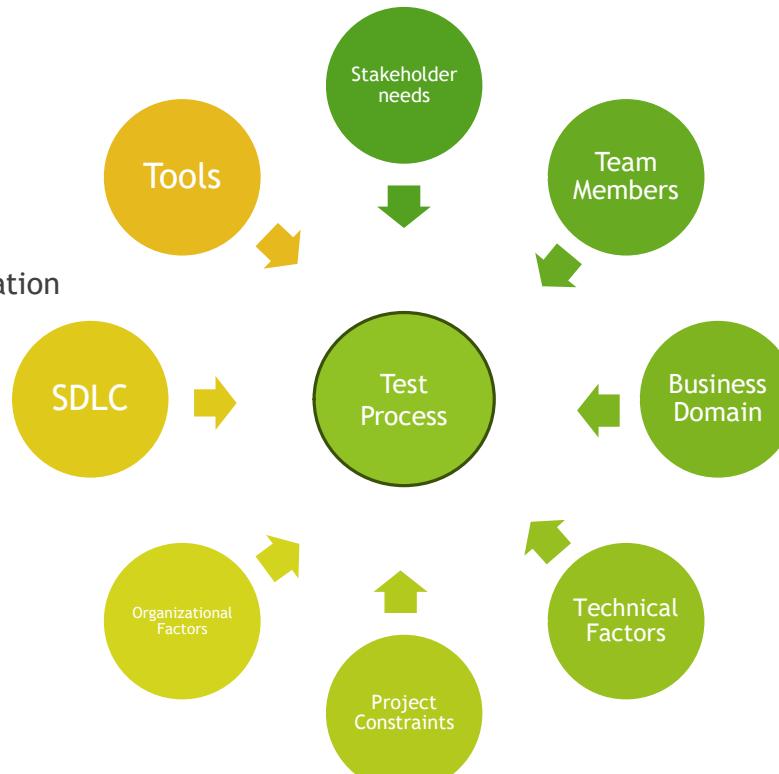
- ▶ Sets of common test activities
- ▶ Test strategy should state the proper test process followed
- ▶ Not having common test activities - less likely to reach the test objectives
- ▶ Decisions on Test Activities are made in Test Planning
- ▶ Test activities do not stand alone
- ▶ In order to be effective, the need to be fully integrated with development activities



The essential goal is not to find bugs, but to help fulfill the stakeholder's business needs; the reason for this is that the stakeholders are paying!

Test Process in Context

- ▶ Test Strategy
- ▶ Test Techniques used
- ▶ Degree of Test Automation
- ▶ Required Level of Coverage
- ▶ Level of Detail of Test Documentation
- ▶ Reporting

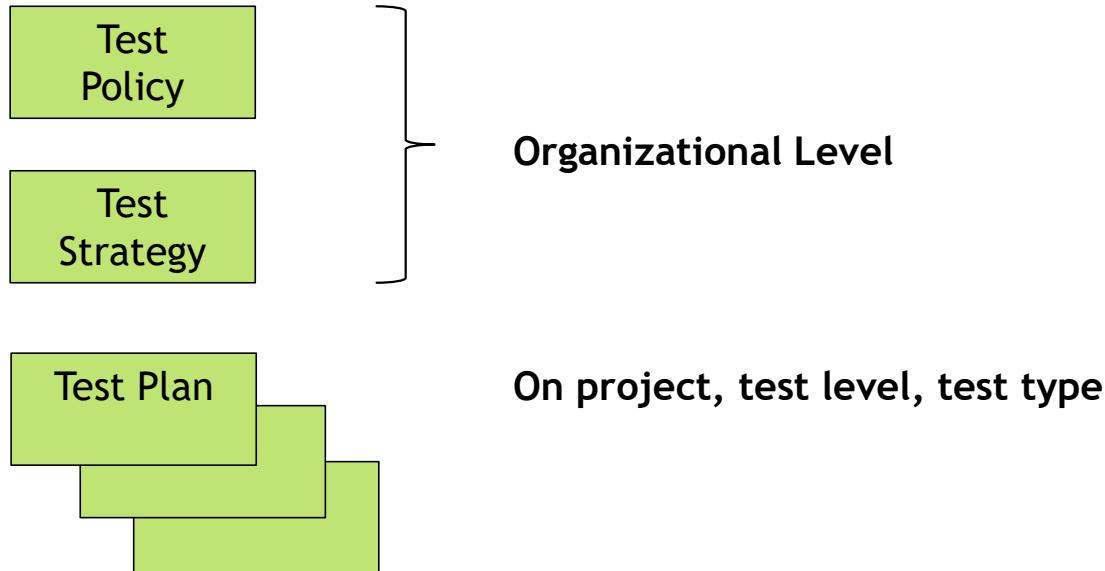


Testware

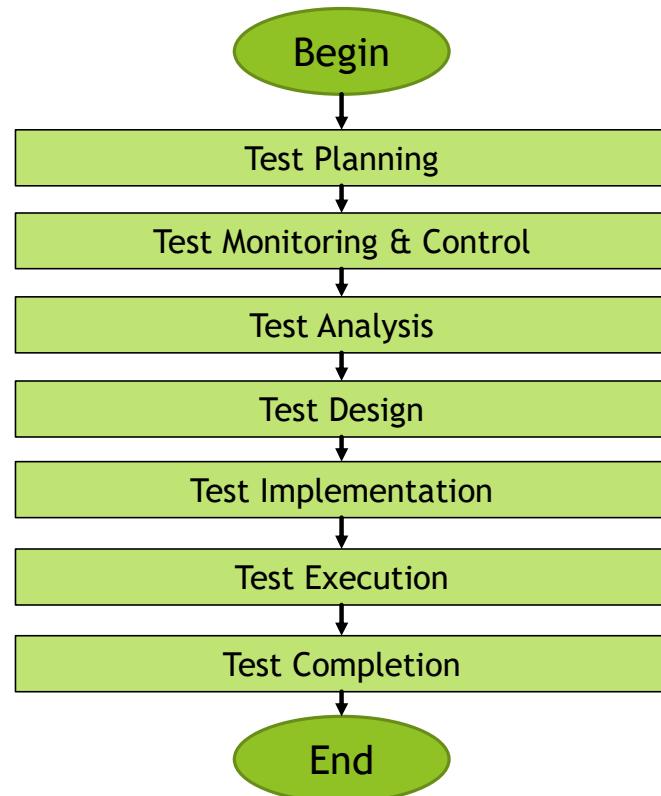
- ▶ Are created as part of the test activities in the test process
- ▶ Each test activity creates test work products (aka testware)
- ▶ Types of work products vary as the test process varies
- ▶ Much testware can be captured and managed using:
 - ▶ Test Management Tools
 - ▶ Defect Management Tools

Proper Configuration Management should be in place
to ensure integer and consistent testware

Fundamental Testware



Fundamental Test Process



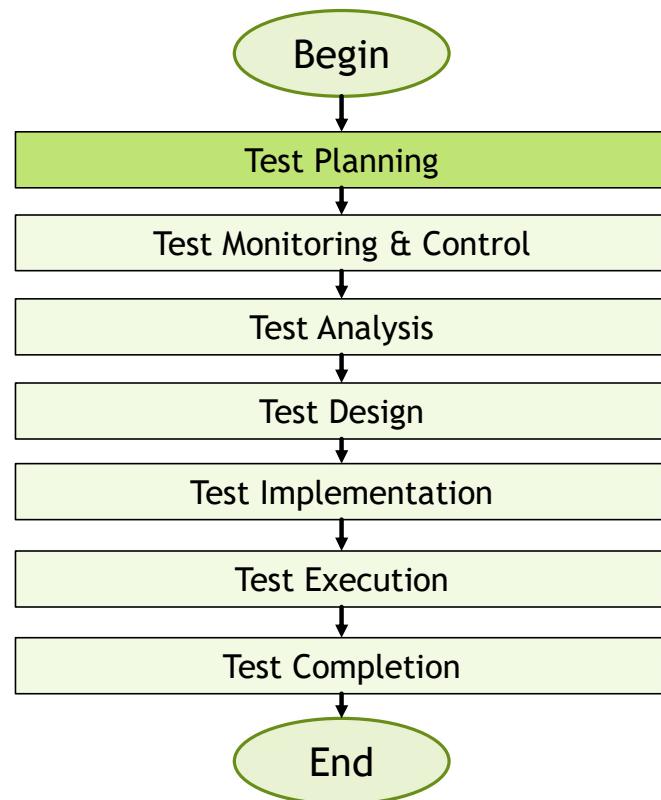


Fundamentals of Testing

Test Activities, Testware

Test Planning

Test Planning - Activities, Tasks, Testware



Test Planning - Test Activities and Tasks

- ▶ Defining the objectives of testing
- ▶ Selecting the approach that achieves the objectives best
- ▶ Starts at the beginning of the software development project
- ▶ Must regularly be checked and may be updated and adjusted based on feedback from the monitoring & control activities



Test Planning - Testware

- ▶ **Test Plan(s)**
- ▶ Includes information on:
 - ▶ Test Basis
 - ▶ Traceability information (from the test basis to other work products)
 - ▶ Exit Criteria (that are used in test monitoring and control)



Test Plan - ISO/IEC/IEEE 29119 Standard

- ▶ Context of testing
 - ▶ Type of Test Plan
 - ▶ Test Items
 - ▶ Test Scope
 - ▶ Assumptions and Constraints
 - ▶ Stakeholders
- ▶ Testing Lines of Communication
- ▶ Risk Register (Risks)
- ▶ Testing activities and estimates
- ▶ Staffing
 - ▶ Roles, tasks, responsibilities
 - ▶ Training needs
 - ▶ Hiring needs
- ▶ Schedule
- ▶ Test Strategy (of the project)
 - ▶ Test Sub-Processes
 - ▶ Test Deliverables
 - ▶ Test Design Techniques
 - ▶ Test completion criteria
 - ▶ Metrics to be collected
 - ▶ Test Data Requirements
 - ▶ Test Environment Requirements
 - ▶ Retesting and regression testing
 - ▶ Suspension and resumption criteria
 - ▶ Deviations from the organizational test strategy

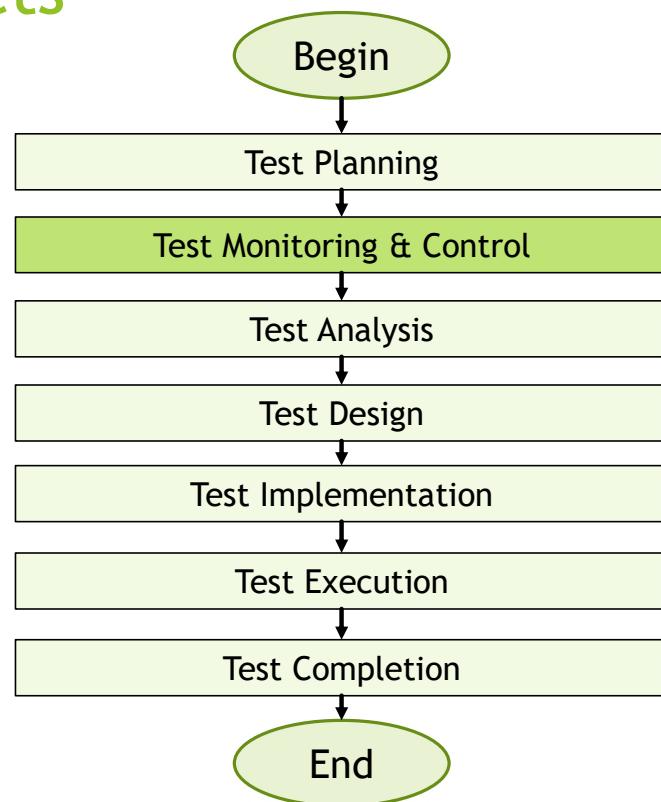


Fundamentals of Testing

Test Activities, Testware

Test Monitoring & Control

Test Monitoring & Control - Activities, Tasks, Work Products



Test Monitoring & Control - Test Activities and Tasks

- ▶ Test Monitoring
 - ▶ ONGOING checking of all test activities
 - ▶ ONGOING comparison of actual progress against the test plan using metrics
- ▶ Test Control
 - ▶ Taking actions needed to meet the objectives of the testing
 - ▶ Updating the Test Plan according to feedback
- ▶ Monitoring and Control are supported by the evaluation of exit criteria (definition of done), which may include:
 - ▶ Checking of test results and logs against coverage criteria
 - ▶ Assessing component or system quality based on the test results and logs
 - ▶ Determining if more tests are needed
- ▶ Progress, deviations and information needed to determine to stop testing is reported in **Test Progress Reports, Documentation of Control Directives and Risk information**



Test Monitoring & Control - Testware

- ▶ Various types of **Test Reports** such as:
 - ▶ Test Progress Reports
 - ▶ Test Summary Reports
 - ▶ Documentation of Control Directives
 - ▶ Risk Information
- ▶ These reports provide relevant information on the current test progress and a summary of the test execution results



Test Progress Report - ISO/IEC/IEEE 29119 Standard

- ▶ Reporting period
- ▶ Progress against the test plan
- ▶ Factors blocking progress
- ▶ Test Measures
- ▶ New and changed risks
- ▶ Planned testing

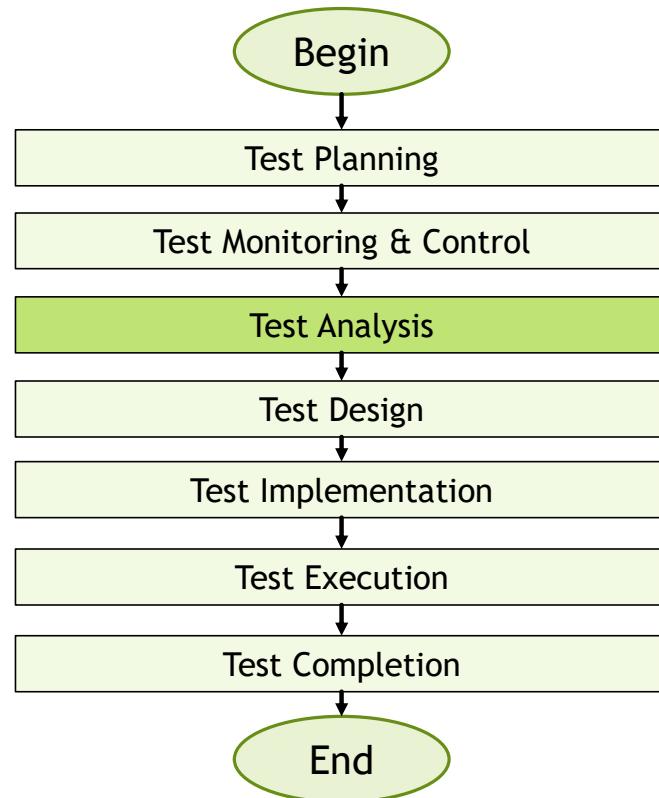


Fundamentals of Testing

Test Activities, Testware

Test Analysis

Test Analysis - Activities, Tasks, Work Products



Test Analysis - Test Activities and Tasks

- ▶ Answers 'what to test?' using measurable coverage criteria
- ▶ Major activities are:
 1. **Analyzing the Test Basis** to:
 - ▶ Identify testable features
 - ▶ Test bases include Requirement Specifications, Design and Implementation information, implementation of the component itself, Risk Analysis reports, etc



Test Analysis - Test Activities and Tasks (2)

2. **Evaluating** the Test Basis and test objects to identify defects and their testability such as:

- ▶ Ambiguities
- ▶ Omissions
- ▶ Inconsistencies
- ▶ Inaccuracies
- ▶ Contradictions
- ▶ Superfluous statements



Test Analysis - Test Activities and Tasks (3)

3. **Defining & Prioritizing** test conditions, their risks and risk levels per feature based on:

- ▶ Analysis of the Test Basis
- ▶ Consideration on functional, non-functional and structural characteristics
- ▶ Business and technical factors
- ▶ Levels of risk

Black Box, White Box and Experience Based test techniques can be useful in Test Analysis to define more precise and accurate test conditions and not miss important test conditions



Test Analysis - Testware

- ▶ Defined and prioritized test conditions (such as Acceptance Criteria)
- ▶ Defect reports regarding defects in the test bases



Test Design Specification - ISO/IEC/IEEE 29119 Standard

- ▶ Feature Sets
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Objective
 - ▶ Priority
 - ▶ Specific strategy
 - ▶ Traceability
- ▶ Test Conditions
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Description
 - ▶ Priority
 - ▶ Traceability

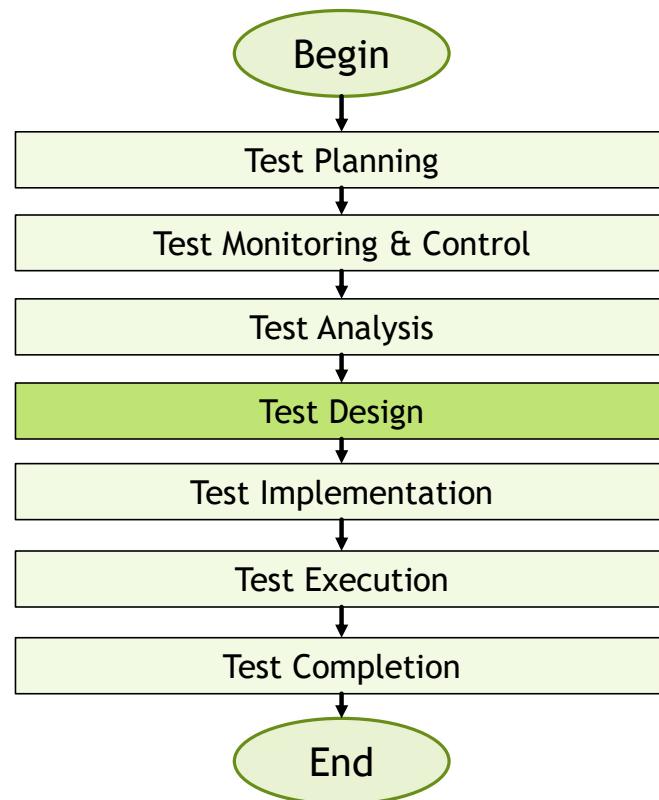


Fundamentals of Testing

Test Activities, Testware

Test Design

Test Design - Activities, Tasks, Work Products



Test Design - Test Activities and Tasks

- ▶ The test conditions are used to create high level test cases and other testware, often involving test design techniques
- ▶ Answers 'how to test?'
- ▶ Includes the following major activities
 - ▶ Identify coverage items
 - ▶ Designing test cases and sets of test cases
 - ▶ Defining the test data requirements to support test cases
 - ▶ Designing the test environment
 - ▶ Identifying required infrastructure and tools
- ▶ Identification of defects during Test Design is a potential benefit



A good test case

- ▶ **Effective** —————— FInds faults
- ▶ **Exemplary** —————— Represents others
- ▶ **Evolvable** —————— Easy to maintain
- ▶ **Economic** —————— Cheap to Use

Abstract (logical) vs Concrete test cases

- ▶ Example:
 - ▶ Application for calculating end-of-year bonuses
 - ▶ Bonuses depend on the time employees work for the company

Time in company	Bonus
Less than 3 years	0 %
More than 3 years	50 %
More than 5 years	75 %
More than 8 years	100 %

Abstract (logical) vs Concrete test cases (2)

Abstract/Logical test case

Test Case nr	Input x (time in company)	Bonus
1	$x \leq 3$	0
2	$3 < x \leq 5$	50
3	$5 < x \leq 8$	75
4	$x > 8$	100

Concrete test case

Test Case nr	Input x (time in company)	Bonus
1	2	0
2	4	50
3	7	75
4	12	100



Test Design - Testware

- ▶ Prioritized test cases and sets of test cases
- ▶ Other work products may include:
 - ▶ Test Charters
 - ▶ Coverage items
 - ▶ Test environment requirements
 - ▶ Test data requirements
- ▶ Good practice to first design high level (abstract) test cases:
 - ▶ Without concrete values for input and expected output
 - ▶ Can be reused across multiple test cycles with different data
- ▶ Each test case should be bi-directionally traceable to the test conditions it covers

Test Case Specification - ISO/IEC/IEEE 29119 Standard

- ▶ Test Coverage Items
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Description
 - ▶ Priority
 - ▶ Traceability
- ▶ Test Cases
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Objective
 - ▶ Priority
 - ▶ Traceability
 - ▶ Preconditions
 - ▶ Inputs
 - ▶ Expected results
 - ▶ Actual results

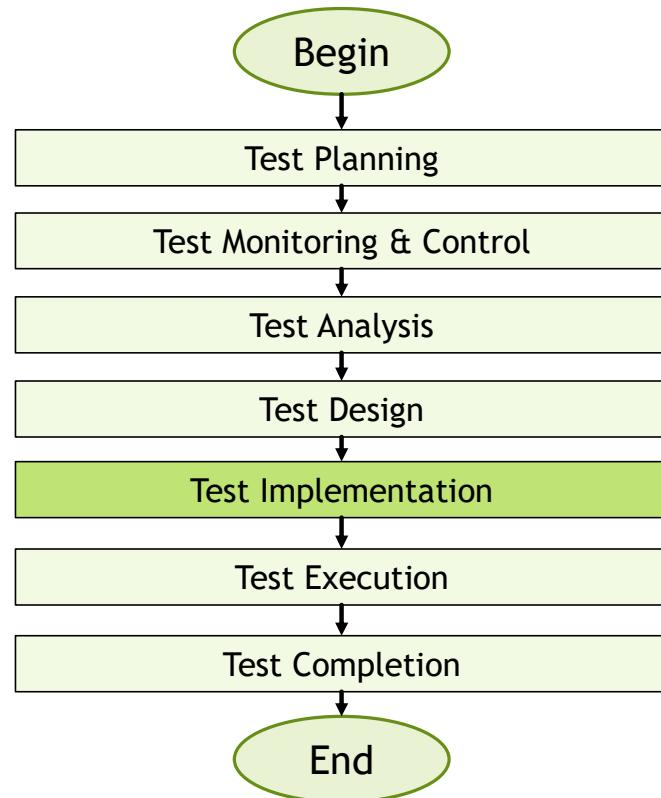


Fundamentals of Testing

Test Activities, Testware

Test Implementation

Test Implementation - Activities, Tasks, Testware



Test Implementation - Test Activities and Tasks

- ▶ In the test implementation phase, testware necessary for test execution is created /acquired
- ▶ ‘Do we now have everything in place to run the tests?’

Ideally, all these tasks are completed before test execution as otherwise precious test execution time might be lost on test implementation tasks.



Test Implementation - Test Activities and Tasks (2)

- ▶ Includes the following major activities:
 - ▶ Organizing test cases into prioritized **test procedures**
 - ▶ Creating **test suites** needed for the test execution
 - ▶ Arranging the test procedures within a test execution schedule for efficient execution
 - ▶ Creating manual and automated test scripts
 - ▶ Building and verifying the test environment
 - ▶ Preparing test data and ensuring it is properly loaded into the test environment
 - ▶ Verifying and updating bi-directional traceability between the test bases, test conditions, test cases, test procedures and test steps



Test Implementation - Testware

- ▶ Testware coming from the test implementation phase include:
 - ▶ Test procedures
 - ▶ Manual and automated test scripts
 - ▶ Test suites
 - ▶ Test execution schedule
 - ▶ Test Data
 - ▶ Test environment elements, for example:
 - ▶ Stubs
 - ▶ Drivers
 - ▶ Simulators
 - ▶ Service virtualizations



Test Procedure Specification - ISO/IEC/IEEE 29119 Standard

- ▶ Test Sets
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Objective
 - ▶ Priority
 - ▶ Contents - Traceability
- ▶ Test Procedures
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Objective
 - ▶ Priority
 - ▶ Start-up
 - ▶ Test cases to be executed (Traceability)
 - ▶ Relationship to other procedures
 - ▶ Stop & wrap-up

Test Data Requirements - ISO/IEC/IEEE 29119 Standard

- ▶ Overview
- ▶ Unique identifier
- ▶ Description
- ▶ Responsibility
- ▶ Period needed
- ▶ Resetting needs
- ▶ Archiving or disposal

Test Environment Requirements - ISO/IEC/IEEE 29119 Standard

- ▶ Overview
- ▶ Unique identifier
- ▶ Description
- ▶ Responsibility
- ▶ Period needed

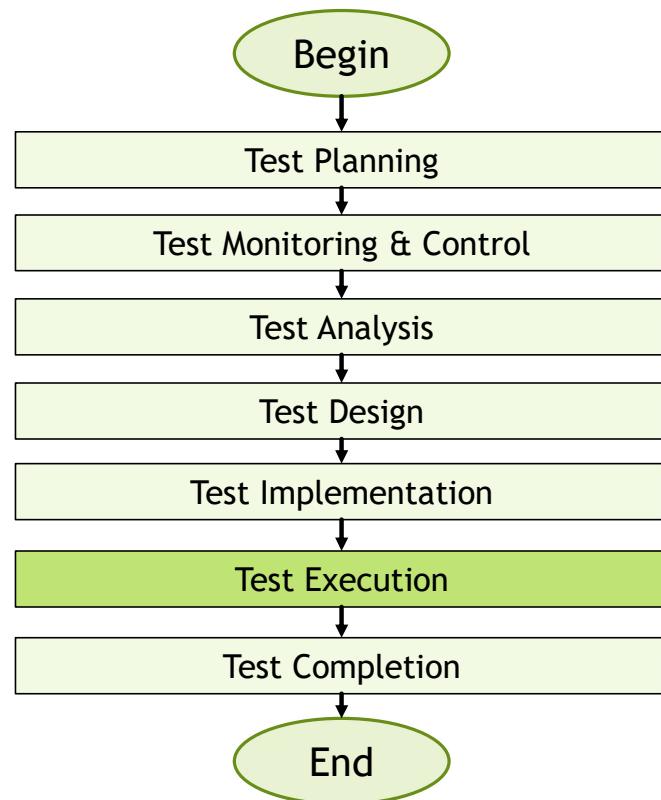


Fundamentals of Testing

Test Activities, Testware

Test Execution

Test Execution - Activities, Tasks, Testware



Test Execution - Test Activities and Tasks

- ▶ Tests are run in accordance with the test execution schedule
- ▶ Includes the following major activities:
 - ▶ Executing tests in the planned sequence (manual or automated)
 - ▶ Comparing actual with expected results and log anomalies as incidents
 - ▶ Analyzing anomalies to find out their likely causes (defects, false positives, false negatives)
 - ▶ Logging the test results (pass, fail, blocked, etc)
 - ▶ Repeating test activities when incidents are solved (confirmation test, run corrected test, regression test)



Test Execution - Testware

- ▶ Work products include:
 - ▶ Test Logs
 - ▶ Defect reports
- ▶ Status of each element of the test basis can be determined and reported on via bi-directional traceability with the associated test procedures



Test Execution Log - ISO/IEC/IEEE 29119 Standard

- ▶ Overview
- ▶ Unique identifier
- ▶ Time
- ▶ Description
- ▶ Impact

Incident Report - ISO/IEC/IEEE 29119 Standard

- ▶ Overview
- ▶ Timing information
- ▶ Originator
- ▶ Context
- ▶ Description of the incident
- ▶ Originator's assessment of the severity
- ▶ Originator's assessment of the priority
- ▶ Risk
- ▶ Status of the incident

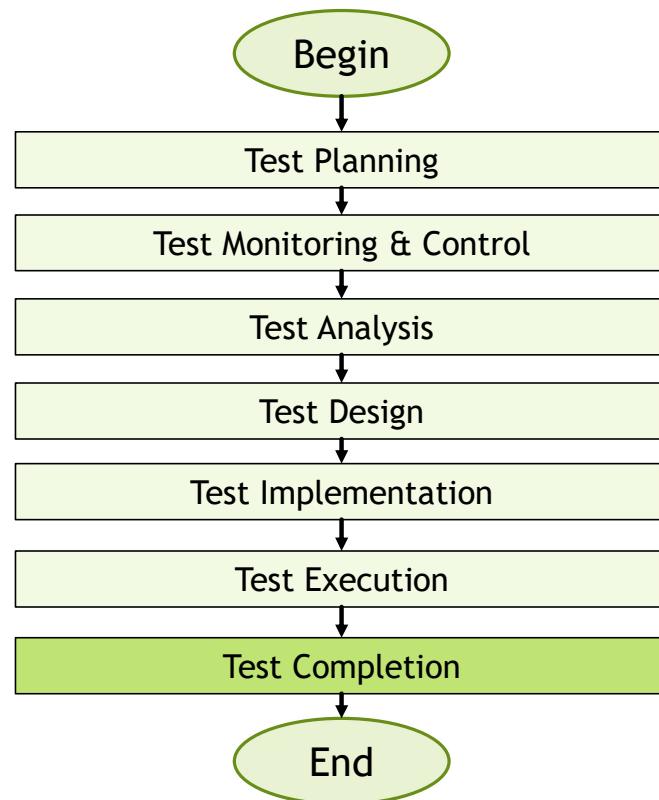


Fundamentals of Testing

Test Activities, Testware

Test Completion

Test Completion - Activities, Tasks, Testware



Test Completion - Test Activities and Tasks

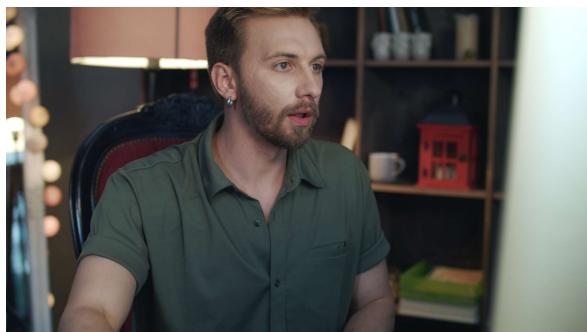
- ▶ Collect data from completed test activities to consolidate experience, testware and other relevant information
- ▶ Includes the following major activities:
 - ▶ Checking whether all defect reports are closed, entering change requests or Product Backlog items for any unsolved defects
 - ▶ Archiving or handing over the testware to the maintenance teams, other project teams or stakeholders
 - ▶ Shutting down the test environment to an agreed state
 - ▶ Analyzing lessons learned from the completed test activities to determine changes needed
 - ▶ Creating a test completion report and communicate to the stakeholders
- ▶ Activities occur at project milestones such as release, completion or cancellation of the test project, an Agile project iteration is finished, a test level is completed or a maintenance release has been completed

Test Completion - Testware

- ▶ Work products include:
 - ▶ Test Completion Report
 - ▶ Action items for improvement
 - ▶ Documented lessons learned
 - ▶ Change requests or Product Backlog Items

Test Completion Report - ISO/IEC/IEEE 29119 Standard

- ▶ Introduction
 - ▶ Scope
 - ▶ References
 - ▶ Glossary
- ▶ Testing performed
 - ▶ Summary of testing performed
 - ▶ Deviations from planned testing
 - ▶ Test completion evaluation
 - ▶ Factors that blocked progress
 - ▶ Test measures
 - ▶ Residual risks
 - ▶ Test deliverables
 - ▶ Reusable test assets
 - ▶ Lessons learned





Fundamentals of Testing

Traceability

Traceability between Test Basis and Test Work Products

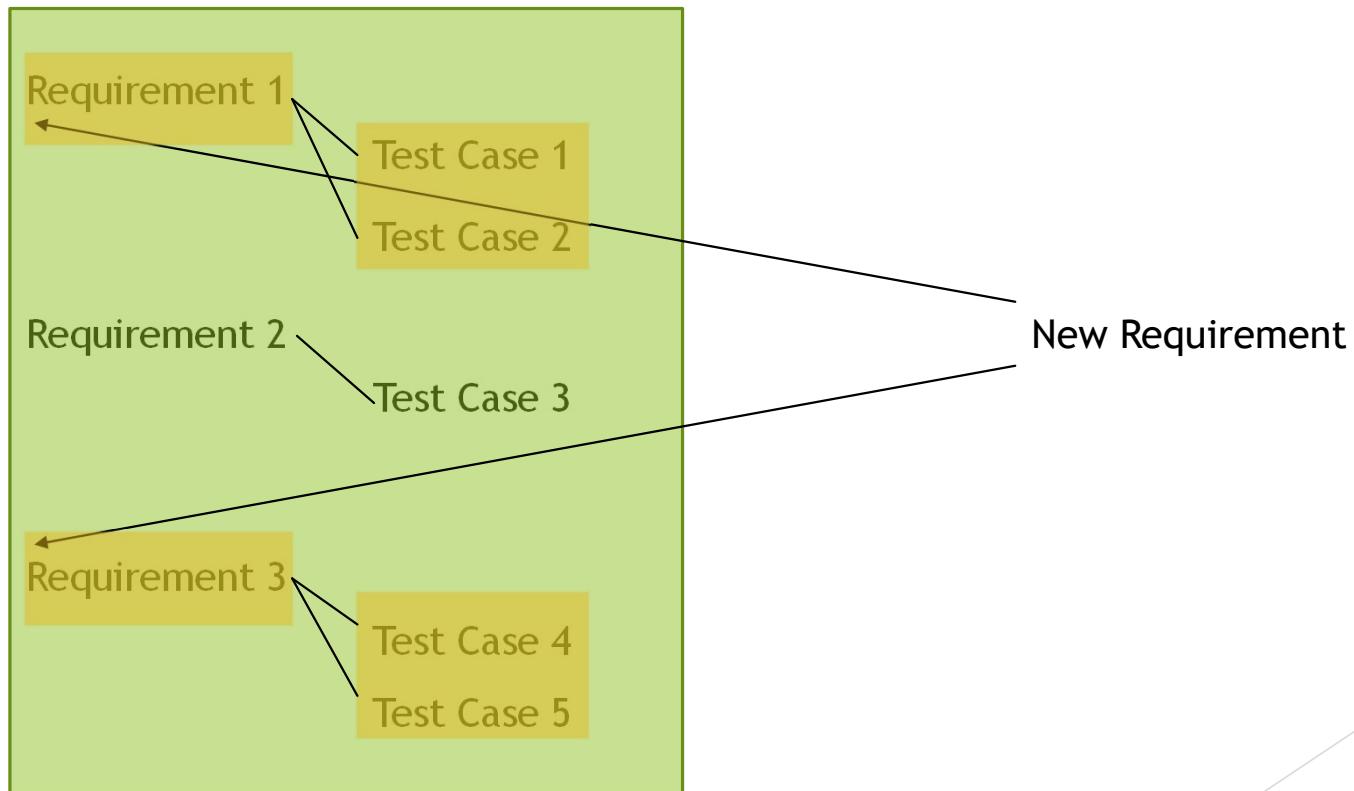
- ▶ Traceability between each element of the test basis and testware associated with these elements is needed for effective test monitoring and control.
- ▶ Traceability enables:
 - ▶ The evaluation of test coverage
 - ▶ Traceability of test cases and requirements - verify that requirement are covered by test cases
 - ▶ Traceability of test cases and risks - evaluate level of residual risk in a test object
 - ▶ Analyzing the impact of changes
 - ▶ Facilitate test audits
 - ▶ Meeting IT governance criteria
 - ▶ Improved clarity of test progress reports and test summary reports
 - ▶ Communicating technical testing aspects to stakeholders in an understandable way
 - ▶ Providing information to assess product quality, product capability and project progress against business goals



Traceability - Example

- ▶ Assume a highly complex, existing system with millions of lines of code
- ▶ 1 module (X) deals with complex algorithms, forecasting events on aircraft
- ▶ Executing the full regression suite for Module X takes 1 day, for all modules (A, B, C and X) it takes 5 days
- ▶ Information from the ‘base’ modules A, B and C is used in this module X
- ▶ Some information from Module X is used in the base modules
- ▶ Without **traceability**, with any change in any of these modules, all would need to be tested with the entire regression suite
- ▶ Traceability enables narrowing down the effect of a change so that no unnecessary tests are done.

Traceability - Example 2





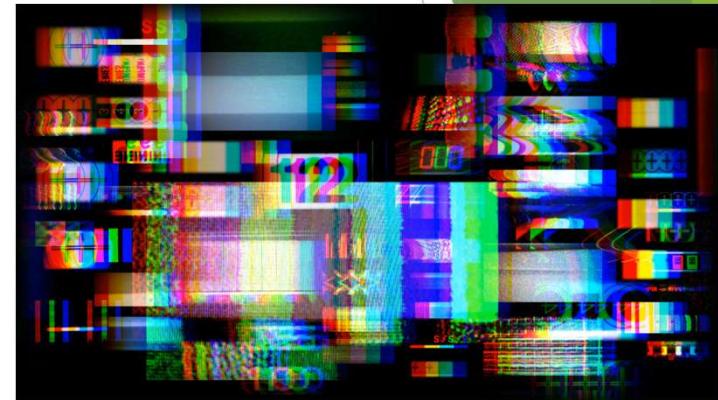
Fundamentals of Testing

Test Activities, Testware

Roles in Testing

Roles in Testing - General

- ▶ Activities and tasks per role depend on many factors:
 - ▶ The project
 - ▶ Product context
 - ▶ Skills of the people who are in the test management and testing roles
 - ▶ The organization



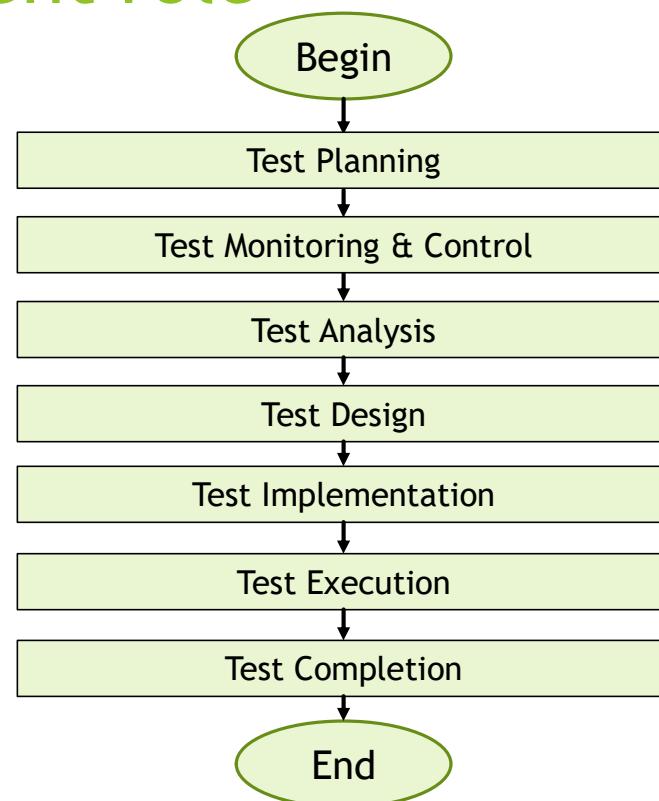
Test Management role

Overall responsibility for:

- ▶ Test Process
- ▶ Test team
- ▶ Leadership of test activities

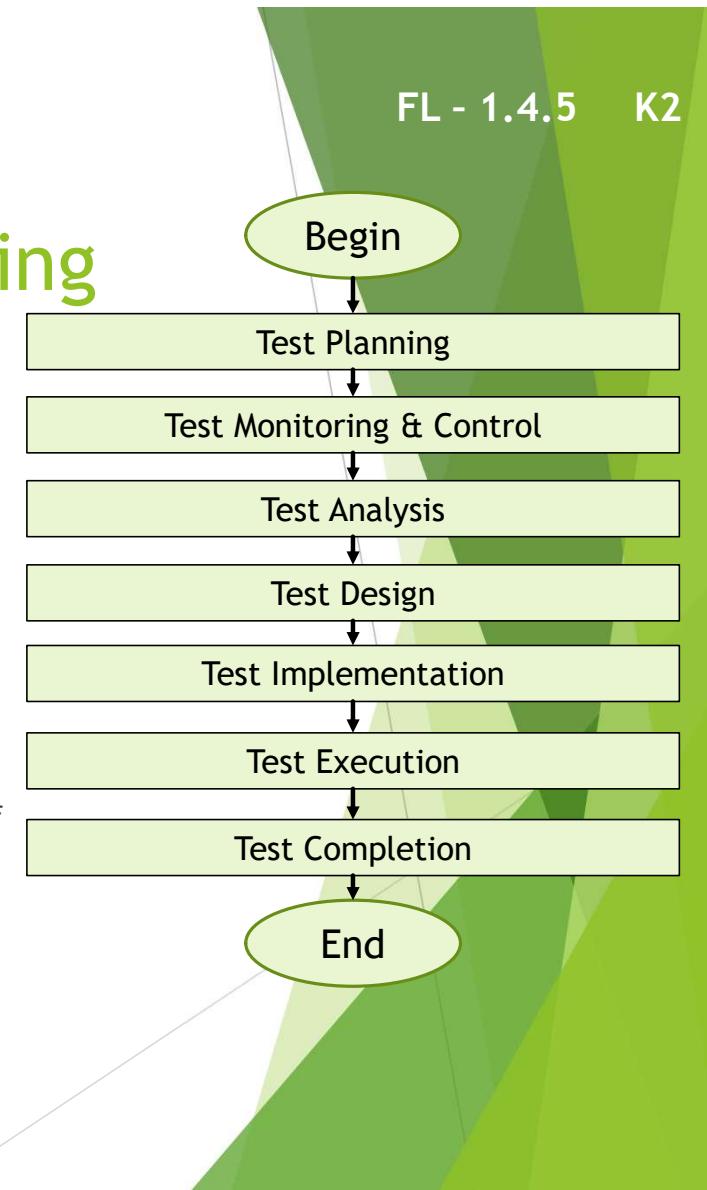
Main focus on:

- ▶ Test Planning
- ▶ Test Monitoring and Control
- ▶ Test Completion



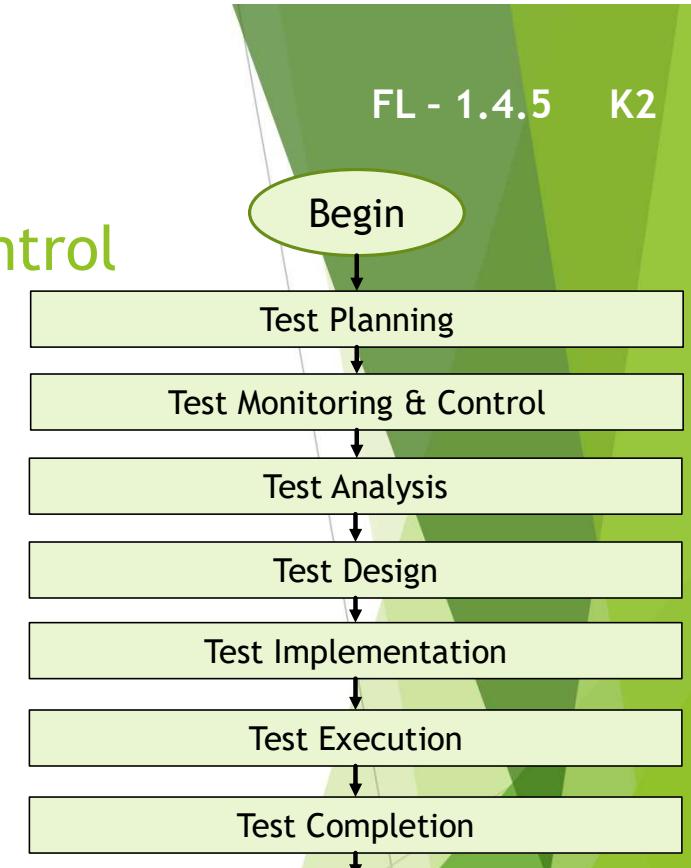
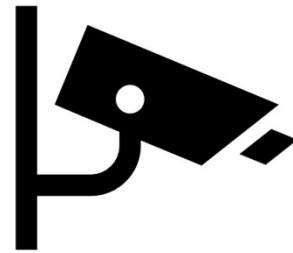
Test Management role - Test Planning

- ▶ Develop or review a test policy and test strategy for the organization
- ▶ Plan the test activities
- ▶ Write and update the test plan(s)
- ▶ Coordinate the test plan(s) with stakeholders
- ▶ Share testing perspective with other project activities such as integration planning
- ▶ Initiate the analysis, design, implementation and execution phases
- ▶ Introduces suitable metrics for measuring test progress and evaluating the quality of testing and product



Test Management role - Test Monitoring & Control

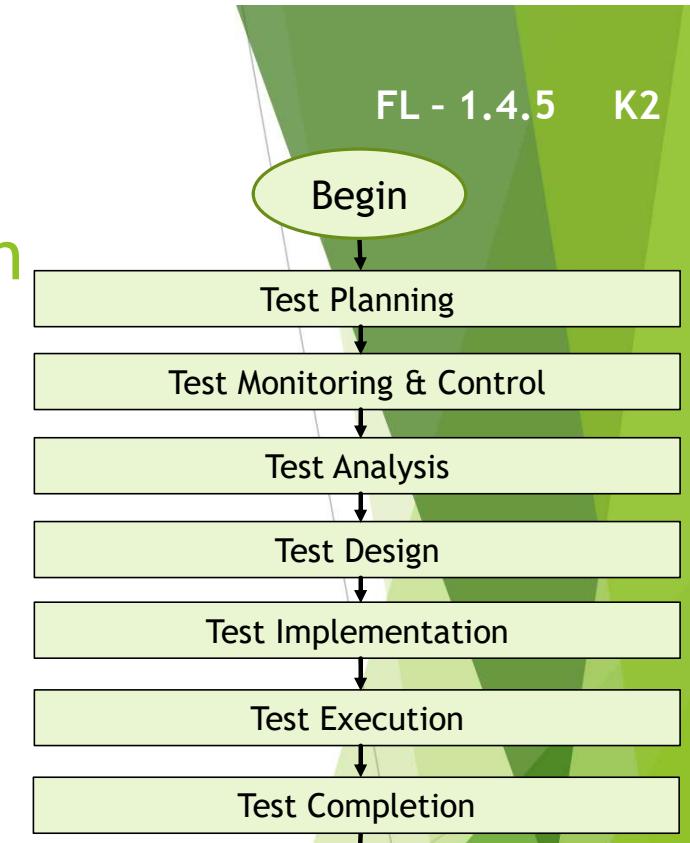
- ▶ Monitors test progress and results, checks the status of the exit criteria
- ▶ Adapts Test Plan and planning based on test results, progress and feedback



FL - 1.4.5 K2

Test Management role - Test Completion

- ▶ Checks the status of the exit criteria - 'Is the testing done?'
- ▶ Prepares and delivers the completion report
- ▶ Initiates the archiving or handing over the testware to the appropriate team(s)
- ▶ Initiates analysis of lessons learned



Test Management role - additionally

- ▶ Supports the selection and implementation of tools to support the test process
- ▶ Decides about the implementation of test environment(s)
- ▶ Promotes and advocates the testers, test team and the test profession within the organization
- ▶ Develops the skills and careers of testers (training, evaluations, coaching, etc)

Way in which the role is carried out depends on the Software Development Lifecycle:

- ▶ Agile - some of the tasks may be carried out by the team
- ▶ Some of the day to day management is taken care of by test managers outside of the development teams (test coaches)

Testing role - Activities

Overall responsibility for the:

- ▶ Engineering
- ▶ Technical

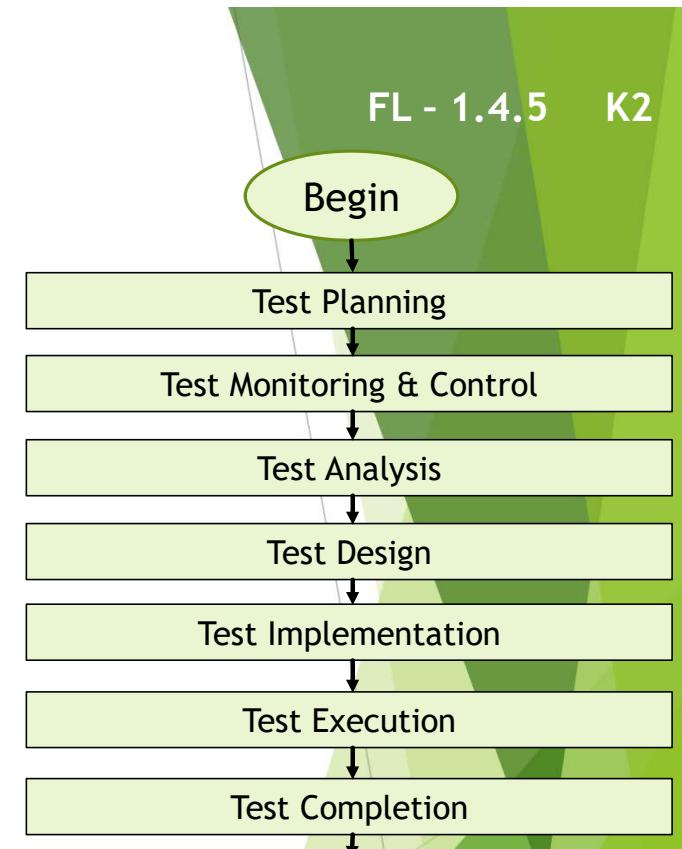
The main focus of the testing role lies in the below phases of the testing process:

- ▶ Test Analysis
- ▶ Test Design
- ▶ Test Implementation
- ▶ Test Execution



Testing role - Activities

- ▶ Reviews and contributes to test plans
- ▶ Analyzes, reviews and assesses requirements, user stories, acceptance criteria, specifications and models for testability
- ▶ Identifies and documents test conditions and captures traceability between test cases, test conditions and the test basis
- ▶ Designs, sets up and verifies test environment (with system admin and network management)
- ▶ Designs and implements test cases, test procedures and test data
- ▶ Creates the detailed test execution schedule
- ▶ Executes tests, evaluates the results and documents deviations from expected results
- ▶ Uses appropriate tools to facilitate the test process
- ▶ Automates tests as needed



FL - 1.4.5 K2

Testing role - Activities (2)

- ▶ Evaluates non-functional characteristics such as performance, efficiency, reliability, usability, security, compatibility, portability
- ▶ Reviews tests developed by others
- ▶ Specialist roles can be related to test analysis, test design, specific test types or test automation

Different roles may take over the role of tester based on the risks related to the project and product and on the Software Development Lifecycle used:

- ▶ Component - Integration level - Developers
- ▶ Acceptance level - Business experts and users
- ▶ System - Integration level - Independent test team
- ▶ Operational Acceptance level - Operators

Role division

Test management role

- ▶ Team Leader
- ▶ Test Manager
- ▶ Development Manager
- ▶ One person who has the Test management AND the Testing roles

Testing Role

- ▶ Tester
- ▶ DevOps Engineer
- ▶ Developer
- ▶ Customer Support



Fundamentals of Testing

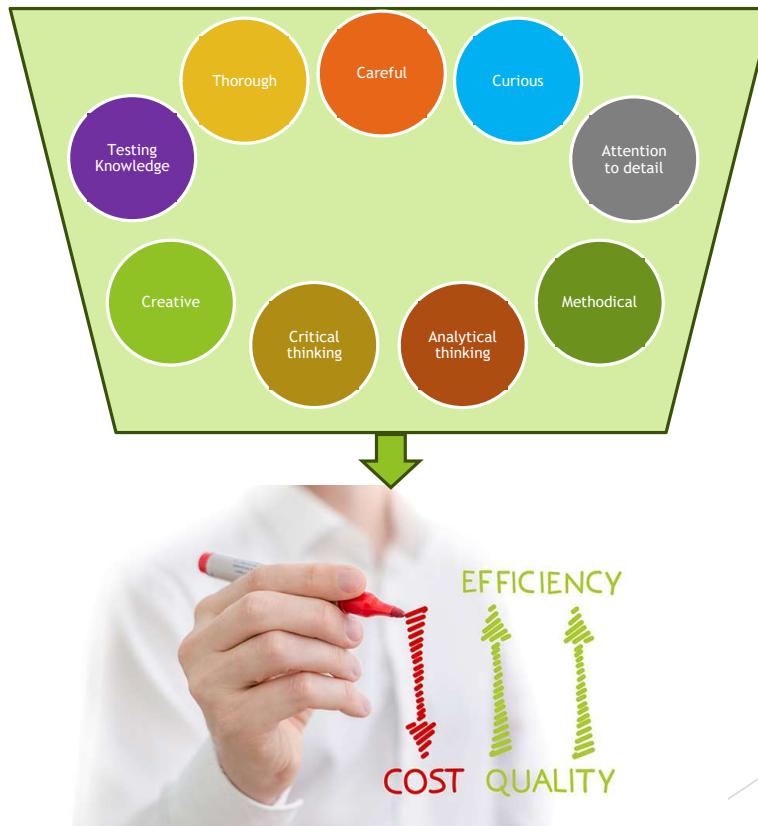
Essential Skills and Good
Practices in Testing



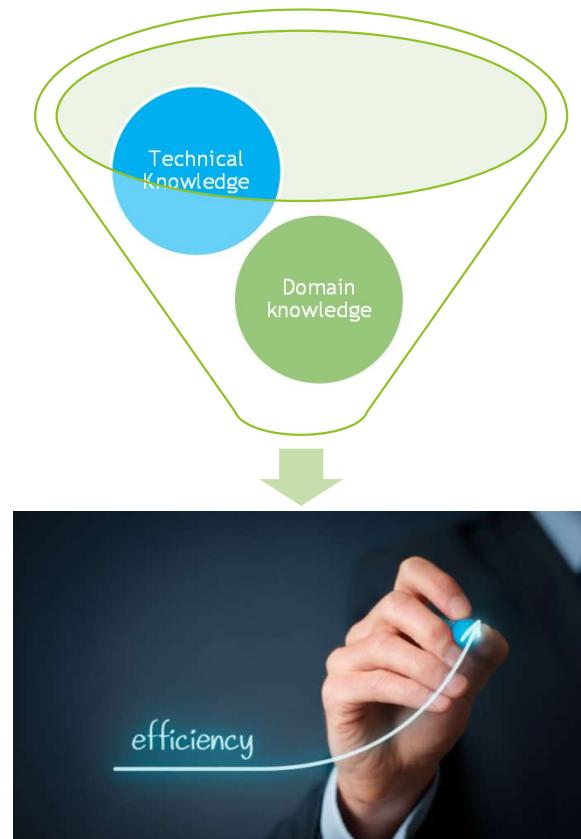
Fundamentals of Testing

Generic Skills Required for Testing

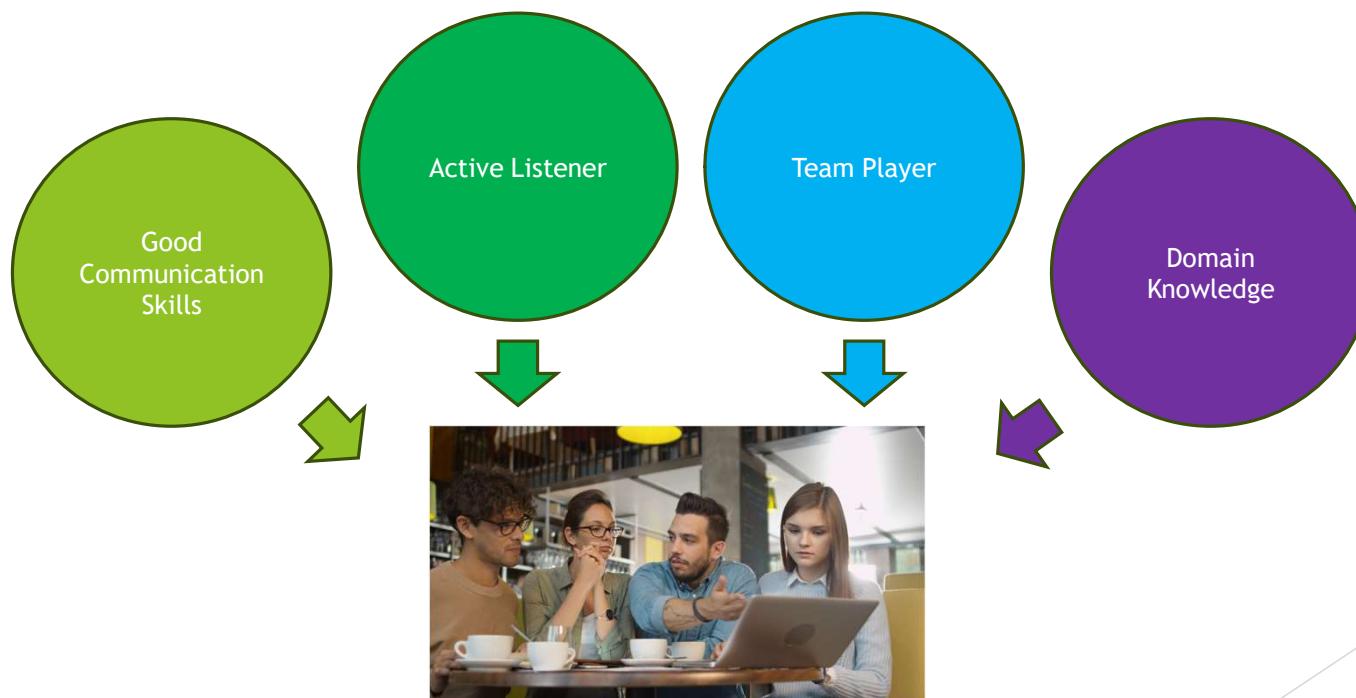
Increasing Effectiveness of Testing



Increasing Efficiency of Testing



Communication with Stakeholders



(Don't) Shoot the Messenger

Keep in mind:

- ▶ Confirmation bias
- ▶ Collaboration and teamwork instead of battles
- ▶ Stay objective and factual, not focused on the person
- ▶ Try to understand the way the other person feels and reacts
- ▶ Be sure the other person understood what you have said
- ▶ Clear test objectives influence plans and individual behavior



Fundamentals of Testing

The Whole Team Approach

What is a team?

'A small number of people with complementary skills who are committed to a common purpose, performance goals and approach for which they hold themselves mutually accountable'

Whole-Team Approach

- ▶ Any team member with the necessary knowledge and skills can perform any task
- ▶ EVERYONE is responsible for quality
- ▶ Ideally the team shares the same workspace (for communication and interaction)
- ▶ The team includes customer and other business representatives
- ▶ Teams should be relatively small (3-9 people), larger teams to be split up
- ▶ Approach is supported through daily stand-up meetings with the entire team
- ▶ One of the main ideas behind Agile development



Whole-Team Approach - Purpose



- ▶ Maximize the business value
- ▶ The whole team is responsible for the quality
- ▶ Collaboration between testers and other team members to achieve the desired quality levels creates **transparency** and knowledge sharing
 - ▶ Testers work with developers and business to achieve desired quality
 - ▶ Testers assist the business in creating proper acceptance tests
 - ▶ Testers work with developers to decide on the testing and automation strategy
 - ▶ Testers can transfer testing knowledge to other members of the team and influence the development
 - ▶ Product features are discussed with the whole team - ‘Power of Three’

Whole-Team Approach - Benefits

- ▶ Enhancing communication and collaboration within the team
- ▶ Improving team dynamics through effective communication, teamwork and collaboration
- ▶ Creating synergy in the team by enabling the various skill sets within the team to be leveraged to the benefit of the project
- ▶ Helps team members learn and share knowledge with each other
- ▶ Making every team collectively responsible for the result
- ▶ Making quality everyone's responsibility





Fundamentals of Testing

Independence of testing

Independence of Testing

Low

- ▶ Authors test their own work product
- ▶ Work products are tested by the author's peers in the same team
- ▶ Independent test team or group within the organization but outside the team
- ▶ Independent testers external to the organization (on-site or off-site)

High

Points of Attention - Independence of Testing

- ▶ Usually best to test with multiple levels of independence. For example:
 - Developers perform component and component integration testing
 - Test team performs system and system integration testing
 - Business representative performs acceptance testing
- ▶ Testing can be done by testers or by people in other roles
- ▶ Independence often makes testers more efficient
- ▶ Independence should not be confused with familiarity
- ▶ Independence of testing can be implemented in many ways

Independence of Testing - benefits - risks

Benefits	Risks
Likely to recognize different kind of failures compared to developers	Isolation from the development team
Independent tester can verify, challenge or disprove assumptions made by stakeholders in specification and implementation	Developers may lose sense of responsibility for the quality
	Independent testers may be seen as bottlenecks or blamed for delays in releases
	Independent testers may lack some important information about the test object



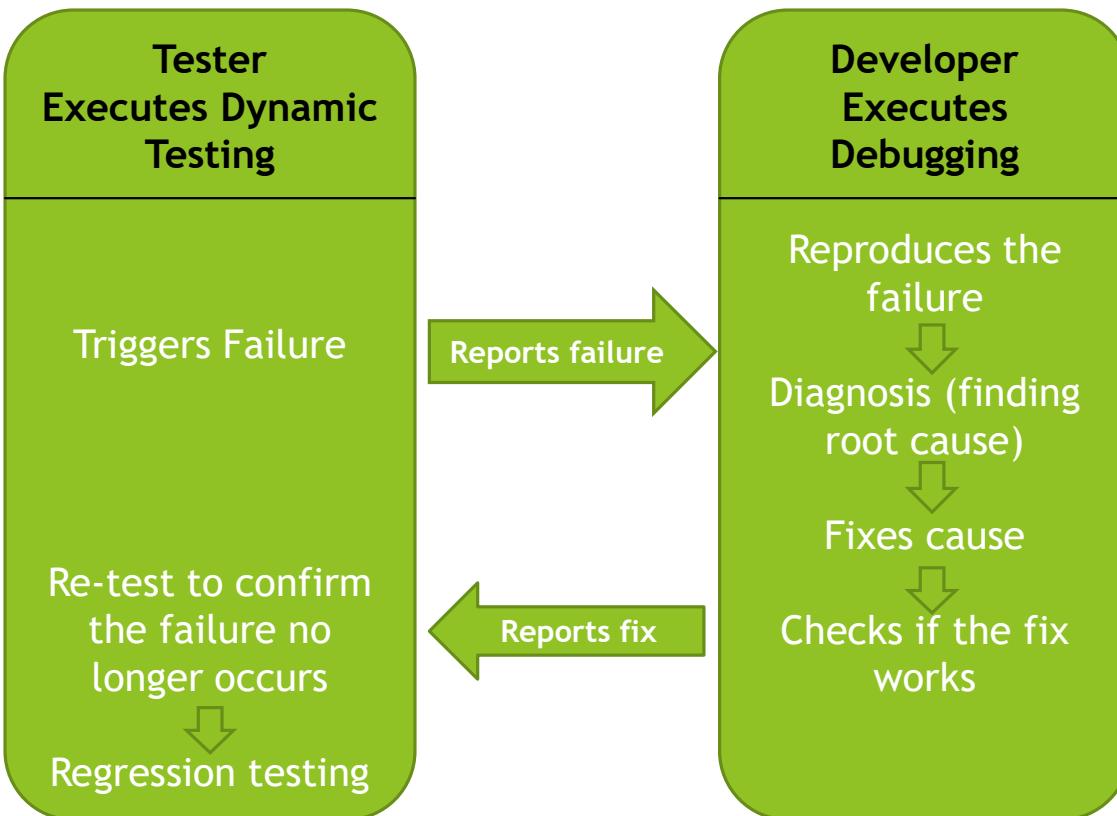
Fundamentals of Testing

Summary - Keywords
Explained

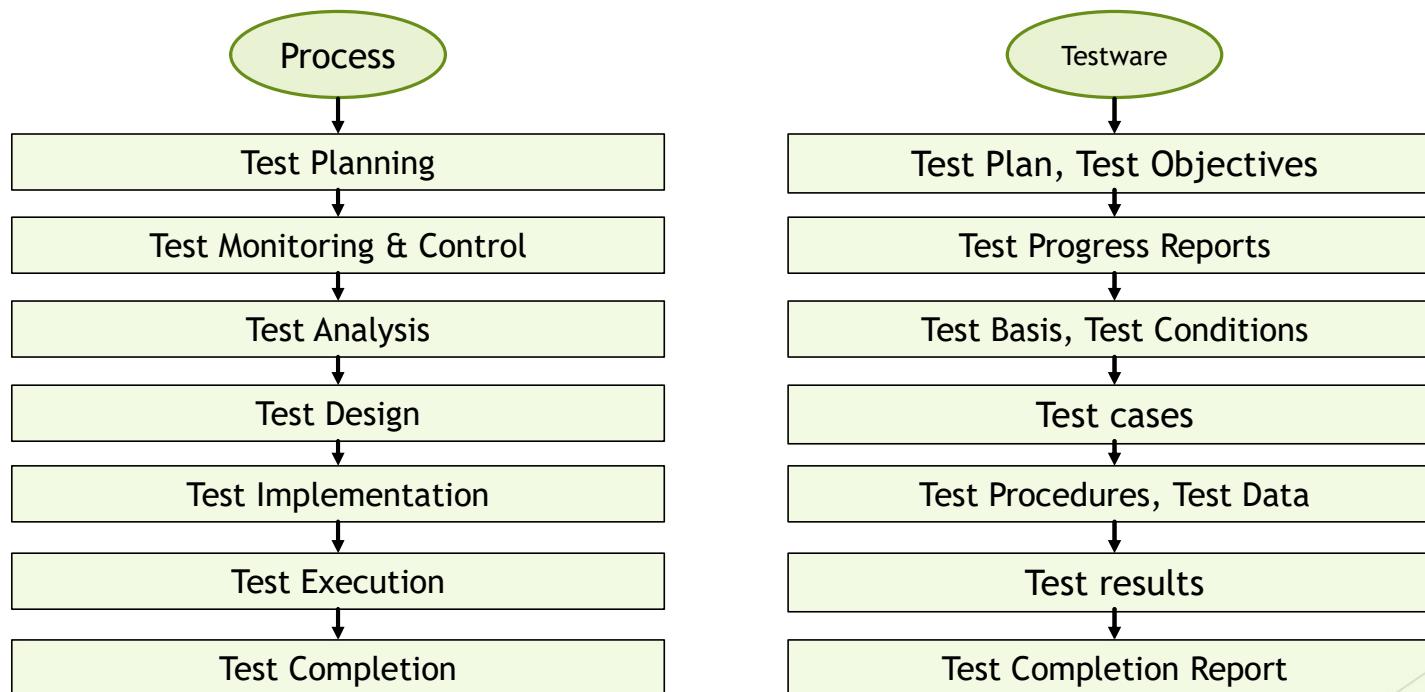
Keywords - explained

- ▶ Error - A human action (mistake) that produces an incorrect result.
- ▶ Defect - An imperfection or deficiency in a work product where it does not meet its requirements or specifications (aka bug, fault)
- ▶ Failure - An event in which a component or system does not perform a required function within specified limits.
- ▶ Debugging - The process of finding, analyzing and removing the causes of failures in a component or system.
- ▶ Root Cause - A source of a defect such that if it is removed, the occurrence of the defect type is decreased or removed.

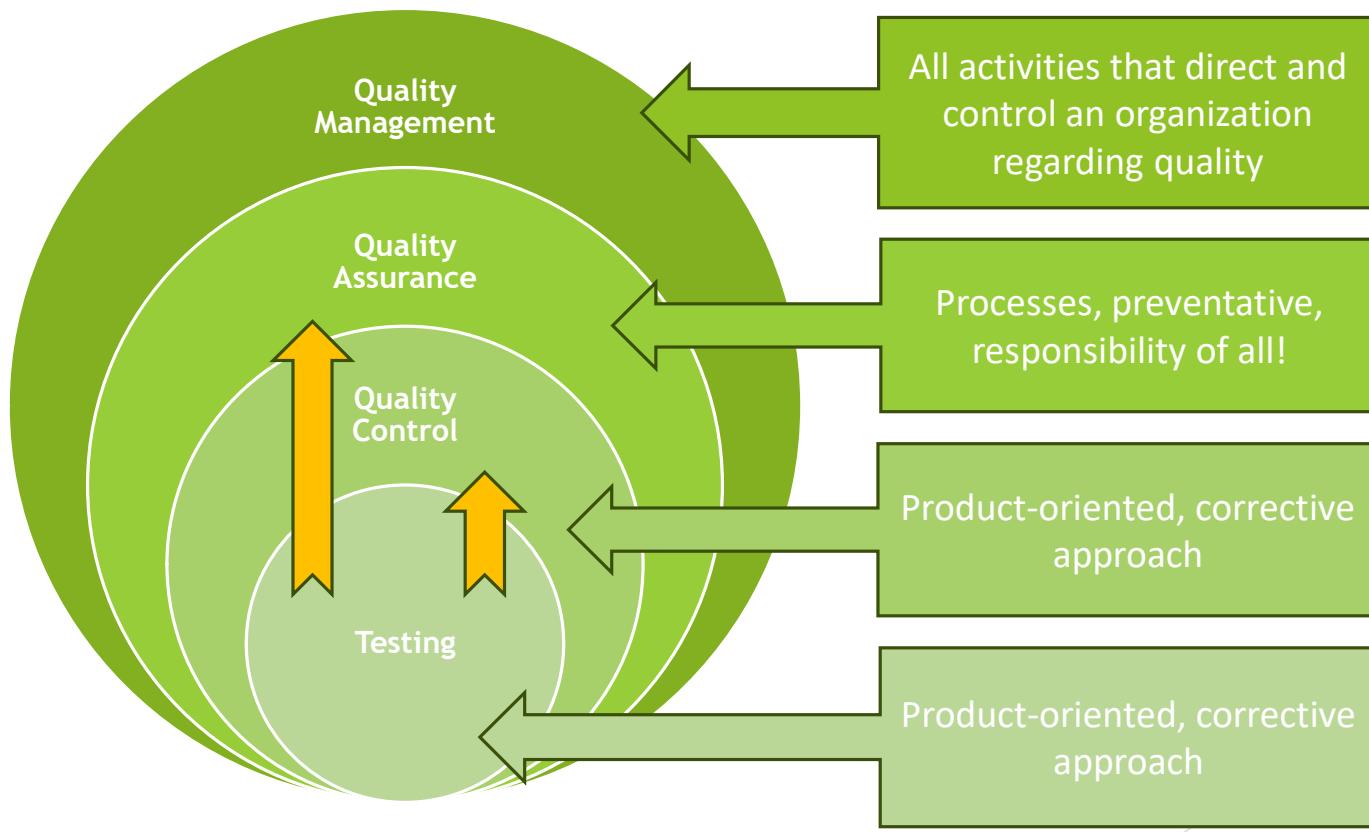
Dynamic Testing and Debugging - visualized



Keywords - explained (2)



Quality Assurance and Testing



Verification vs Validation

Software Verification

- ▶ Confirmation by examination and through provision of objective evidence that specified **requirements** have been fulfilled.
- ▶ Is the product built according to the specification?

Software Validation

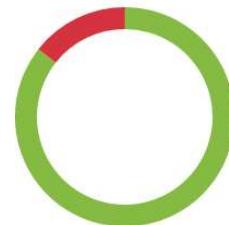
- ▶ Confirmation by examination that a work product matches the **user needs and requirements**.
- ▶ Is the product fit for the purpose (does it solve the problem?)

Coverage

The degree to which specified coverage items are exercised by a test suite, expressed as a percentage.

Can be measured on anything:

- ▶ Statements
- ▶ Decisions
- ▶ Paths
- ▶ Branches
- ▶ Features
- ▶ Automation
- ▶ ...



%

—
Coverage





Testing Throughout the Software Development Lifecycle

Introduction

Agenda

2.1 Testing in the Context of a Software Development Lifecycle

- ▶ FL-2.1.1 (K2) Explain the impact of the chosen software development lifecycle on testing
- ▶ FL-2.1.2 (K1) Recall good testing practices that apply to all software development lifecycles
- ▶ FL-2.1.3 (K1) Recall the examples of test-first approaches to development
- ▶ FL-2.1.4 (K2) Summarize how DevOps might have an impact on testing
- ▶ FL-2.1.5 (K2) Explain the shift-left approach
- ▶ FL-2.1.6 (K2) Explain how retrospectives can be used as a mechanism for process improvement

2.2 Test Levels and Test Types

- ▶ FL-2.2.1 (K2) Distinguish the different test levels
- ▶ FL-2.2.2 (K2) Distinguish the different test types
- ▶ FL-2.2.3 (K2) Distinguish confirmation testing from regression testing

2.3 Maintenance Testing

- ▶ FL-2.3.1 (K2) Summarize maintenance testing and its triggers



Testing in the context of a Software Development Lifecycle

Introduction

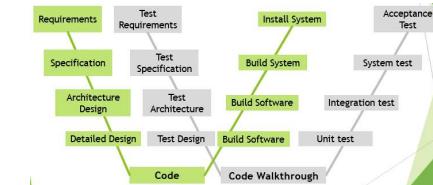
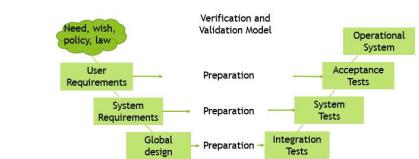
Software Development Lifecycle Model

- ▶ Abstract, high-level representation of the Software Development Process
- ▶ Defines relation between Development Phases and types of activities:
 - ▶ Logically
 - ▶ Chronologically
- ▶ Many different SDLC models
- ▶ Many different types of activities described by
 - ▶ More detailed software development methods
 - ▶ Agile practices

Software Development Lifecycle Models

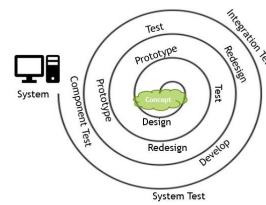
► Sequential

- Waterfall model
- V-model
- W-Model



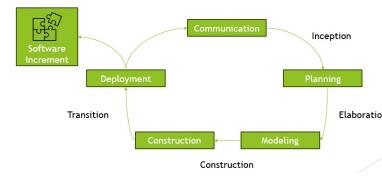
► Iterative

- Spiral
- Prototyping



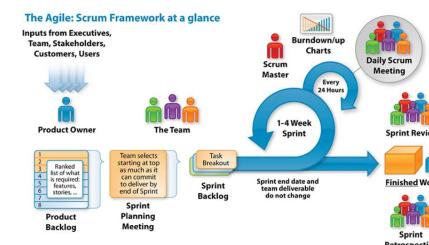
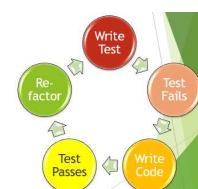
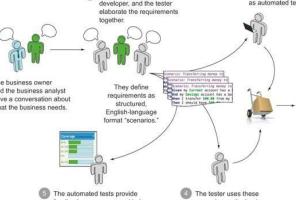
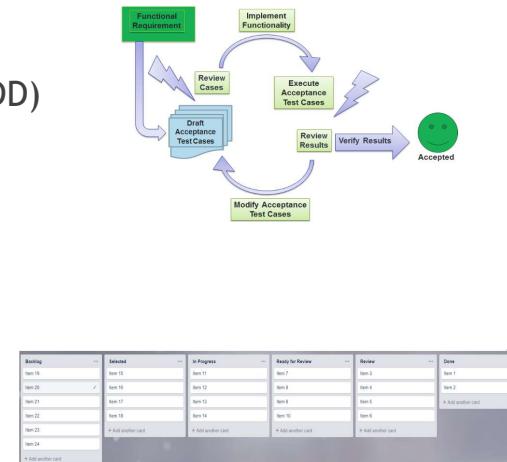
► Incremental

- Unified Process



Specific Methods and Agile Practices

- ▶ Acceptance-test driven Development (ATDD)
- ▶ Behavior-driven Development (BDD)
- ▶ Domain-driven Design (DDD)
- ▶ Extreme programming (XP)
- ▶ Feature-driven Development (FDD)
- ▶ KANBAN
- ▶ Lean IT
- ▶ SCRUM
- ▶ Test-driven Development (TDD)





Impact of the Software Development Lifecycle on testing

Impact of the Software Development Lifecycle

- ▶ Testing must be adapted to the SDLC
- ▶ SDLC has an impact on:
 - ▶ Scope and timing of Test Activities
 - ▶ Level of Detail of Test Documentation
 - ▶ Choice of Techniques and Test Approach
 - ▶ Extent of Test Automation
 - ▶ Role and responsibilities of a tester



Testing Throughout the Software Development Lifecycle

Verification vs Validation

Verification vs Validation

Software Verification

- ▶ Evaluating a work product, component or system to determine whether it meets the **requirements set**
- ▶ Is the product built according to the specification?

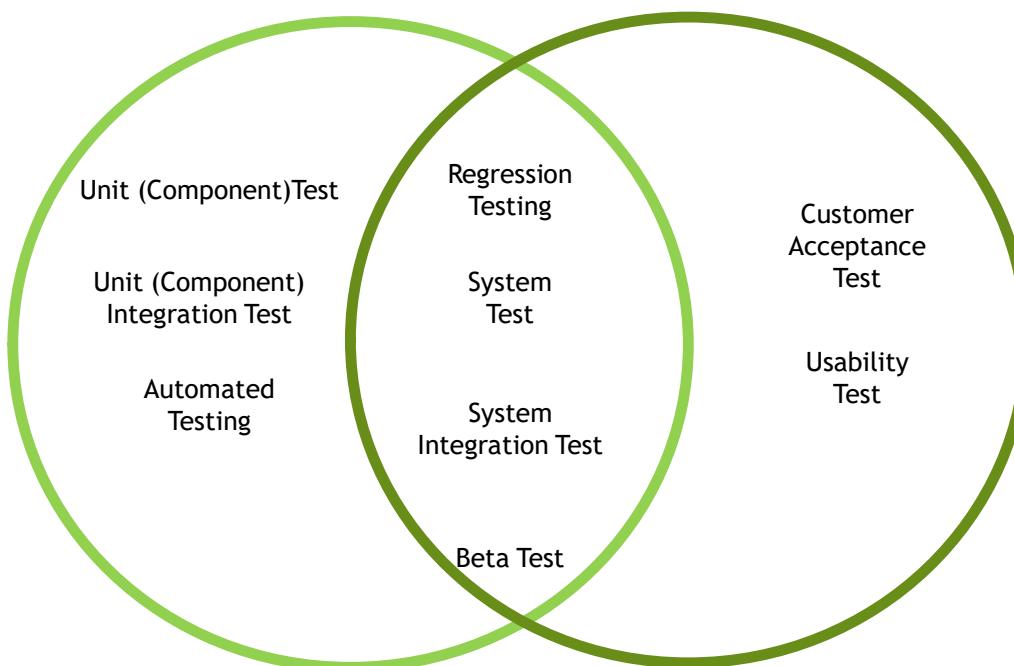
Software Validation

- ▶ Evaluating a work product, component or system to determine whether it meets the **user needs and requirements**
- ▶ Is the product fit for the purpose (does it solve the problem?)

Verification vs Validation - Visualized

Verification

Building the product right



Validation

Building the right product



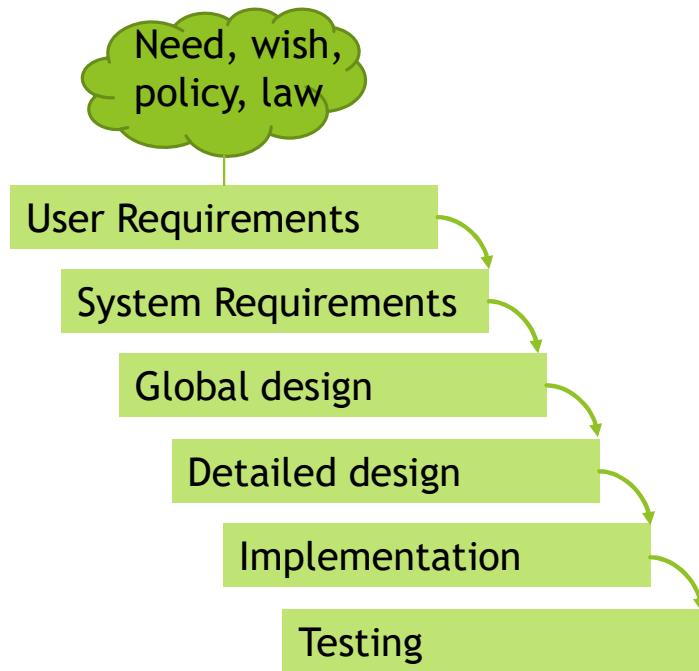
Impact of the Software Development Lifecycle on Testing

Sequential Development

Testing in Sequential Models

- ▶ Entire system is built at once
- ▶ Strict sequence of activities
 - ▶ Define
 - ▶ Implement
 - ▶ Test
- ▶ Each phase must be finished before the next phase starts
- ▶ Testers start participating in requirement reviews, test analysis and test design
- ▶ Dynamic testing cannot start early in the SDLC

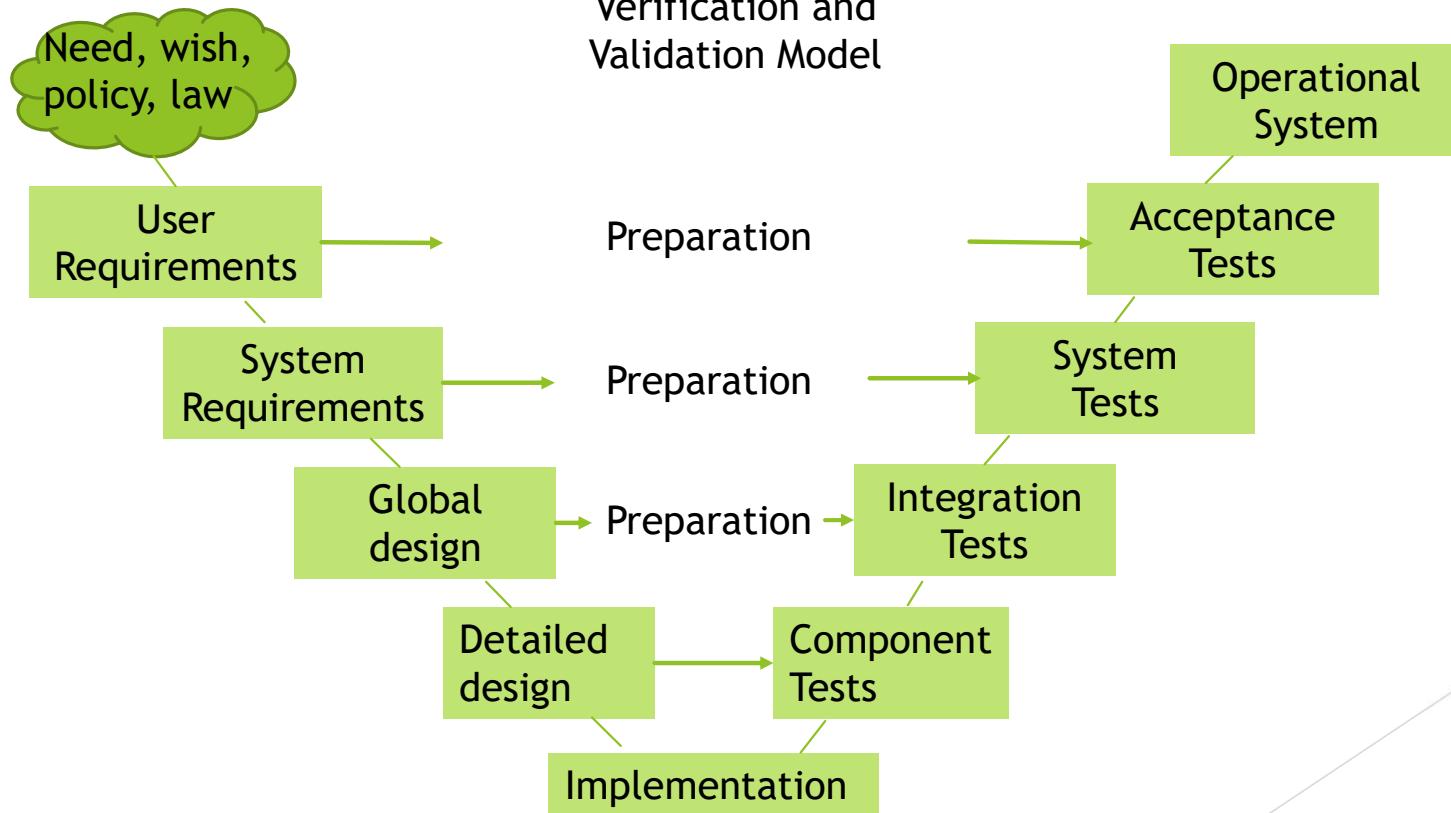
Testing in the Waterfall model



Issues

- ▶ Testing at the end of the coding process
- ▶ Defects detected close to implementation
- ▶ Difficult to pass feedback up the waterfall

Testing in the V-model



Advantages and disadvantages of the V-Model

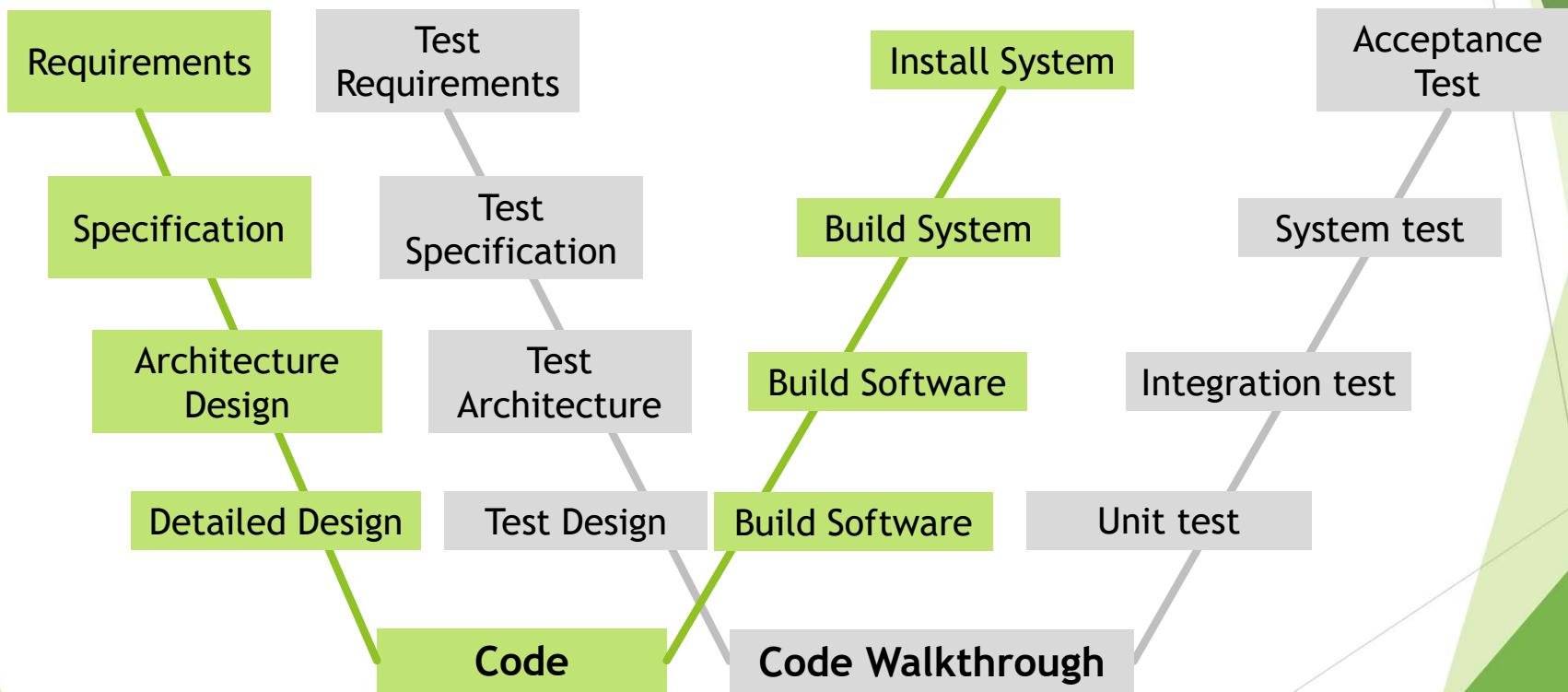
Advantages

- ▶ Simple and easy
- ▶ Systematic
- ▶ Easy to track
- ▶ Testing starts from requirement phase
- ▶ All the functional areas can be covered
- ▶ Instructions and recommendations included
- ▶ Detailed explanations of problems
- ▶ Defects can be found at an early stage
- ▶ Works well for small projects

Disadvantages

- ▶ Not flexible
- ▶ Regular updates required if the project changes
- ▶ Can't be used in complex projects
- ▶ No scope for risk management and mitigation
- ▶ It ends when the project is over

Testing in the W-Model



FL - 2.1.1 K2

Advantages and disadvantages of the W-Model

Advantages

- ▶ Testing can run in parallel with development process
- ▶ No division between development and testing tasks - all are integrated
- ▶ Often developer are responsible for removing defects

Disadvantages

- ▶ Complex to implement
- ▶ Resource allocation might not be sufficient in most of the cases
- ▶ Testing has equal weight as many activities in the development process



Impact of the Software Development Lifecycle on Testing

Incremental and Iterative Models

What is incremental development?

- ▶ The system is not built in one go, but in pieces
- ▶ These pieces are called ‘product increments’
- ▶ Generally, after each iteration there should be an increment or working prototype delivered
- ▶ The features in the software grow ‘incrementally’
- ▶ Each part of the process is done in pieces:
 - ▶ Establishing requirements
 - ▶ Designing
 - ▶ Building
 - ▶ Testing

Incremental development visualized

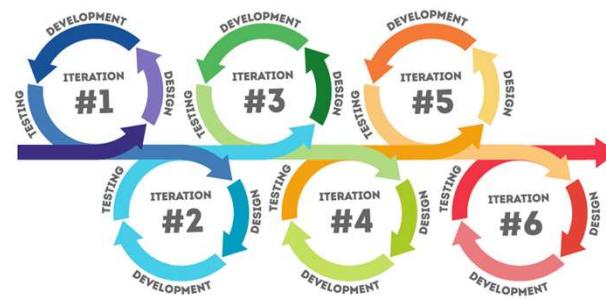


FL - 2.1.1 K2

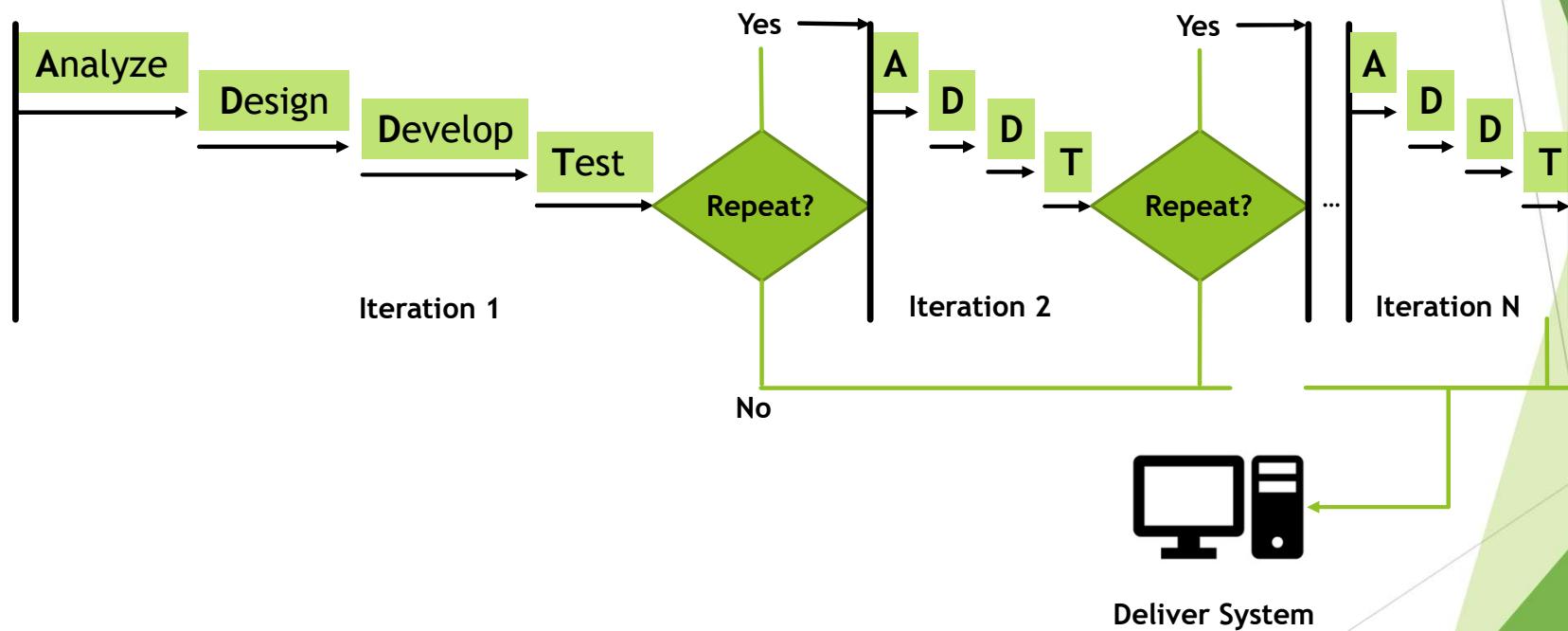


What is Iterative development

- ▶ Builds are provided in iterations
- ▶ Each iteration involves cross functional teams working simultaneously on the different phases
- ▶ End of each iteration is a shippable product
- ▶ In each iteration, static and dynamic testing may be done on ALL test levels
- ▶ Fast feedback and extensive regression testing is needed for frequent delivery of increments



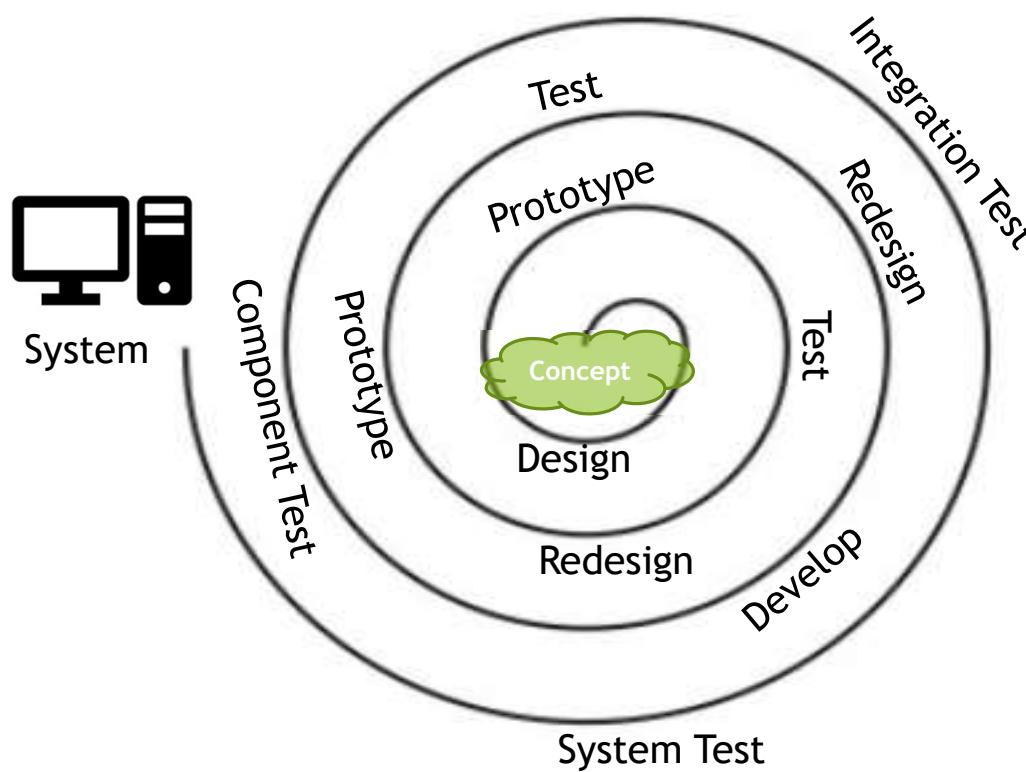
Iterative development visualized



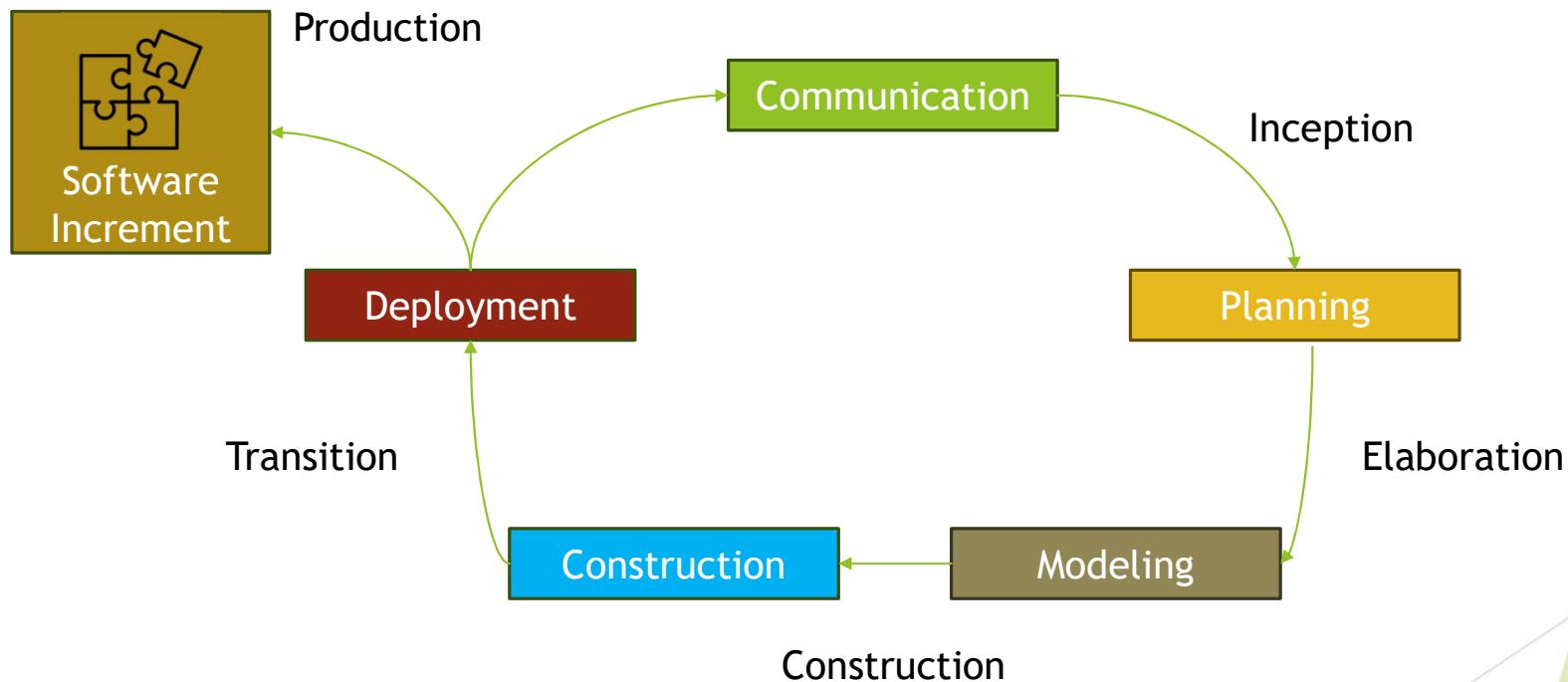
Testing in Spiral Models

- ▶ Early prototypes are used to design the system
- ▶ Development work passes through a sequence of prototypes that are:
 - ▶ Tested
 - ▶ Redesigned
 - ▶ Reprototyped
 - ▶ Retested
 - ▶ ...
 - ▶ Until all design decisions have been proven by testing
- ▶ Most useful on projects with a large number of unknowns

Testing in Spiral Models (2)



Unified Process





Impact of the Software Development Lifecycle on Testing

Agile Software Development

The Fundamentals of Agile Software Development

The Agile Manifesto

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

- ▶ The Agile manifesto values the concepts on the right, but believes the ones on the left have greater value

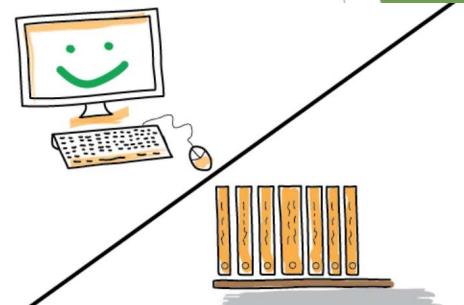
Individuals and Interactions

- ▶ Agile is people-centred
- ▶ Teams build software
- ▶ Continuous communication and interaction enable teams to work most effectively
- ▶ Processes and tools are important, but not as important as people and communication



Working Software

- ▶ Customers do not care about detailed documentation
- ▶ Working software is more useful and valuable
- ▶ Enables rapid feedback to the development team
- ▶ Availability of working software can result in time-to-market advantage
- ▶ Agile development is especially useful in:
 - ▶ Fast-changing business environments
 - ▶ Environments where problems and solutions are unclear
 - ▶ Environments where the business wants to innovate in new problem domains



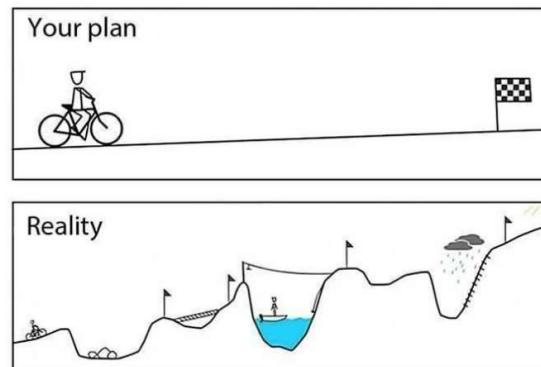
Customer Collaboration

- ▶ Customer finds it difficult to specify requirements
- ▶ Working strictly on the original requirements might end up with useless software
- ▶ Collaboration results in better understanding of what the customer wants
- ▶ Contracts are important, working closely together with customers is more likely to bring success



Responding to change

- ▶ Change is inevitable and should be responded to
- ▶ Flexibility is more important than sticking to the plan
- ▶ Smart and flexible planning is needed
- ▶ Many factors can have a major influence on the project:
 - ▶ Environment
 - ▶ Business domain
 - ▶ Legislation
 - ▶ Technological advances



Whole-Team Approach

- ▶ Involving everyone with the needed skills and knowledge to ensure success.
- ▶ The team includes customer and other business representatives
- ▶ Teams should be relatively small (3-9 people), larger teams to be split up
- ▶ Ideally the team shares the same workspace (for communication and interaction)
- ▶ Approach is supported through daily stand-up meetings with the entire team
- ▶ Promotes more effective and efficient team dynamics
- ▶ One of the main ideas behind Agile development



Early and Frequent Feedback

- ▶ Short iterations enable early and continuous feedback on product quality
- ▶ If you have to fail, **fail fast!**
- ▶ Agile teams can better implement new changes into the development process
- ▶ The team can better focus on features with the highest business value and risk
- ▶ Team management is better as the capability is transparent to all:
 - ▶ How much work can we do?
 - ▶ What could help us go faster?
 - ▶ Why are we not doing that then?

Agile Testing Specifics

- ▶ Changes will occur, so:
 - ▶ Use lightweight documentation
 - ▶ Implement extensive test automation
 - ▶ Use experience-based testing for manual test activities
- 
- Make Regression testing easier and faster
- Not to waste time on extensive test analysis & design

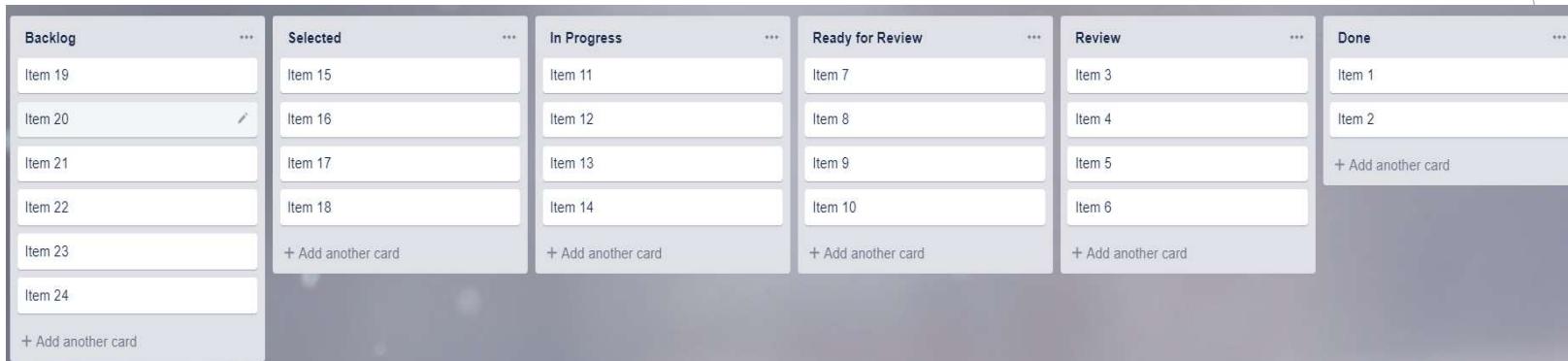
Agile Approaches - SCRUM

The Agile: Scrum Framework at a glance

Inputs from Executives,
Team, Stakeholders,
Customers, Users

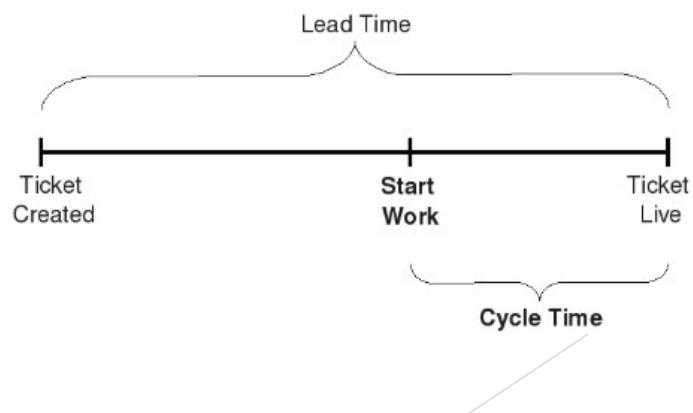


Agile Approaches - Kanban



Kanban utilizes three instruments

- Kanban Board
- Work-in-Progress Limit
- Lead Time



Agile Approaches - Kanban vs SCRUM

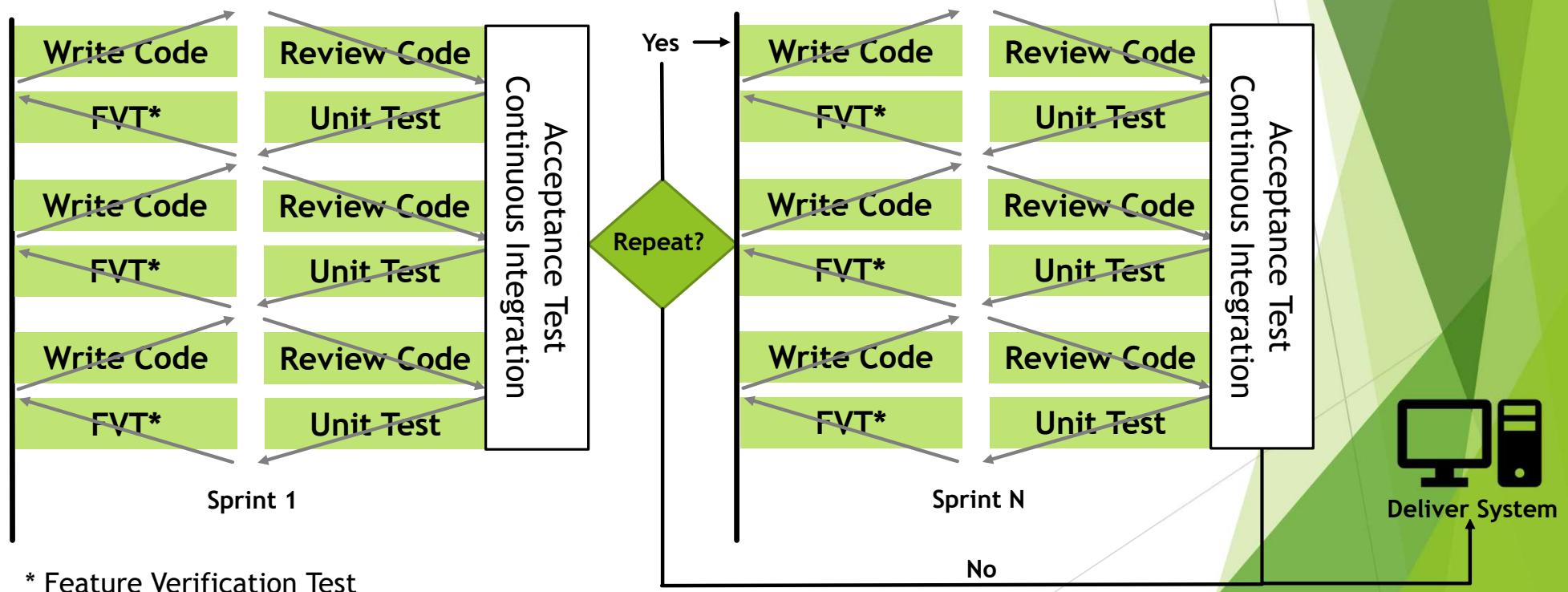
Similarities

- ▶ Visualization of active tasks
- ▶ Transparency on content and progress
- ▶ Inactive tasks are in the backlog
- ▶ Tasks get placed on the backlog when there is capacity available

Differences

- ▶ Iterations (sprints) are optional in Kanban
- ▶ Deliverables are released item by item instead of in a release
- ▶ Timeboxing is optional, unlike in SCRUM

Testing in Agile Models



Agile & Testing

Benefits

- ▶ Focus on working software and good quality code
- ▶ Testing as starting point of development
- ▶ Business stakeholders accessible
- ▶ Whole team responsible for the quality
- ▶ Simple design is easier to test

Challenges

- ▶ Less well-documented requirements
- ▶ Testers feel less needed as dev does more component testing
- ▶ More of a coaching role, can be difficult
- ▶ Constant time pressure and less time to think on testing new features
- ▶ Regression becomes extremely important and automation more beneficial

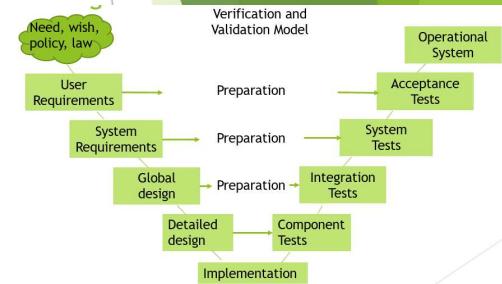


Good Testing Practices

Testing in a life cycle model

Whichever development model is used, the below is always the case:

- ▶ For each development activity there is a corresponding testing activity;
- ▶ Each test level has test objectives specific to that level;
- ▶ Test analysis and 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 cycle.





Testing as a Driver for Software Development

Test-First Development Approaches

Approaches to produce quality products - introduce testing as early as possible:

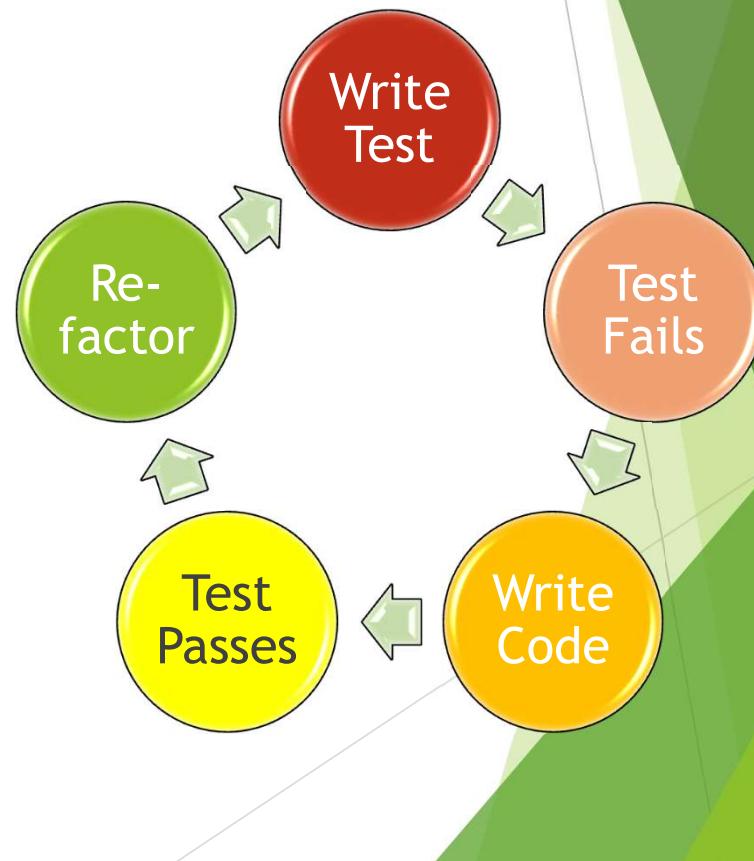
- ▶ Writing tests in advance, before the code
- ▶ Focusing on early defect prevention, detection and removal
- ▶ Following a shift-left approach
- ▶ Ensuring the right test types are run at the right time as part of the right test level
- ▶ Use automated tests to ensure code quality in future adaptations or code refactoring

Agile testers play a key role in guiding the use of these testing practises

Test-Driven Development (TDD)

- ▶ Became popular through XP
- ▶ Develop code guided by automated test cases
- ▶ Mainly on unit level and code-focused
- ▶ CAN also be on integration/system level
- ▶ Is used in Agile and sometimes in sequential
- ▶ Helps developers focus on clearly expected results
- ▶ Tests are automated and used in CI

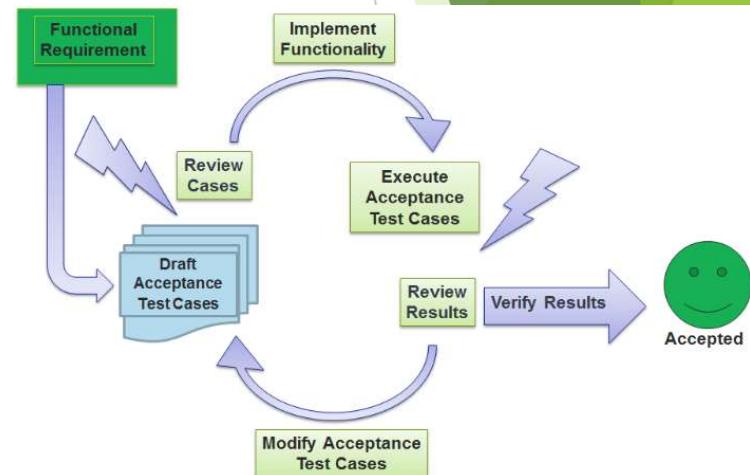
Is the code correct?



Acceptance Test-Driven Development

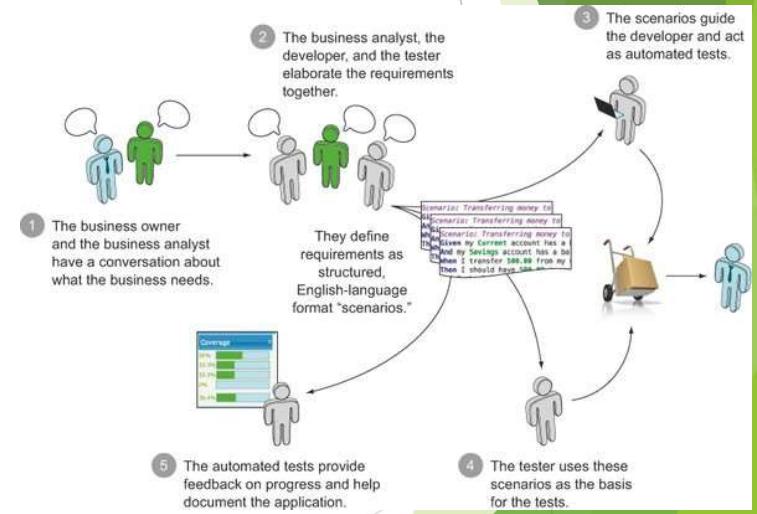
- ▶ Acceptance criteria and tests are defined during the creation of the user stories
- ▶ Encourages collaboration amongst the business, developer and tester
- ▶ Every stakeholder should understand how the software component has to behave and what is needed to ensure this behaviour
- ▶ ATDD creates reusable tests for regression testing
- ▶ Tools support creation and execution of tests (often in the CI process)
- ▶ Tools can connect to data and service layers of the application (enabling execution on system or acceptance levels)
- ▶ ATDD allows quick resolution of defects and validation of feature behaviour
- ▶ Helps determine if the acceptance criteria are met for the feature

Is the code doing what it is supposed to do?



Behavior-Driven Development

- ▶ BDD helps developers collaborate with other stakeholders to define accurate tests **focused on business needs**, so on the **desired behavior** of an application
- ▶ Allows a developer to focus on testing the code based on the desired behaviour
- ▶ As tests are based on the desired behaviour, the tests are easier to understand
- ▶ Specific frameworks to define acceptance criteria based on the ‘given / when / then’ format:
 - ▶ Given some initial context,
 - ▶ When an event occurs,
 - ▶ Then ensure some outcomes.
- ▶ From these requirements, test cases are automatically translated into executable tests



TDD vs BDD vs ATDD

Parameters	TDD	BDD	ATDD
Definition	TDD is a development technique that focuses more on the implementation of a feature	BDD is a development technique that focuses on the system's behavior	ATDD is a technique similar to BDD focusing more on capturing the requirements
Participants	Developer	Developers, Customer, QAs	Developers, Customers, QAs
Language used	Written in a language similar to the one used for feature development (Eg. Java, Python, etc)	Simple English, (Gherkin)	Simple English, Gherkin
Main Focus	Unit Tests	Understanding Requirements	Writing Acceptance Tests
Tools used	JDave, Cucumber, JBehave, Spec Flow, BeanSpec, Gherkin, Concordian, FitNesse	Gherkin, Dave, Cucumber, JBehave, Spec Flow, BeanSpec, Concordian	TestNG, FitNesse, EasyB, Spectacular, Concordian



DevOps and Testing

What is DevOps?

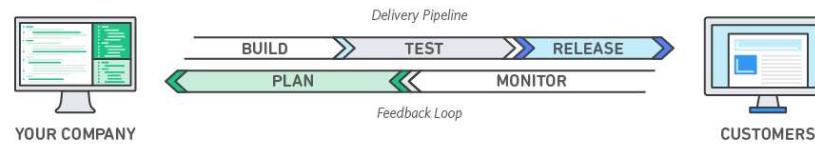
- ▶ DevOps is an Agile approach
- ▶ Removing the siloes between development and operations
- ▶ Working together towards achieving a set of common goals
- ▶ There are no separate agendas for development and operations
- ▶ Implementation requires an organizational and cultural shift
- ▶ All functions are treated with equal value
- ▶ The team has full autonomy



Why DevOps?

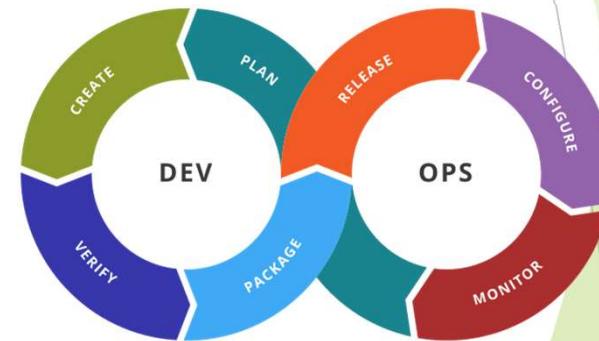
If implemented correctly in the organization:

- ▶ Team autonomy
- ▶ Everyone being on the same page
- ▶ Integrated toolchains, enabling (amongst others)
- ▶ Continuous integration (CI) and Continuous Delivery (CD):
 - ▶ Build
 - ▶ Test and
 - ▶ Release **high quality** software fast through DevOps Delivery Pipelines
- ▶ Fast feedback loops in all phases
 - ▶ Code built with unit tests
 - ▶ Code coverage defined and measured with each code check-in
 - ▶ Fast and better view on non-functional quality characteristics
 - ▶ Fast releases means fast feedback from the customer base



Benefits of DevOps on Testing

- ▶ Fast feedback on the code quality (new functionalities and regression)
- ▶ Shift-left approach in testing is promoted
 - ▶ Checking in high quality code
 - ▶ Unit tests
 - ▶ Static analysis
- ▶ CI/CD processes create stable environments
- ▶ Better view on non-functional quality characteristics
- ▶ Need for repetitive manual testing is reduced through automated testing and delivery pipelines
- ▶ Minimized risk in regression through automation



Risks of DevOps on Testing

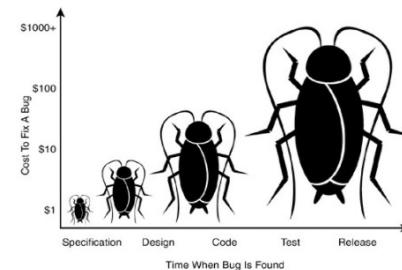
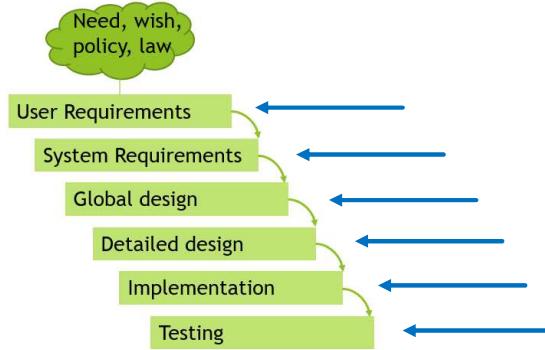
- ▶ Define and establish the CI/CD Pipelines
- ▶ CI/CD Tools implementation and maintenance
- ▶ Test Automation requires additional resources



Shift-Left Approach

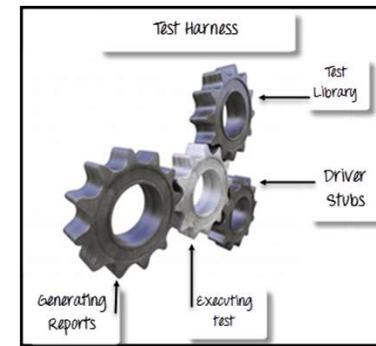
What is the Shift-Left Approach?

- ▶ Early Testing
- ▶ Don't wait until the code has been written
- ▶ Let's visualize this in a sequential development model



Shift-Left Best Practices

- ▶ Static Testing (reviewing) of the specification
- ▶ Write test cases before the code is written
- ▶ Run the code in a test harness during implementation
- ▶ Implement CI and CD
- ▶ Complete Static Analysis before Dynamic Testing
- ▶ Non-Functional tests should **also** be done earlier (shift-left)



Points of Attention in Shift-Left

- ▶ All stakeholders need to be in on it
- ▶ Investments are needed in the earlier development phases:
 - ▶ Training
 - ▶ Effort
 - ▶ Cost

BUT

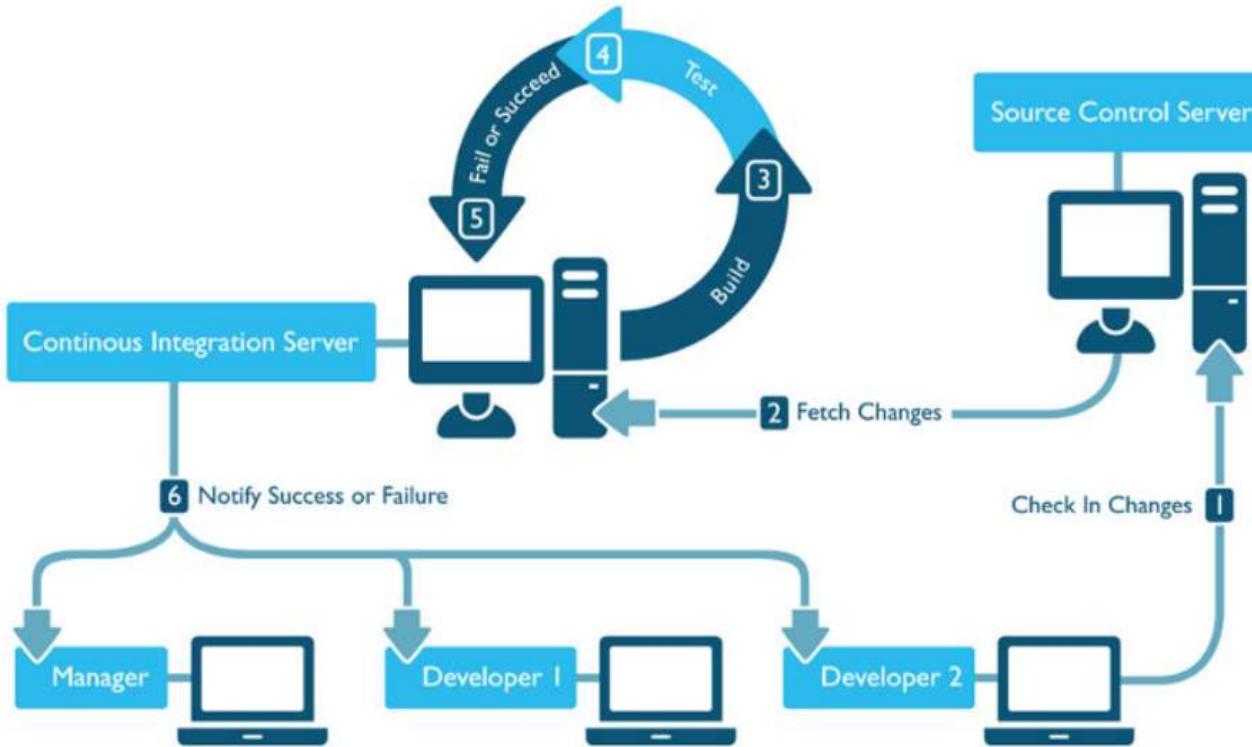


It Will Pay Off



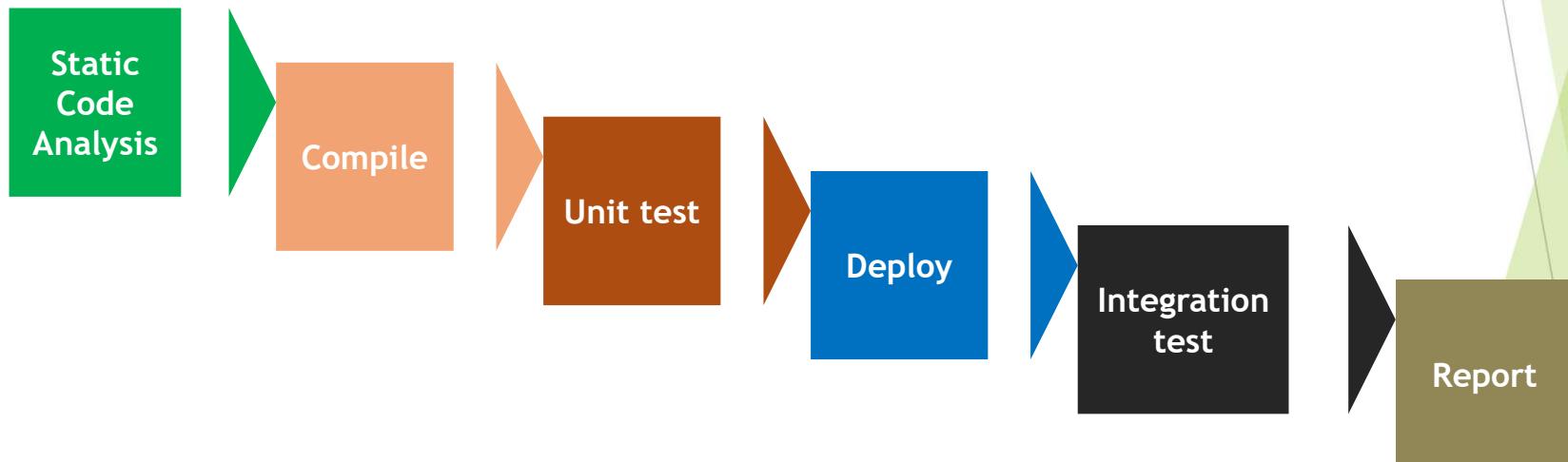
Continuous Integration

Continuous Integration - Visualized



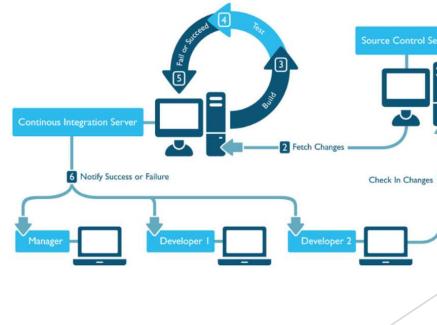
Continuous Integration Process

After the coding, debugging, and check-in of the code into the source code repository, the automated CI process kicks off:



CI - Continuous automated testing

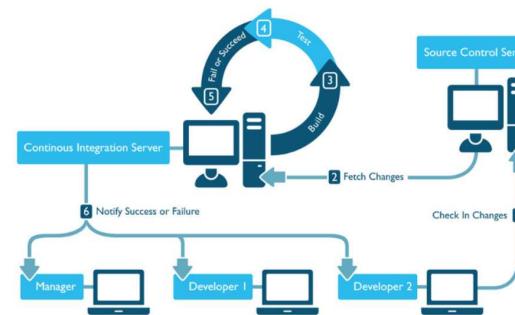
- ▶ Build and test process - on daily basis - integration issues found early and quickly
- ▶ Tests run daily - feedback on code quality is sent to the team
- ▶ Test results are visible to all team members - especially with automated reports
- ▶ Regression testing can be continuous throughout the iteration, covering as much functionality as possible
- ▶ Good coverage in regression tests helps building large integrated systems
- ▶ Regression automation creates time for the testers to concentrate manual testing on:
 - ▶ New features
 - ▶ Implemented changes
 - ▶ Confirmation testing of defect fixes



CI - Continuous quality control

Automated build tools are used to:

- ▶ Run Unit and Integration tests
- ▶ Run static and dynamic tests
- ▶ Measure and profile performance
- ▶ Extract and format documentation from the source code
- ▶ Facilitate manual quality assurance processes



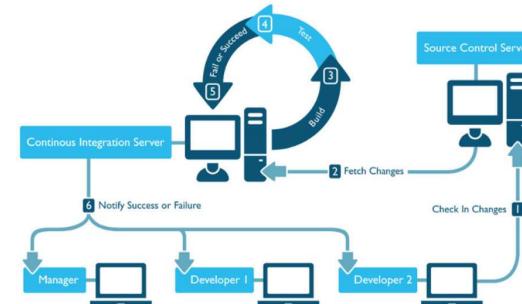
Objectives are to:

- ▶ Improve the quality of the product
- ▶ Reduce the time to deliver (no longer quality control AFTER the development is done)

CI - Automatic deployments

Build tools are often linked to automatic deployment tools, which:

- ▶ Fetch the appropriate build from the CI or build server
- ▶ Deploy this build into one or more environments:
 - ▶ Development
 - ▶ Test
 - ▶ Staging
 - ▶ Production



This process reduces errors and delays associated with having to rely on specialized staff or programmers to install releases.

CI - Possible benefits

- ▶ Allows earlier detection and root cause analysis of integration problems
- ▶ Gives the development team regular feedback on whether the code is working
- ▶ Keeps the version of the software being tested within a day of the version being developed
- ▶ Reduces regression risk associated with developer code being refactored
- ▶ Provides confidence that each day's development work is based on a solid foundation
- ▶ Makes progress toward the completion of the product increment visible
- ▶ Eliminates the schedule risks associated with big-bang integration
- ▶ Provides constant availability of executable software throughout the sprint
- ▶ Reduces repetitive manual testing activities
- ▶ Provides quick feedback on decisions made to improve quality and tests

CI - Risks and challenges

- ▶ Continuous integration tools have to be introduced and maintained
- ▶ The continuous integration process must be defined and established
- ▶ Test automation requires additional resources and can be complex
- ▶ Thorough test coverage is essential to achieve automated testing advantages
- ▶ Teams sometimes over-rely on unit tests and perform too little system and acceptance testing

Tools are needed for Continuous Integration:

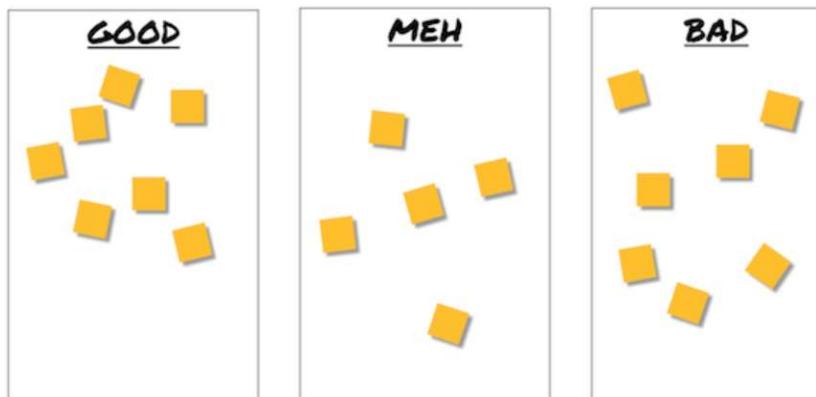
- ▶ Testing
- ▶ Automating the build process
- ▶ Version control



Retrospectives and Process Improvements

Retrospectives explained

- ▶ A meeting at the end of each project milestone about:
 - ▶ What went well?
 - ▶ What did not go well and could be improved?
 - ▶ How to keep the good and improve the bad in the future?
- ▶ Aka ‘post-project’ meetings or ‘project retrospectives’
- ▶ Timing and organization fully depends on the SDLC model
- ▶ Participants include developers, testers, architects, product owners, business analysts, etc
- ▶ Topics include:
 - ▶ Process
 - ▶ People
 - ▶ Organizations
 - ▶ Relationships
 - ▶ Tools



Retrospectives - Benefits

If regularly held and follow-ups occur, retrospectives can result in:

- ▶ Better self-organization
- ▶ Continuous improvement of development and testing
- ▶ Test-related improvements focused on:
 - ▶ Increased test effectiveness and efficiency
 - ▶ Increased quality of testware
 - ▶ Team bonding and learning
 - ▶ Improved quality of the test basis
 - ▶ Better cooperation between developers and testers
- ▶ Better testability of:
 - ▶ Applications
 - ▶ User stories
 - ▶ Features
 - ▶ System interfaces

Retrospectives - Effectivity

- ▶ Timing depends on the SDLC used and the organization
- ▶ Root cause analysis of defects can drive development and testing improvements
- ▶ The nr of improvements should be limited to allow for continuous improvement
- ▶ The results should be recorded and can be part of the test completion report
- ▶ Recommended improvements should be followed up
- ▶ The meeting is run by a facilitator
- ▶ Should occur in an environment based on trust
- ▶ All team-members can provide input on both testing and non-testing activities
- ▶ Testers play an important role in retrospectives due to:
 - ▶ Testers are part of the team
 - ▶ Testers have a unique perspective
 - ▶ Testing occurs in each sprint and contributes to success

Retrospectives - Success factors

Similar to the success factors of reviews:

- ▶ Defects found are welcomed and expressed objectively
- ▶ People issues and psychological aspects are dealt with
- ▶ Checklists or roles are used if appropriate
- ▶ Training is given in techniques used
- ▶ Management support



Test Levels

Introduction

Test Levels

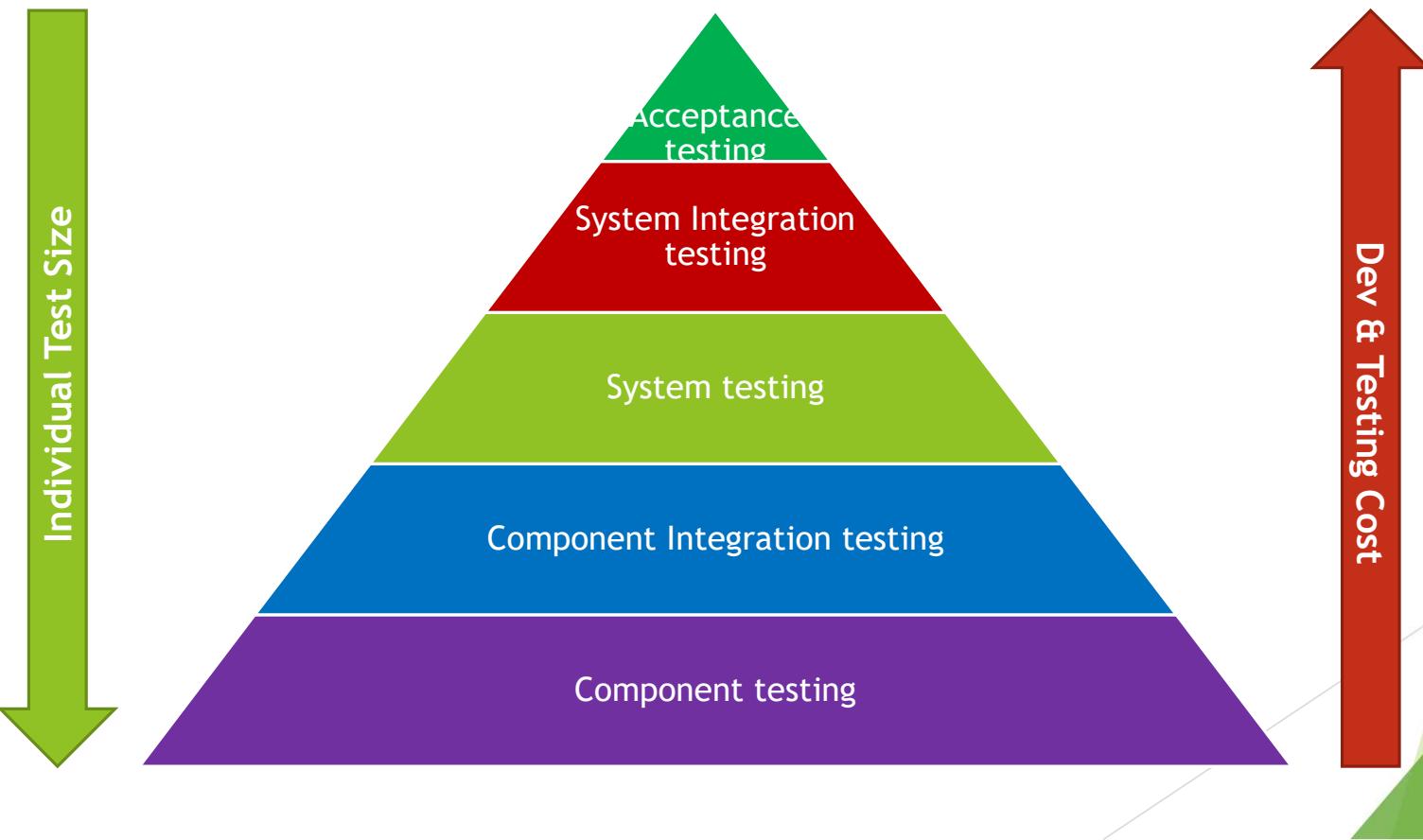
- ▶ Groups of test activities that are organized and managed together
- ▶ Performed in relation to software at a given level of development
- ▶ Identify missing areas and prevent overlap and repetition of tests in the SDLC
- ▶ Implementation of Test levels depends on the SDLC
- ▶ Test levels described are:
 - ▶ Component testing
 - ▶ Component Integration testing
 - ▶ System testing
 - ▶ System Integration Testing
 - ▶ Acceptance testing

Test Levels (2)

- ▶ Each test level requires a suitable test environment
- ▶ Each test level has certain attributes:

- | | |
|--|--------------------------------------|
| ▶ Specific Objectives | Why |
| ▶ Test Basis | Deriving Test Cases |
| ▶ Test Object | What |
| ▶ Typical defects and failures | What to expect |
| ▶ Specific approaches and responsibilities | How to achieve the objectives |

The Test Pyramid



FL - 2.2.1 K2

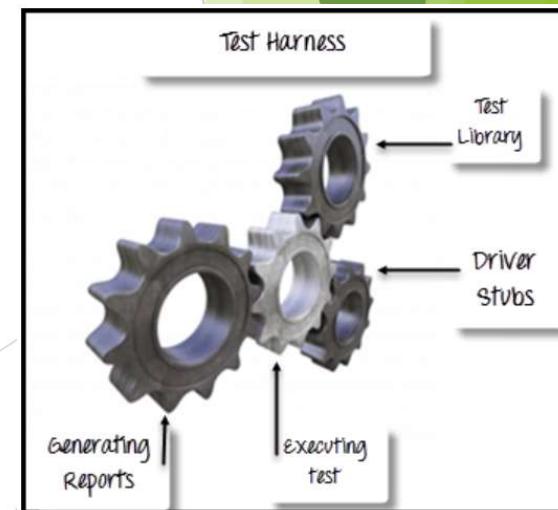


Test Levels

Component (Unit) Testing

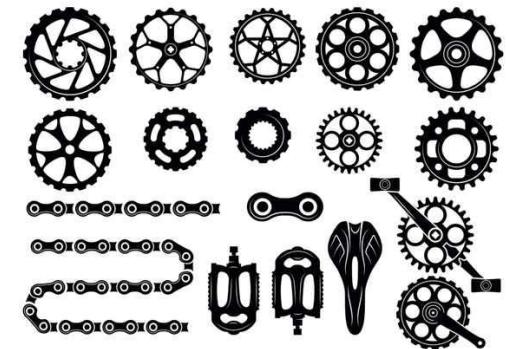
Component Testing - General

- ▶ Aka unit testing, module testing
- ▶ Focuses on software components that are separately testable
- ▶ Components are tested in isolation and **stubs** and **drivers** are used to mimic the interface between components
- ▶ **Test harness** - A test environment comprised of stubs and drivers needed to execute a test - find middleware issues
- ▶ Automated component regression is very important



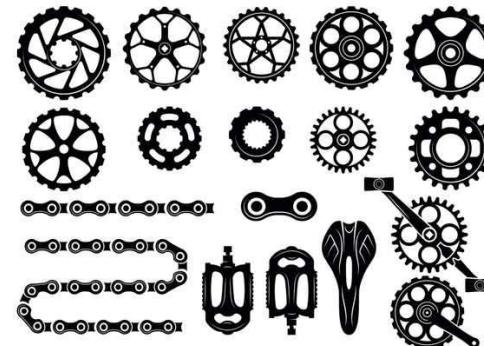
Component Testing - Objectives

- ▶ Reducing risk
- ▶ Verifying whether the functional and non-functional behavior of the component is as designed and specified
- ▶ Building confidence in the quality of the component
- ▶ Finding defects in the component
- ▶ Preventing defects from getting into higher test levels



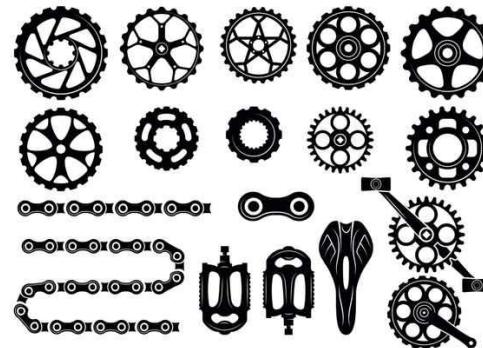
Component Testing - Test Basis

- ▶ Examples of where the tests can be derived from are:
 - ▶ Detailed design
 - ▶ Code
 - ▶ Data model
 - ▶ Component specifications



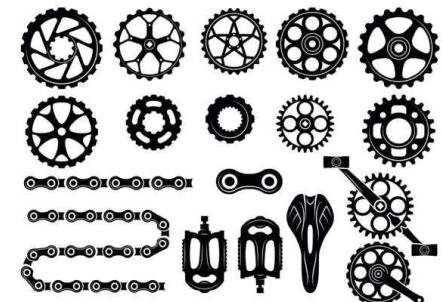
Component Testing - Test Object

- ▶ Typical test objects are:
 - ▶ Components, units or modules
 - ▶ Code and data structures
 - ▶ Classes
 - ▶ Database modules



Component Testing - Typical defects & failures

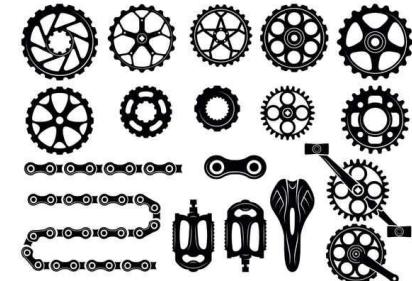
- ▶ Incorrect functionality (not as described in the test basis)
- ▶ Data flow problems
- ▶ Incorrect code and logic



Defects are usually fixed as soon as they are found without formal defect management. Found defects gives proper information for root cause analysis and process improvements

Component Testing - Approach and responsibilities

- ▶ Usually executed by the developer who wrote the code
- ▶ Requires access to the tested code
- ▶ Tests can be written after the code, or vice versa (Agile)
- ▶ TDD - first write tests, then write code





Test Levels

Component Integration
Testing

Component Integration Testing - General

- ▶ Aka Unit Integration Testing
- ▶ Focuses on interfaces and interactions between components
- ▶ Interactions and interfaces between integrated components
- ▶ Performed after Component testing - mainly by developers
- ▶ In general an automated process
- ▶ Part of the Continuous Integration process



Integration Testing - Objectives

- ▶ Reducing risk
- ▶ Verifying whether the functional and non-functional behavior of the interfaces is as designed and specified
- ▶ Building confidence in the quality of the interfaces - mainly through automation
- ▶ Finding defects in the interfaces, interactions or components
- ▶ Preventing defects from getting into higher test levels
- ▶ Prevent regression of existing interfaces, interactions or components using automated regression tests



Component Integration Testing - Test Basis

- ▶ Examples of where the tests can be derived from are:
 - ▶ Software and design
 - ▶ Sequence diagrams
 - ▶ Interface and communication protocol specifications
 - ▶ Architecture at component level



Component Integration Testing - Test Object

► Typical test objects are:

- Databases
- Infrastructure
- Interfaces
- APIs
- Microservices
- Modules



Component Integration Testing - Typical defects & failures

- ▶ Incorrect data, missing data or incorrect data encoding
- ▶ Incorrect sequencing or timing of interface calls
- ▶ Interface mismatch
- ▶ Failures in communication between components
- ▶ Unhandled or improperly handled communication failures between components
- ▶ Incorrect assumptions about the meanings, units or boundaries of the data being passed between components



Component Integration Testing - Approach and responsibilities

- ▶ Should focus on the integration itself, not on the individual modules
- ▶ Functional, non-functional and structural test types are applicable
- ▶ Best practice is to first plan integration tests and integration strategy before building the components
- ▶ Component integration testing heavily depends on the Integration strategy (top-down, bottom-up, big bang, incremental)
- ▶ Risk analysis helps to focus component integration testing
- ▶ Greater scope means more difficulty in isolating defects - Continuous Integration including test automation can help



Component Integration Strategies

- ▶ **Top - Down**
 - ▶ Testing takes place from the GUI down. Components or systems are substituted by stubs
- ▶ **Bottom up**
 - ▶ Testing takes place from the bottom of the control flow upward. Components or systems are substituted by drivers
- ▶ **Functional Incremental**
 - ▶ Integration and testing takes place based on the functions or functionalities as per the functional specification document.
- ▶ **Big-Bang**
 - ▶ All units are linked at once, resulting in a complete system.





Test Levels

System Testing

System Testing - General

- ▶ Focuses on the behavior and capabilities of a whole system or product
- ▶ Looks at the functional end-to-end tasks the system can perform and the non-functional behavior when performing these tasks
- ▶ Automated system regression tests provide confidence
- ▶ Often produces information for the stakeholders to make release decisions
- ▶ May also satisfy legal or regulatory standards or requirements
- ▶ Test environment should be as close as possible to the production environment spec



System Testing - Objectives

- ▶ Reducing risk
- ▶ Verifying whether the functional and non-functional behavior of the system is as designed and specified
- ▶ Validating that the system is complete and will work as expected
- ▶ Building confidence in the quality of the system as a whole
- ▶ Finding defects
- ▶ Preventing defects from getting into higher test levels or production
- ▶ In some cases - verifying data quality



System Testing - Test Basis

- ▶ Examples of where the tests can be derived from are:
 - ▶ System and software requirements
 - ▶ Risk Analysis reports
 - ▶ Use cases
 - ▶ Epics and user stories
 - ▶ Models of system behavior
 - ▶ State diagrams
 - ▶ System and user manuals



System Testing - Test Object

- ▶ Typical test objects are:
 - ▶ Applications
 - ▶ Hardware / Software systems
 - ▶ Operating systems
 - ▶ System under test (SUT)
 - ▶ System configuration and configuration data



System Testing - Typical defects & failures

- ▶ Incorrect calculations
- ▶ Incorrect or unexpected functional or non-functional system behavior
- ▶ Incorrect control or data flows within the system
- ▶ Failure to properly and completely carry out end-to-end functionality
- ▶ Failure of the system to work properly in the production environment
- ▶ Failure of the system to work as described in the system and user manuals



System Testing - Approach and responsibilities

- ▶ Should focus on the overall, end-to-end behavior of the system as a whole
- ▶ Includes **functional** and **non-functional**
- ▶ Can be tested using specification-based techniques (black box) or structure-based techniques (white-box)
- ▶ Typically performed by independent testers
- ▶ Specification defects can lead to misunderstanding of expected system behavior, which can lead to false positives or false negatives
- ▶ Early involvement of testers reduces the risk of above situations occurring





Test Levels

System Integration Testing

System Integration Testing - General

- ▶ Focuses on interactions between systems, packages and microservices (also external)
- ▶ External dependencies can cause challenges
- ▶ Can be performed after or during System testing - mainly by testers
- ▶ Test environments should be as close to the operational environment as possible



System Integration Testing - Objectives

- ▶ Reducing risk
- ▶ Verifying whether the functional and non-functional behavior of the integrated system(s) is as designed and specified
- ▶ Building confidence in the quality of the system
- ▶ Finding defects in the system(s)
- ▶ Preventing defects from getting into the acceptance level
- ▶ Prevent regression of existing systems using (automated) regression tests



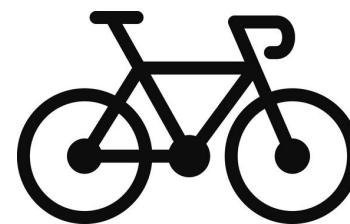
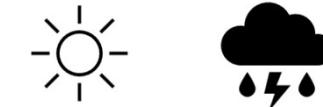
System Integration Testing - Test Basis

- ▶ Examples of where the tests can be derived from are:
 - ▶ System design
 - ▶ Use cases
 - ▶ Architecture at system level
 - ▶ Workflows
 - ▶ External interface definitions



System Integration Testing - Test Object

- ▶ Typical test objects are:
 - ▶ Subsystems
 - ▶ Interfaces between systems
 - ▶ Microservices
 - ▶ Modules



System Integration Testing - Typical defects & failures

- ▶ System interface mismatch
- ▶ Failures in communication between systems
- ▶ Unhandled or improperly handled communication failures between systems
- ▶ Incorrect assumptions about interactions between systems



System Integration Testing - Approach and responsibilities

- ▶ Should focus on the integration itself, not on individual systems
- ▶ Functional, non-functional and structural test types are applicable
- ▶ Best practice is to first plan integration test and integration strategy before building the systems
- ▶ System integration testing heavily depends on the Integration strategy (top-down, bottom-up, big bang, incremental)
- ▶ Risk analysis helps to focus system integration testing
- ▶ Greater scope means more difficulty in isolating defects - Continuous Integration including test automation can help





Test Levels

Acceptance Testing

Acceptance Testing - General

- ▶ Focuses on the behavior and capabilities of a whole system or product
- ▶ Demonstrating the readiness for release, that the user's business needs are fulfilled



Acceptance Testing - Objectives

- ▶ Establishing confidence in the quality of the system as a whole
- ▶ Validating that the system is complete and will work as expected
- ▶ Verifying that functional and non-functional behavior of the system is as specified
- ▶ May also satisfy legal or regulatory standards or requirements
- ▶ Finding defects is often NOT an objective and can be considered a project risk



Types of Acceptance Testing

- ▶ Different typical acceptance tests are:
 - ▶ **User Acceptance Test**
 - ▶ Building confidence that the users can use the system with minimum difficulty, cost and risk
 - ▶ **Operational Acceptance Test**
 - ▶ Building confidence that the system administrators can keep the system working in production
 - ▶ **Contract and regulation Acceptance Test**
 - ▶ Building confidence that contractual or regulatory compliance has been achieved
 - ▶ **Alpha and Beta Test**
 - ▶ Building confidence amongst customers that they can use the system with minimum difficulty, cost and risk
 - ▶ Detection of defects related to the conditions and environments in which the system will be used



Acceptance Testing - Test Basis

- ▶ Examples of where the tests can be derived from are:
 - ▶ Business processes
 - ▶ User or business requirements
 - ▶ Regulations, legal contracts and standards
 - ▶ Use cases
 - ▶ System requirements
 - ▶ System or user documentation
 - ▶ Installation procedures
 - ▶ Risk analysis reports



Acceptance Testing - Test Basis (2)

- ▶ Examples of where the tests can be derived from for **Operational Acceptance Testing** are:
 - ▶ Backup and restore procedures
 - ▶ Disaster recovery procedures
 - ▶ Non-functional requirements
 - ▶ Operations documentation
 - ▶ Deployment and installation instructions
 - ▶ Performance targets
 - ▶ Database packages
 - ▶ Security standards or regulations



Acceptance Testing - Test Object

- ▶ Typical test objects are:
 - ▶ System under test
 - ▶ System configuration and configuration data
 - ▶ Business processes for a fully integrated system
 - ▶ Recovery systems and hot sites
 - ▶ Operational and maintenance processes
 - ▶ Forms
 - ▶ Reports
 - ▶ Existing and converted production data



Acceptance Testing - Typical defects & failures

- ▶ System workflows do not meet business or user requirements
- ▶ Business rules are not implemented correctly
- ▶ System does not satisfy contractual or regulatory requirements
- ▶ Non-functional failures such as:
 - ▶ Security vulnerabilities
 - ▶ Inadequate performance under high loads
 - ▶ Improper operation on a supported platform



Acceptance Testing - Approach and responsibilities

- ▶ Often the responsibility of the customers, business users, product owners or operators
- ▶ Mostly the final test level in a sequential development lifecycle
- ▶ Can also occur at other times:
 - ▶ COTS software - when installed or integrated
 - ▶ New functional enhancement - before system testing
- ▶ Iterative development
 - ▶ various forms of acceptance testing during and at the end of each iteration
 - ▶ Alpha and Beta tests may be done at the end of an iteration or after a series of iterations
 - ▶ User, Operational, regulatory and contractual acceptance testing can be done at the end of each iteration, after the completion of each iteration or after a series of iterations



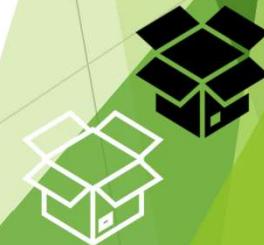


Test Types

Test Types

A group of test activities aimed at specific characteristics of a software system or a part of the system. A test type is based on specific test objectives:

- ▶ Evaluating the **functions** the system or component should perform
- ▶ Evaluating **non-functional** quality characteristics
- ▶ Checking the system's behavior against its' specifications
- ▶ Cover the underlying structure by the tests to an acceptable level



Test Types - General

- ▶ Each test type can be applied to all test levels
 - ▶ Focus will be different per test level
- ▶ Test Techniques can be used to derive:
 - ▶ Test conditions
 - ▶ Test cases



Functional Testing

- ▶ Checking the functional completeness, functional correctness and functional appropriateness
- ▶ Evaluates the functions the system should perform
- ▶ The functions are ‘**what**’ the system does
- ▶ Mostly described in business requirement specs, epics, user stories, use cases
- ▶ Considers the external behavior of the software
- ▶ Thoroughness can be measured through functional coverage
- ▶ May involve specific skills or knowledge:
 - ▶ Business domain
 - ▶ Particular role the software serves



Non-Functional Testing

- ▶ Checking the non-functional software quality characteristics
- ▶ Evaluates attributes other than functional characteristics
- ▶ Non-functional testing tests ‘how well’ the system behaves
- ▶ Considers the external behavior of the software
- ▶ Mainly derived from functional tests
- ▶ Thoroughness can be measured through non-functional coverage
- ▶ Includes the following classification: performance efficiency, compatibility, usability, reliability, security, maintainability, portability (amongst others)
- ▶ Early testing is essential
- ▶ Very specific test environments are required
- ▶ May involve specific skills or knowledge:
 - ▶ Weaknesses of a design or technology
 - ▶ Particular use base



Black-box Testing

- ▶ Checking the system's behavior against its' specifications
- ▶ Specification-based test type
- ▶ Tests are derived from documentation external to the test object (specification documents, user stories, etc)



White-box Testing

- ▶ Cover the underlying structure by the tests to an acceptable level
- ▶ Structure-based test type
- ▶ Tests are derived from the system's internal structure or implementation (code, architecture, work-flows, data flows, etc)
- ▶ Considers the internal behavior of the software
- ▶ Measuring the thoroughness of testing through assessment of coverage of a type of structure - Code coverage
- ▶ Tools can be used to measure the code coverage of elements (like statements or decisions)



Test Types and Test Levels - examples

May include functional, non-functional, specification-based and structure-based testing and can be performed at all test levels

	Component	Component Integration	System	System Integration	Acceptance
Functional	How a component should calculate compound interest	How account info is passed from UI to business logic	How account holders can apply for credits	How an external microservice is used to check an account holder's credit score	How the banker handles approval or decline of the credit application
Non-functional	Nr of CPU cycles required to perform to calculate a complex interest percentage	Security tests for buffer overflow vulnerabilities due to data passed from UI to business logic	Portability tests to check whether the UI works on all supported browsers and mobile devices	Reliability tests to evaluate robustness if the credit score microservice fails to respond	Usability tests to evaluate accessibility of the banker's credit processing interface for people with disabilities
White-box	Tests to achieve complete statement and decision coverage for all components that perform financial calculations	Tests to exercise how each screen in the browser interface passes data to the next screen and to the business logic	tests to cover sequences of web pages that can occur during a credit line application	Tests to exercise all possible inquiry types sent to the credit score microservice	Tests to cover all possible financial data file structures and value ranges for bank to bank transfers
Black-box	In a new component, use Equivalence partitioning to check the acceptance of proper account numbers and denial of invalid ones	Tests to make sure that the account nr and account name components are working together	Tests to enter account information in the credit score system, tests to cover the entire process of credit approvals	Tests that will call and display information from the account microservice to the credit score microservice	Front end tests that will satisfy the user need to have all information for credit approvals



Confirmation Testing and Regression testing

Confirmation testing

May include functional, non-functional and structural testing and is performed at all test levels

► Confirmation testing (re-testing)

- ▶ Confirms that a defect has been successfully fixed
- ▶ The software version (including the fix) can be tested in several ways (based on RISK)
 - ▶ Re-executing all test that failed before because of the fixed defect
 - ▶ Adding new tests to cover any needed changes that were needed to fix the defect
- ▶ Might have introduced or uncovered different defects elsewhere in the software
- ▶ Might only be following the steps to trigger the failure and checking it does not occur (time, money)



Regression testing

- ▶ Confirms that a change has not caused adverse consequences in others areas
- ▶ Re-testing previously passed tests to prevent changes having caused defects in the component, other components, different systems or even the environment; to prevent the system from ‘regressing’
- ▶ Impact Analysis needed to identify potentially impacted areas on all test levels
- ▶ Regression test suites are created and executed when the software or environment are changed
- ▶ As these tests evolve slowly, increase constantly and are run many times, they are a good candidate for automation
- ▶ Regression automation should start as early as possible - possibly in CI pipelines



Change-related testing - Example

- ▶ System testing has started
- ▶ Tester finds a failure halfway through the system test
- ▶ Developer fixes the underlying defect and deploys it to the test environment
- ▶ Activities that should be done include:
 - ▶ An **Impact Analysis** to identify affected areas
 - ▶ Decide whether to re-execute that previously failed tests or add new tests
 - ▶ A **confirmation** test to make sure that the failure was indeed fixed
 - ▶ Targeted **regression** tests on previously passed tests in that area to make sure the bug-fix did not break anything else

Change-related testing - Example

	Component	Component Integration	System	System Integration	Acceptance
Change-related Testing	Automated regression for each component and included in the CI framework.	Tests to confirm fixes related defects as the fixes are checked in.	All tests for a workflow are re-executed if any screen on that workflow changes.	Tests of the application interacting with the credit scoring microservice are re-executed daily as part of continuous deployment.	All previously failed tests are re-executed after a defect found in acceptance testing is fixed.



Maintenance Testing

Maintenance Testing

- ▶ Testing the changes to an operational system or the impact of a changed environment on an operational system
- ▶ Changes are inevitable and include:
 - ▶ Corrective - Fixes of defects
 - ▶ Adaptive to changes in the environment
 - ▶ Preserve or improve non-functional quality aspects such as:
 - ▶ Performance
 - ▶ Compatibility
 - ▶ Reliability
 - ▶ Security
 - ▶ Portability
 - ▶ Maintainability

Maintenance Testing (2)

Can involve:

- ▶ Planned releases/ deployments
- ▶ Unplanned releases / deployments (hotfixes)
- ▶ Confirmation Testing
- ▶ Regression Testing

Common practice:

- ▶ Have maintenance slots available
- ▶ Use if needed, ignore if not needed

Triggers for Maintenance

Why software maintenance and maintenance testing takes place:

- ▶ Modifications (planned enhancements, corrective and (emergency) changes, patches for defects and vulnerabilities)
- ▶ Upgrades or migrations of the operational environment:
 - ▶ From one platform to another - tests in the new environment AND the changed software
 - ▶ Data from one application into the system being maintained
- ▶ Retirement of an application:
 - ▶ Data archiving
 - ▶ Restore and retrieval procedures

Impact Analysis for Maintenance

Evaluates changes made to:

- ▶ Identify intended consequences
- ▶ Identify expected and possible side effects
- ▶ Identify areas in the system that will be affected by the change
- ▶ Help to identify the impact of a change on existing tests

Side effects and affected areas need to be tested for regressions, which might mean first updating any existing tests affected by the change

Impact Analysis for Maintenance (2)

- ▶ May be done before a change is made and can be taken into account to determine whether a change **should** be made
- ▶ Impact Analysis can be difficult if:
 - ▶ Specifications are old or missing
 - ▶ Test cases are old or not documented
 - ▶ There is no maintained bi-directional traceability
 - ▶ Tool support is weak or non-existent
 - ▶ People involved do not have domain and/or system knowledge
 - ▶ Insufficient attention has been paid to the software's maintainability during development

Scope of Maintenance Testing

Depends on:

- ▶ The degree of risk to the system
- ▶ The size of the existing system
- ▶ The size of the change

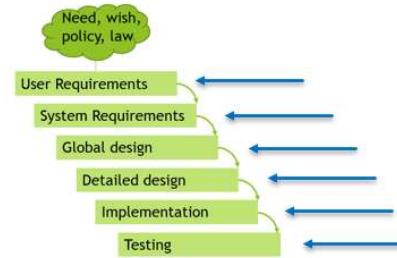


Testing Throughout the Software Development Lifecycle

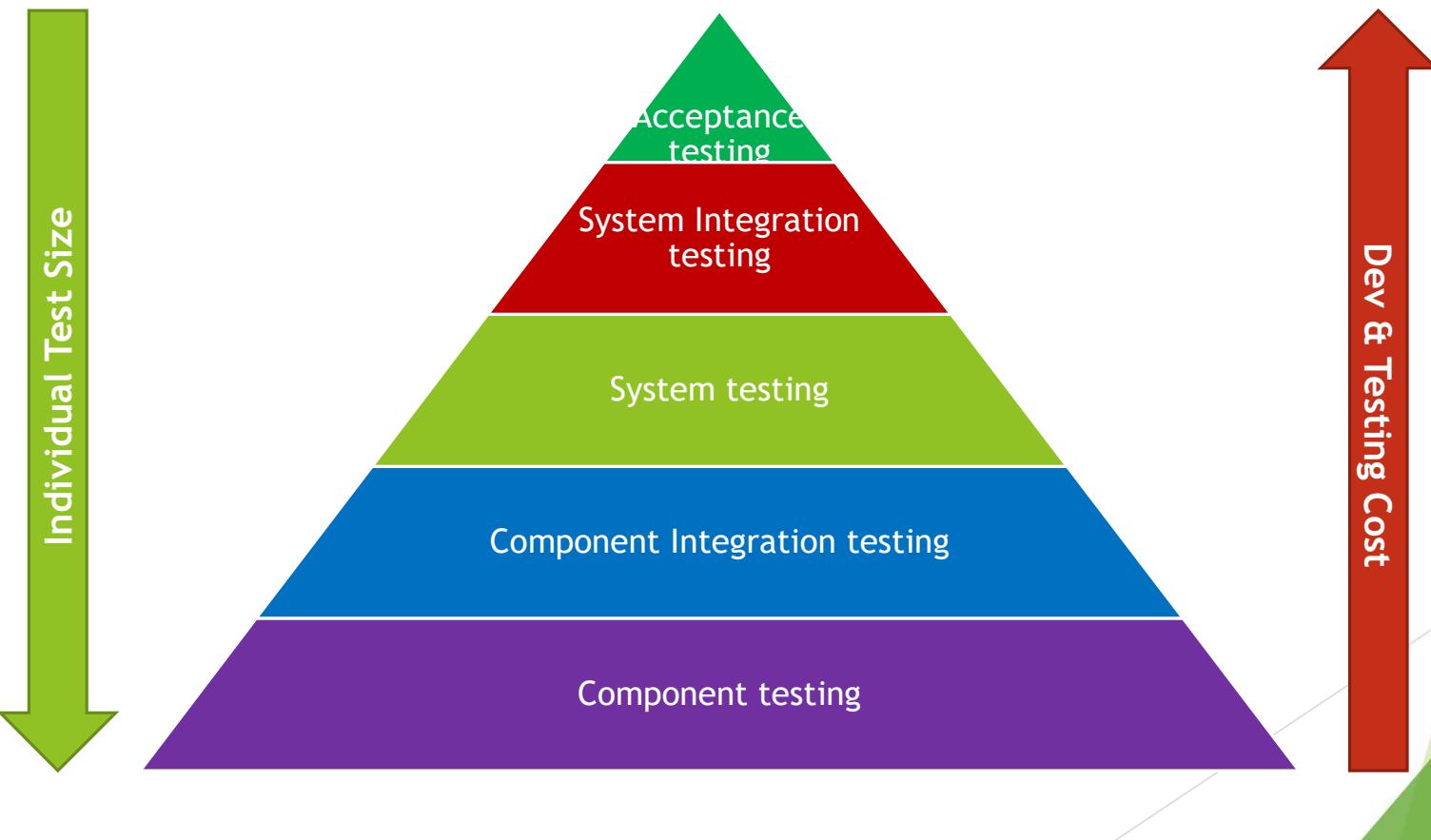
Summary - Keywords Explained

General

- ▶ Shift-left
 - ▶ An approach to performing testing and quality assurance activities as early as possible in the software development lifecycle.
- ▶ Test object
 - ▶ The work product to be tested.
- ▶ Test level
 - ▶ A specific instantiation of a test process.
- ▶ Test type
 - ▶ A group of test activities based on specific test objectives aimed at specific characteristics of a component or system.



Test Levels



Test Levels (2)

- ▶ Component testing (Unit testing)
 - ▶ A test level that focuses on individual hardware or software components.
- ▶ Component integration testing (Unit integration testing)
 - ▶ The integration testing of components.
- ▶ System testing
 - ▶ A test level that focuses on verifying that a system as a whole meets specified requirements.
- ▶ System integration testing
 - ▶ The integration testing of systems.
- ▶ Integration testing
 - ▶ A test level that focuses on interactions between components or systems.
- ▶ Acceptance testing
 - ▶ A test level that focuses on determining whether to accept the system.

Test Types

- ▶ Functional testing
 - ▶ Testing performed to evaluate if a component or system satisfies functional requirements.
- ▶ Non-functional testing
 - ▶ Testing performed to evaluate that a component or system complies with non-functional requirements.
- ▶ White-box testing
 - ▶ Testing based on an analysis of the internal structure of the component or system.
- ▶ Black-box testing
 - ▶ Testing based on an analysis of the specification of the component or system.

Change-related testing

- ▶ Confirmation testing
 - ▶ A type of change-related testing performed after fixing a defect to confirm that a failure caused by that defect does not reoccur.
- ▶ Regression testing
 - ▶ A type of change-related testing to detect whether defects have been introduced or uncovered in unchanged areas of the software.
- ▶ Maintenance testing
 - ▶ Testing the changes to an operational system or the impact of a changed environment to an operational system.



Static Testing

Introduction

Static testing

3.1 Static Testing Basics

- ▶ FL-3.1.1 (K1) Recognize types of products that can be examined by the different static test techniques
- ▶ FL-3.1.2 (K2) Explain the value of static testing
- ▶ FL-3.1.3 (K2) Compare and contrast static and dynamic testing

3.2 Feedback and Review Process

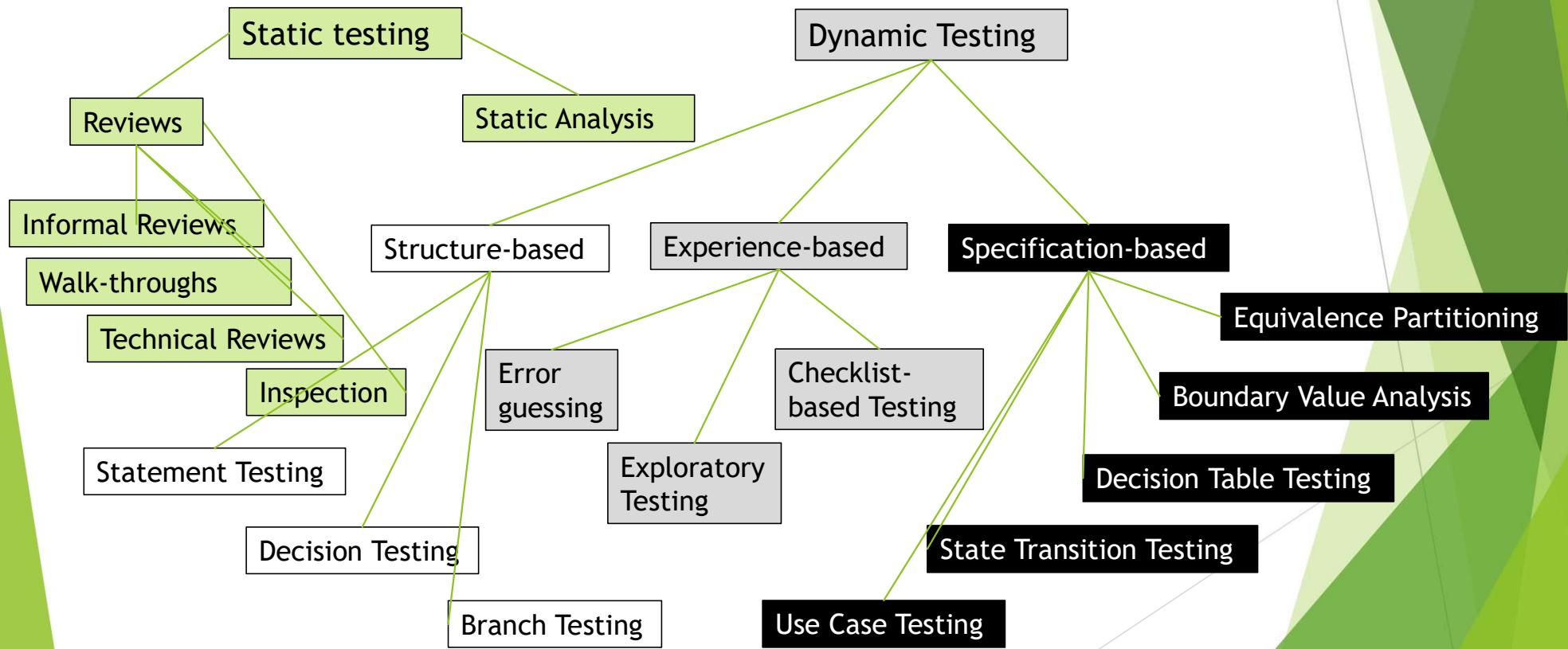
- ▶ FL-3.2.1 (K1) Identify the benefits of early and frequent stakeholder feedback
- ▶ FL-3.2.2 (K2) Summarize the activities of the review process
- ▶ FL-3.2.3 (K1) Recall which responsibilities are assigned to the principal roles when performing reviews
- ▶ FL-3.2.4 (K2) Compare and contrast the different review types
- ▶ FL-3.2.5 (K1) Recall the factors that contribute to a successful review



Static Testing

Basics

Testing Techniques - visualized



Static testing - overview

*Testing the software under test **without** executing the software.*

- ▶ Relies on manual examination (**reviews**) and automated analysis (**static analysis**) of the code or any other work products
- ▶ Main objectives:
 - ▶ Improving quality
 - ▶ Detecting defects
 - ▶ Preventing defects from being coded
- ▶ Assessing quality characteristics
- ▶ Both in Verification and Validation

Readability

Completeness

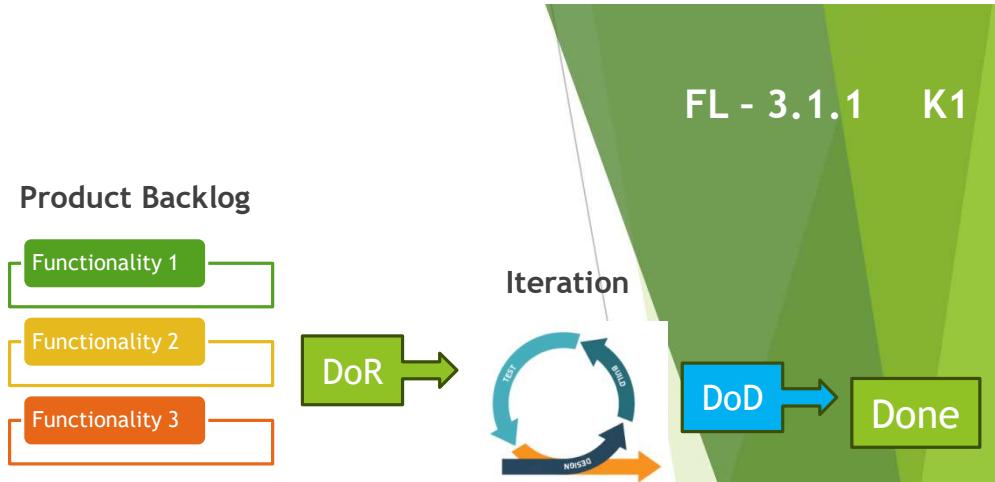
Consistency

Correctness

Testability

Static testing

- ▶ Testers, business representatives and developers
- ▶ Work together in their evaluation of work products
- ▶ Ensure that work products meet defined criteria, the ‘Definition of Ready’
- ▶ Review techniques can be applied to ensure:
 - ▶ User stories are complete and understandable and include testable acceptance criteria.
 - ▶ By asking the right questions, testers explore, challenge and help improve the proposed user stories.



Static Analysis

- ▶ The process of evaluating a component or system without executing it
- ▶ Identify issues BEFORE dynamic testing
- ▶ Less effort needed because:
 - ▶ No tests are needed
 - ▶ Tools are used (i.e. spelling checkers, readability tools, etc)
- ▶ Often implemented in the CI Framework
- ▶ Typically used to:
 - ▶ Detect specific code defects
 - ▶ Evaluate maintainability
 - ▶ Evaluate security



Static Analysis by tools

- ▶ Tools analyze program code (e.g. control flow and data flow) as well as generated output (e.g. html, xml)
- ▶ Tools are mostly used by developers (before or during component and integration testing) and designers (during software modelling). Compilers can also be seen as SA tool.
- ▶ Typical defects found are:

Referencing a variable with an undefined value	Syntax violations of code and software models
Inconsistent interfaces between modules and components	Highly complex functions
Improper declaration of variables	Programming standards violations
Unreachable (dead) code that should be removed	Security vulnerabilities
Missing or erroneous logic (e.g, infinite loops)	



Work Products Examinable by Static Testing

Static testing - For what it can be used

Static testing can be used on almost any work product that can be **read and understood**:

- ▶ Specifications & requirements
- ▶ Epics, user stories, acceptance criteria
- ▶ Architecture and design specifications
- ▶ Source code
- ▶ Testware
- ▶ User guides
- ▶ Project documentation
- ▶ Contracts, project plans, schedules, budgets
- ▶ Models (such as activity diagrams)



Static Analysis - For what it can be used

Static analysis can be used on any work product:

- ▶ That has a formal structure (mainly code and models);
- ▶ For which an appropriate static analysis tool exists;

Static Analysis:

- ▶ Is ideally performed before formal reviews
- ▶ Goal is to find defects rather than failures

Static Testing - When can it NOT be used

- ▶ Work Products that can not be read or understood
- ▶ Work products that are difficult to interpret by human beings
- ▶ Work Products that should not be analyzed by tools:
 - ▶ Code
 - ▶ Models





Value of Static Testing

Static Testing - Benefits

- ▶ Early defect detection and correction
- ▶ More efficient defect detection and correction
- ▶ Find defects that are not (easily) found by dynamic testing
- ▶ Preventing defects in design or coding
- ▶ Increased development productivity
- ▶ Reduced development time and cost
- ▶ Reduced testing time and cost
- ▶ Reducing total cost of quality due to fewer failures later in the lifecycle
- ▶ Evaluate the quality of and build confidence in work products
- ▶ Improved communication between team members



Static Testing vs Dynamic Testing

Static vs Dynamic testing



- ▶ Static testing finds **defects** in work products - Dynamic testing finds **failures** caused by defects when running the software
- ▶ Static testing is used to improve the **consistency** and **internal quality** - dynamic testing focuses on **externally visible** behavior
- ▶ Find different types of defects
- ▶ Static testing is possible on non-executable work products
- ▶ Both measure quality characteristics, BUT:
 - ▶ Static testing - quality characteristics **not** depending on executing code (code quality, maintainability, etc)
 - ▶ Dynamic testing - quality characteristics depending on executing code (performance efficiency, functionality, etc)

Typical defects found in Static Testing

Typical defects that are easier and cheaper to find with static testing compared to dynamic testing:

- ▶ Requirement defects
- ▶ Design defects
- ▶ Coding defects
- ▶ Deviations from standards
- ▶ Incorrect interface specifications
- ▶ Security vulnerabilities
- ▶ Gaps or inaccuracies in test basis traceability or coverage
- ▶ Most maintainability defects can only be found by static testing



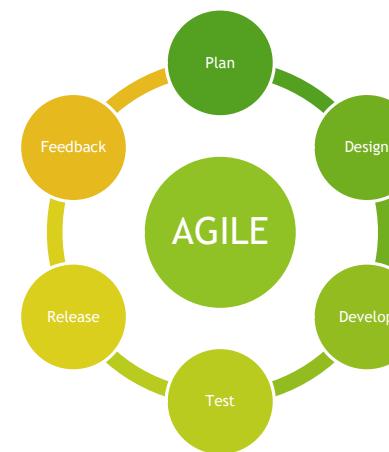


Static Testing

Feedback and Review
Process

What do you mean ‘early and frequent feedback’

- ▶ One of the cornerstones of Agile software development
- ▶ ‘Early Testing saves Time and Money’
- ▶ The Shift-Left Approach
- ▶ Test First approaches
- ▶ Continuous Integration
- ▶ Continuous Deployment



If you have to fail, **fail fast!**

Benefits of Early and Frequent Stakeholder Feedback

- ▶ Build the right product through early communication of quality issues
 - ▶ Stakeholders have expectations on what will be delivered
 - ▶ Expectations based on stakeholder vision **will** change during the project
 - ▶ Early and Continuous feedback **will** identify whether the product meets these expectations
 - ▶ If the expectations are not met, this can result in:
- ▶ Focus on the right items by preventing misunderstandings in requirements and priorities
 - ▶ Requirements and changes to requirements are often misunderstood
 - ▶ The earlier feedback is received on requirements **will** prevent misunderstandings
 - ▶ The team can improve their understanding
 - ▶ Focus on the right features

Deliver most value
to the stakeholders

Have the most
positive impact on
identified risks

Expensive
rework

Missed
deadlines

Blame
games

PROJECT
FAILURE



Review Process Activities

Review Process - General

- ▶ Generic review process described by the ISO/IEC 20246 standard
- ▶ Generic process is structured and flexible
- ▶ Any specific review process should be tailored to that specific situation
- ▶ More formality results in more tasks per activity
- ▶ Not all work products can be covered in one review. Depends on:
 - ▶ Size
 - ▶ Complexity



Review Process - Activities

The formal review process comprises the following main activities



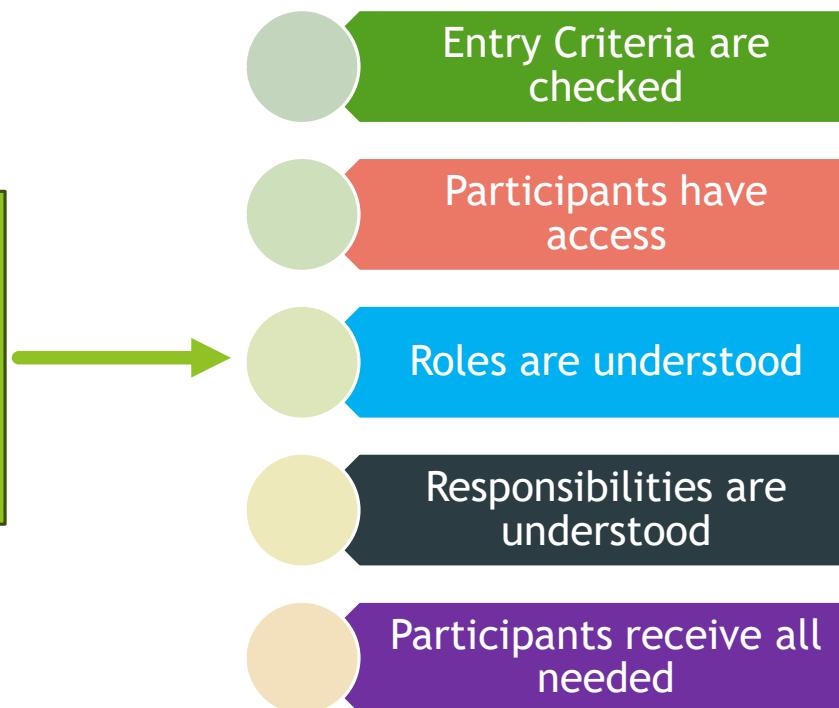
Review Process - Planning



-  Purpose
-  Work Product
-  Quality Characteristics
-  Focus Areas
-  Entry and Exit Criteria
-  Supporting information
-  Effort
-  Timeframes
-  Selecting the participants for the review and allocating the roles

Review Process - Review initiation

Goal is that
all is
prepared to
start the
review

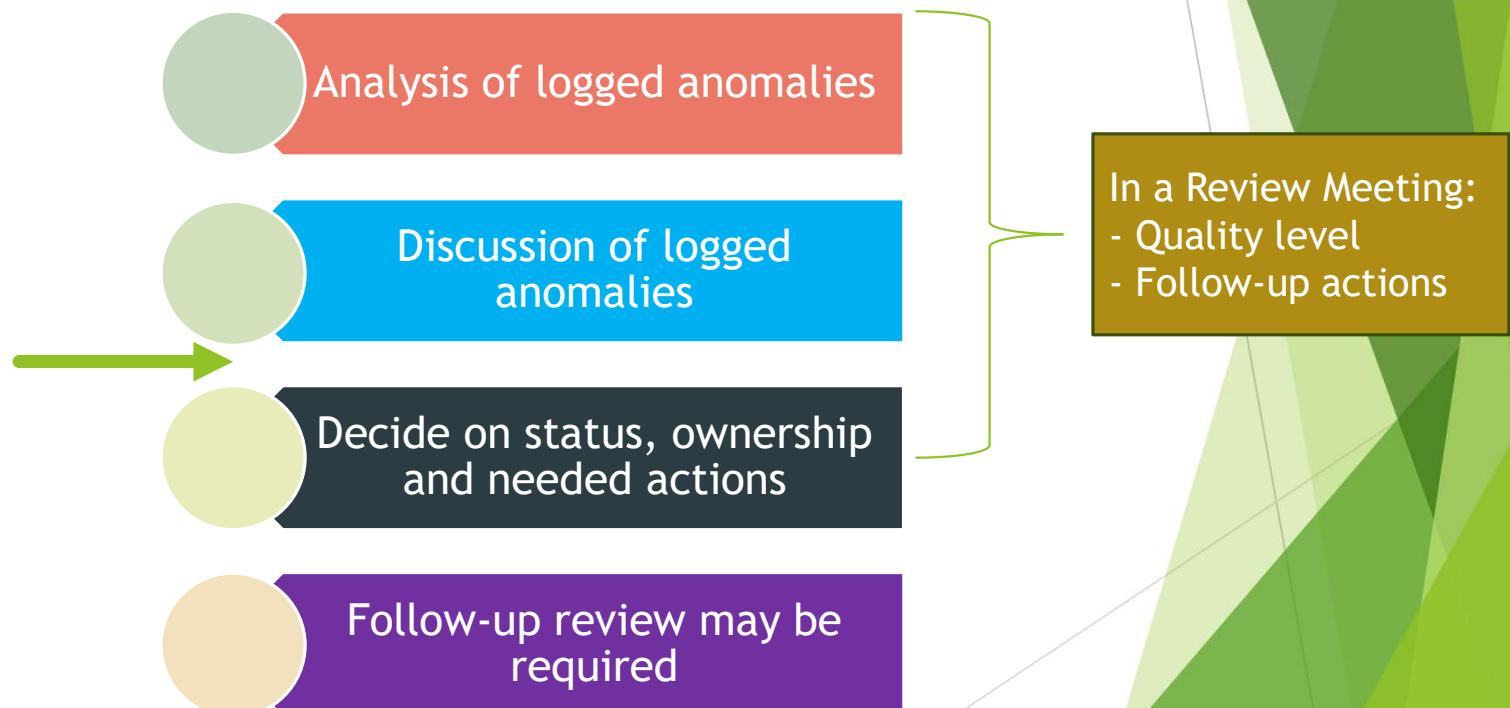


Review Process - Individual review



Review Process - Communication and analysis

**Analysis
and
Discussion**



Review Process - Fixing and reporting





Static Testing

Applying Review
Techniques

Types of Review Techniques

- ▶ Review techniques can be applied in the individual review activity
- ▶ Can be used across all the review types
- ▶ Effectiveness of any technique depends on the type of review used
- ▶ Review techniques can be:
 - ▶ Ad hoc
 - ▶ Checklist-based
 - ▶ Scenarios and dry runs
 - ▶ Role-based
 - ▶ Perspective-based

Review Techniques - Ad hoc

- ▶ Little or no guidance on how the task should be performed
- ▶ Review of the work product is done sequentially - identifying and documenting issues as they are encountered
- ▶ Commonly used
- ▶ Needs little preparation
- ▶ Highly dependent on skills of the reviewer
- ▶ May lead to duplicate issues reported by different reviewers

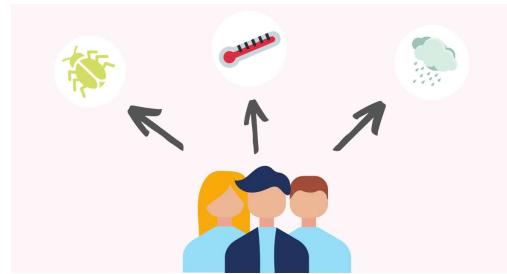
Review Techniques - Checklist-based

- ▶ Systematic technique - detect issues based on checklists provided by the initiator
 - ▶ Consists of a set of questions based on potential defects (derived from experience)
 - ▶ Checklists should be specific to the type of work product being reviewed
 - ▶ Checklists should be maintained regularly (learning from the past)
 - ▶ Main advantage - systematic coverage of typical defect types
 - ▶ Reviewers should not only limit the review to the checklist but also look for other defects



Review Techniques - Scenarios and dry runs

- ▶ Structured guidelines on how to read through the work product
- ▶ Supports reviewers to perform 'dry runs' based on the expected usage of the work product
- ▶ Above is only possible if the work product is documented appropriately (i.e. use cases)
- ▶ Scenarios provide better guidelines to the reviewers than checklists
- ▶ Reviewers should not only limit the review to the scenarios but also look for other defects



Review Techniques - Role-based

- ▶ Reviewers evaluate the work product from the perspective of individual stakeholder roles
- ▶ Typical roles are specific end user types and roles in the organization such as:
 - ▶ Experienced, inexperienced
 - ▶ Child, senior
 - ▶ User administrator
 - ▶ System administrator
 - ▶ Performance tester

Review Techniques - Perspective-based

- ▶ Similar to the role-based technique - reviewers take on different stakeholder viewpoints
- ▶ Typical stakeholder viewpoints include end user, marketing, designer, tester, operations
- ▶ Leads to more depth in reviewing with less duplication
- ▶ Reviewers should try to create the product from the work product under review (tester - acceptance tests)
- ▶ Checklists are expected to be used
- ▶ Most effective technique for reviewing requirements and technical work products



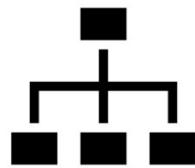
Roles & Responsibilities in Reviews

Reviews - Roles & Responsibilities



Reviews - Roles & Responsibilities - Manager

Role	Responsibility
Manager	<ul style="list-style-type: none">- Decides on what is to be reviewed- Decides on the planning of the review- Allocates staff, budget and time- Monitors ongoing cost-effectiveness- Executes control decisions in the event of inadequate outcomes



Reviews - Roles & Responsibilities - Author

Role	Responsibility
Author	<ul style="list-style-type: none">- Creates the work product under review- Fixes defects in the work product under review



Reviews - Roles & Responsibilities - Moderator

Role	Responsibility
Moderator / Facilitator	<ul style="list-style-type: none">- Ensures effective running of review meetings- Mediates between the various points of view- Plans the review with regards to time and time management- Ensures a safe review environment- Often the person upon whom the success of the review depends



Reviews - Roles & Responsibilities - Scribe

Role	Responsibility
Scribe / Recorder	<ul style="list-style-type: none">- Collects and documents all anomalies found during the individual review activity by the reviewers- Records new anomalies, open points and decisions from the review meeting



Often, there is no need for a scribe except in more official review types.

Reviews - Roles & Responsibilities - Reviewer

Role	Responsibility
Reviewer	<ul style="list-style-type: none">- Reviewing of the work products- Can be a project participant, subject matter expert, any other stakeholder- Identify and log anomalies in the work product under review- May represent different perspectives



Reviews - Roles & Responsibilities - Review Leader

Role	Responsibility
Review leader	<ul style="list-style-type: none">- Takes overall responsibility for the review- Decides who will be involved and organizes when and where it will take place



One person may play more than one role and activities per role may vary.



Review Types

Review Process - General

- ▶ Range from informal to formal
- ▶ Informal reviews are not documented and don't follow a defined process
- ▶ **Formal** reviews follow a formal process including documentation and are characterized by:
 - ▶ Team participation
 - ▶ Documented results
 - ▶ Documented review procedures

Review Process - Formal or Informal?

- ▶ Formality depends on:
 - ▶ Software development lifecycle model
 - ▶ Maturity of the development process
 - ▶ Complexity of the work product to be reviewed
 - ▶ Legal or regulatory requirements
 - ▶ Need for an audit trail
- ▶ Focus depends on the agreed objectives of the review

Types of Reviews

- ▶ Each review type has several main characteristics
- ▶ A work product can be subject to multiple review types
- ▶ Defect types vary depending on:
 - ▶ The work product being reviewed
 - ▶ Objectives of the review
 - ▶ Project needs
 - ▶ Available resources
 - ▶ Work product type
 - ▶ Business domain
 - ▶ Company culture



Informal Review

Aka	Buddy check, pairing, pair review
Main purpose	Detecting anomalies
Possible additional purposes	Generating new ideas or solutions, quickly solving minor problems
Characteristics	<ul style="list-style-type: none">- Not based on a formal documented process- May not involve a review meeting- May be performed by a colleague of the author or by more people- Results may be documented- Varies in usefulness depending on the reviewers- Use of checklists is optional- Very common in Agile development

Walkthrough

Main purpose	Find anomalies, evaluate quality, building confidence in the work product, improve the software product, consider alternative implementations, evaluate conformance to standards and specifications
Possible additional purposes	Exchanging new ideas about techniques or style variations, educating reviewers, gaining consensus, motivate authors to improve and detect anomalies
Characteristics	<ul style="list-style-type: none">- Individual preparation before the review meeting is optional- Review meeting is typically led by the author of the work product- Use of checklists is optional- May take the form of scenarios, dry runs or simulations- Potential defect logs and review reports may be produced- May vary from quite informal to very formal

Technical Review

Main purpose	Detecting potential defects, gaining consensus, make decisions on technical problems
Possible additional purposes	Evaluating quality and building confidence in the work product, generating new ideas, motivating and enabling authors to improve future work products, considering alternative implementations
Characteristics	<ul style="list-style-type: none">- Reviewers should be technical peers of the author and technical experts- Individual preparation before the review meeting is required- Led by a trained moderator (not the author)- Use of checklists is optional- Potential defect logs and review reports are typically produced

Inspection

Main purpose	Detecting the maximum nr of anomalies, evaluating quality and building confidence in the work product, preventing future similar defects through author learning and root cause analysis
Possible additional purposes	Motivating and enabling authors to improve future work products and the software development process, achieving consensus
Characteristics	<ul style="list-style-type: none">- Follows a defined process with formal documented outputs, based on rules and checklists- Uses clearly defined roles (mandatory) and may include a dedicated reader- Individual preparation before the review meeting is required- Reviewers are peers of the author or experts in other relevant disciplines- Specified entry and exit criteria are used- Scribe is mandatory- Review meeting is led by a trained facilitator (not the author)- Author cannot act as review leader, reader or scribe- Potential defect logs and review reports are produced- Metrics are collected and used to improve the entire software development process



Success Factors for Reviews

Success factors



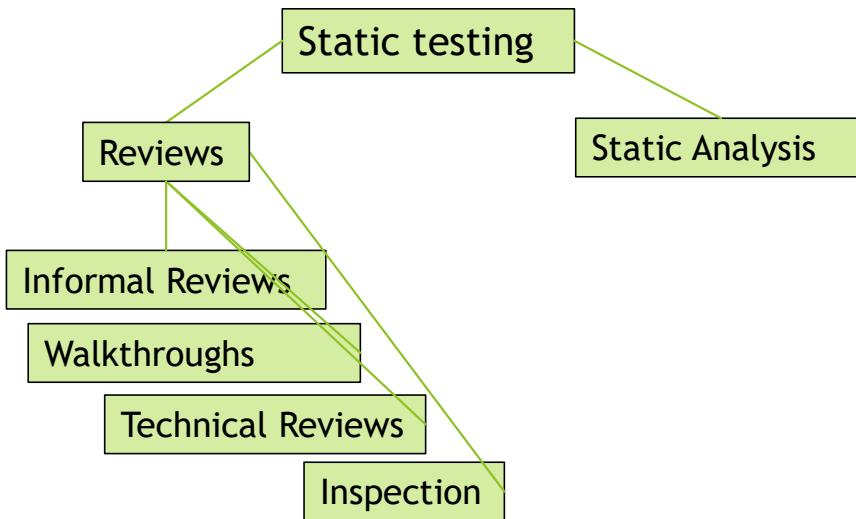
- ▶ Clear, predefined objectives
- ▶ Measurable exit criteria
- ▶ The review is conducted in an atmosphere of trust; the outcome will not be used for the evaluation of the participants
- ▶ Review types are applied that are suitable to achieve the objectives
- ▶ Large work products are written and reviewed in small chunks
- ▶ Feedback is given to the participants so the process and their activities can be improved
- ▶ Participants have adequate time to prepare for the review
- ▶ Reviews are made part of the organizational culture to promote learning and improvement
- ▶ Management supports the review process
- ▶ Adequate training is given; especially for the more formal review types
- ▶ Facilitate meetings



Static Testing

Summary - Keywords Explained

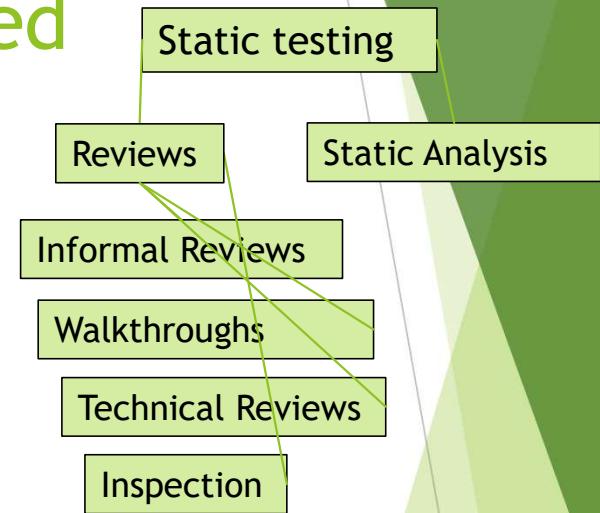
Static Testing



- ▶ **Formal Review** - A type of review that follows a defined process with a formally documented output.
- ▶ **Informal Review** - A type of review that does not follow a defined process and has no formally documented output.

Static Testing - keywords explained

- ▶ **Static testing:** Testing that does not involve the execution of a test item.
- ▶ **Static analysis:** the process of evaluating a component or system without executing it, based on its form, structure, content, or documentation.
- ▶ **Review:** A type of static testing in which a work product or process is evaluated by one or more individuals to detect defects or to provide improvements.
- ▶ **Informal Review** - A type of review that does not follow a defined process and has no formally documented output.
- ▶ **Walkthrough:** A type of review in which an author leads members of the review through a work product and the members ask questions and make comments about possible issues.
- ▶ **Technical review:** A formal review by technical experts that examine the quality of a work product and identify discrepancies from specifications and standards.
- ▶ An **Inspection** is a type of formal review that uses defined team roles and measurement to identify defects in a work product and improve the review process and the software development process.



Static and Dynamic Testing

Static testing

Static testing: Testing that does not involve the execution of a test item.

Dynamic Testing

- ▶ **Dynamic testing:** Testing that involves the execution of the test item.

- ▶ **Anomaly:** A condition that deviates from expectation.



Test Analysis and Design

Introduction

Test Analysis and Design

4.1 Test Techniques Overview

- ▶ FL-4.1.1 (K2) Distinguish black-box, white-box and experience-based test techniques

4.2 Black-box Test Techniques

- ▶ FL-4.2.1 (K3) Use equivalence partitioning to derive test cases
- ▶ FL-4.2.2 (K3) Use boundary value analysis to derive test cases
- ▶ FL-4.2.3 (K3) Use decision table testing to derive test cases
- ▶ FL-4.2.4 (K3) Use state transition testing to derive test cases

4.3 White-box Test Techniques

- ▶ FL-4.3.1 (K2) Explain statement testing
- ▶ FL-4.3.2 (K2) Explain branch testing
- ▶ FL-4.3.3 (K2) Explain the value of white-box testing

Test Analysis and Design

4.4 Experience-based Test Techniques

- ▶ FL-4.4.1 (K2) Explain error guessing
- ▶ FL-4.4.2 (K2) Explain exploratory testing
- ▶ FL-4.4.3 (K2) Explain checklist-based testing

4.5. Collaboration-based Test Approaches

- ▶ FL-4.5.1 (K2) Explain how to write user stories in collaboration with developers and business representatives
- ▶ FL-4.5.2 (K2) Classify the different options for writing acceptance criteria
- ▶ FL-4.5.3 (K3) Use acceptance test-driven development (ATDD) to derive test cases



Test Techniques

Overview

Test Techniques - General

- ▶ Test techniques are used in test analysis ('what to test') and test design ('how to test')
- ▶ Test techniques help to create a small, but sufficient number of test cases in a systematic way
- ▶ Use of test techniques can vary from very formal to very informal
- ▶ Formality depends on context, test maturity and development maturity, time constraints, type of SDLC, safety and regulatory requirements, knowledge and skills of the people involved
- ▶ Test techniques help the tester in:
 - ▶ Defining **Test Conditions**
 - ▶ Identifying **Coverage items**
 - ▶ Identifying **Test Data**



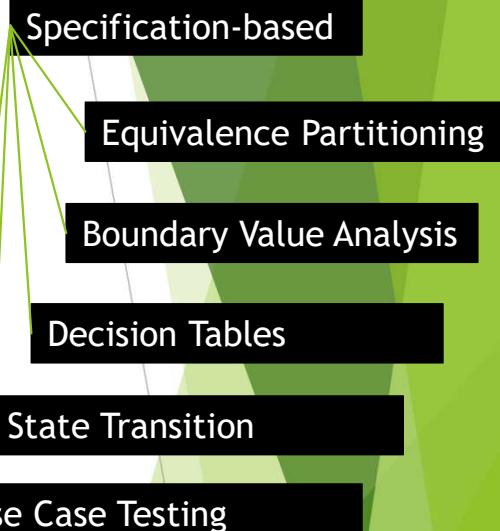
Choosing Test Techniques

- ▶ Some techniques are more applicable to certain test levels and situations, others can be used in all test levels
- ▶ In general, a combination of test techniques can achieve the best result
- ▶ A lot of factors can determine which test technique to choose, for example:

Type of component or system	Tester knowledge and skills
Complexity of the component or system	Available tools, time and budget
Regulatory standards	Software development lifecycle model
Customer or contractual requirements	Expected use of the software
Risk levels and risk types	Previous experience
Test objectives	Types of defects expected
Available documentation	

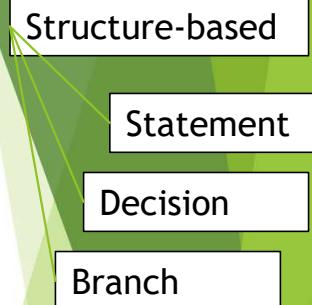
Black-box Test Techniques

- ▶ Aka specification-based techniques
- ▶ Based on an analysis of the specified behavior od the test object
- ▶ View the test object (software under test) as black box with inputs/outputs
without looking at how the system or component is structured inside the box
- ▶ Does NOT take the internal structure into account
- ▶ Test cases are independent of the way the software is implemented
- ▶ If the implementation changes, the test cases are still valid
- ▶ Functional and non-functional



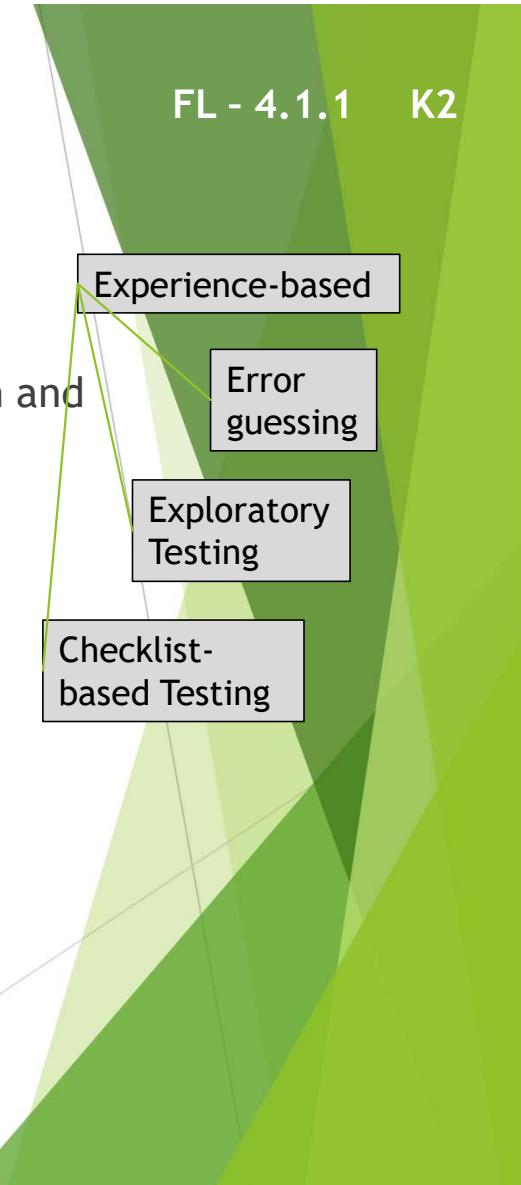
White-box Test Techniques

- ▶ Aka structure-based techniques
- ▶ Based on an analysis of the architecture, detailed design, internal structure, code and processing **within** the test object
- ▶ Test cases are dependent on the **internal** structure and processing within the test object
- ▶ Software design is the basis for test cases, so test cases can not be created before the design or implementation of the test object

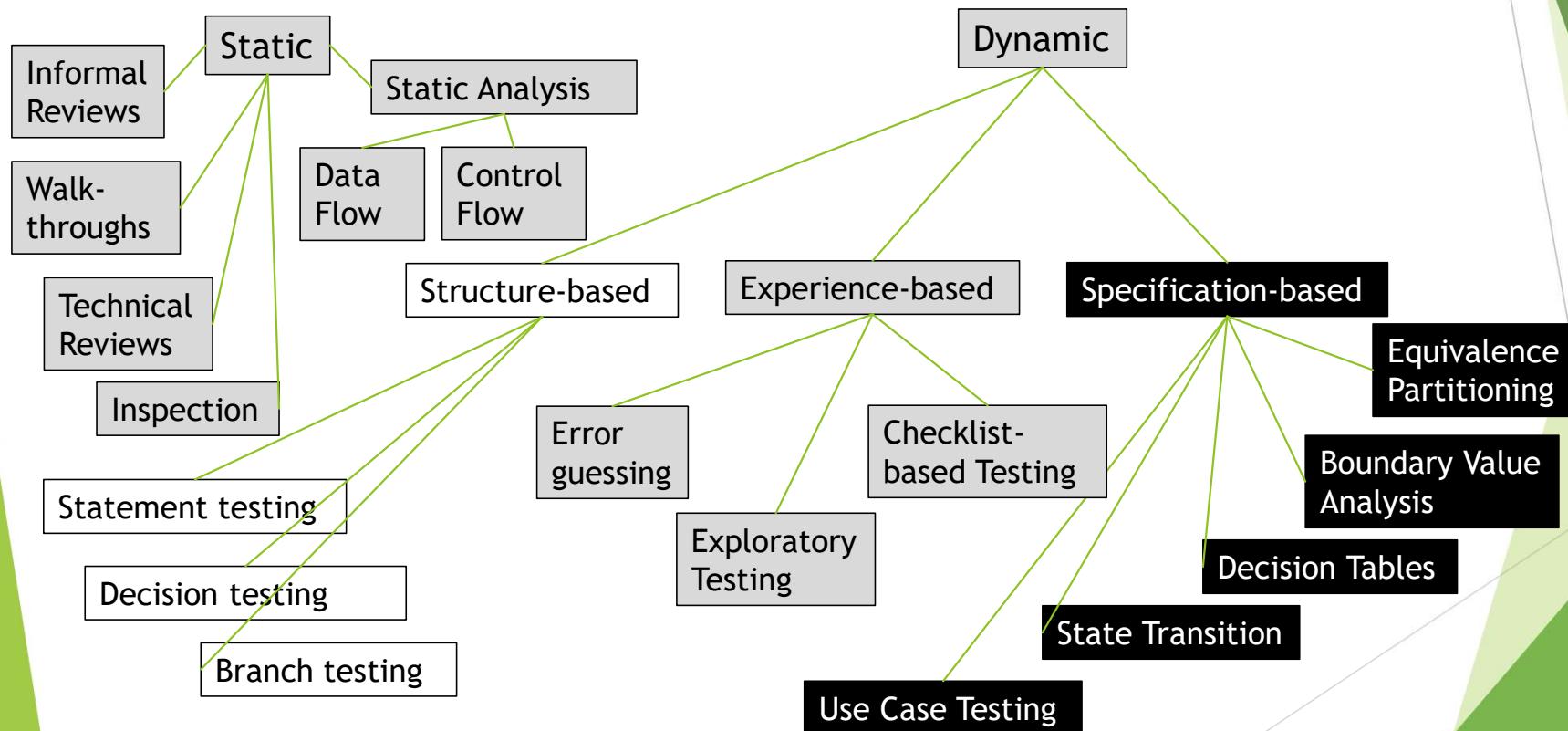


Experience-based Test Techniques

- ▶ Leverage the experience, knowledge and intuition of the testers to design and implement tests
- ▶ Effectiveness heavily depends on the skills and experience of the tester
- ▶ Experience-based testing can find defects that could be missed by more formal test techniques (black-box and white-box)
- ▶ Often complement white/box and black/box test techniques



Testing Techniques - visualized





Black-box Test Techniques

Introduction

Black-box Test Techniques - Characteristics

- ▶ Test conditions, test cases and test data are derived from a test basis that may include
 - ▶ Software requirements
 - ▶ Specifications
 - ▶ Use cases
 - ▶ User stories
- ▶ Test cases may be used to detect gaps between requirements and the implementation of the requirements as well as deviations from the requirements
- ▶ Coverage is measured based on the items tested in the test basis and the technique applied to the test basis

Specification-based

Equivalence Partitioning

Boundary Value Analysis

Decision Tables

State Transition

Use Case Testing



Black-box Test Techniques

Equivalence Partitioning

Equivalence Partitioning

- ▶ Can be applied on all testing levels
- ▶ Idea is to divide (partition) data into groups or sets for which the behavior is assumed to be the same (test object should handle each value in a partition **equivalently**)
- ▶ Equivalent partitions are also known as equivalence classes
- ▶ Test cases designed to execute representatives from **ALL** equivalence partitions
 - ▶ Valid values are values that should be accepted by the component or system and are in a **valid equivalence partition**
 - ▶ Invalid values are values that should **NOT** be accepted by the component or system and are in an **invalid equivalence partition**
- ▶ Test cases are designed to cover each partition once and **ONLY** once (valid and invalid)

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

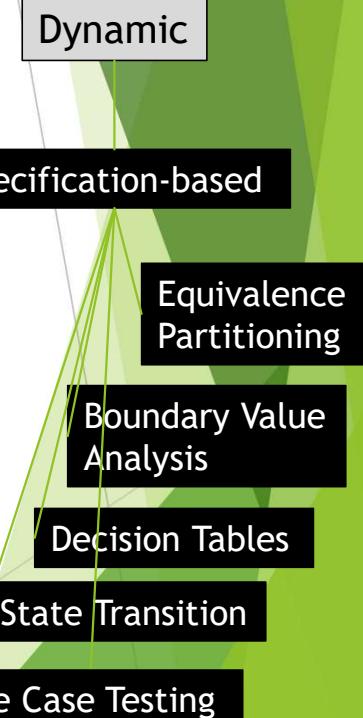
State Transition

Use Case Testing

Equivalence Partitioning (2)

- ▶ If one condition in a partition works (or does not work) we assume they all work (or do not work) - so no need to test more conditions
- ▶ Invalid equivalence partitions should be tested individually
- ▶ Each partition can be divided into sub partitions if needed
- ▶ Partitions can be identified for **any** data element related to the test object
- ▶ Coverage is calculated as

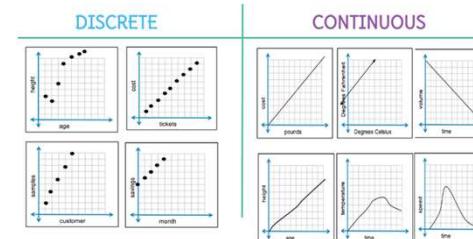
$$\frac{\text{Nr of equivalence partitions tested by at least one value}}{\text{Total nr of identified equivalence partitions}} * 100 \%$$



Points of Attention in Equivalence Partitioning

- ▶ Partitions must NOT overlap and must NOT be empty
- ▶ Partitions do NOT have to be ordered or bordered; they can be:
 - ▶ Continuous or discrete
 - ▶ Ordered or not ordered
 - ▶ Finite or infinite
- ▶ In simple test objects, the test technique is not hard
- ▶ In practice, there are often difficulties understanding how the test object will treat different values
- ▶ A very powerful test technique, if done properly and with care

Invalid	Valid	Valid	Invalid
<=20	18-50	30-70	>70



Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

Valid or Invalid Partitions?

- ▶ It seems simple:
 - ▶ A partition containing VALID values is a VALID partition
 - ▶ A partition containing INVALID values is an INVALID partition

BUT

- ▶ Companies and teams have different definitions of valid and invalid:
 - ▶ Valid values could be:
 - ▶ Values that should be processed by the test object
 - ▶ Values for which the processing is defined in the specification
 - ▶ Invalid values could be:
 - ▶ Values that should be ignored or rejected by the test object (as specified)
 - ▶ Values for which the processing is not defined in the specification



Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

Invalid Equivalence Partitions - example

Scenario:

For every child parents get a certain amount of money from the government for the support of the child.

- ▶ If no children, parents get nothing
- ▶ For 1-2 children parents get 50 EUR per child
- ▶ For 3-5 children parents get 75 EUR per child
- ▶ For 6-8 children parents get 100 EUR per child

0	1-2	3-5	6-8	> 8
0	50	75	100	-

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

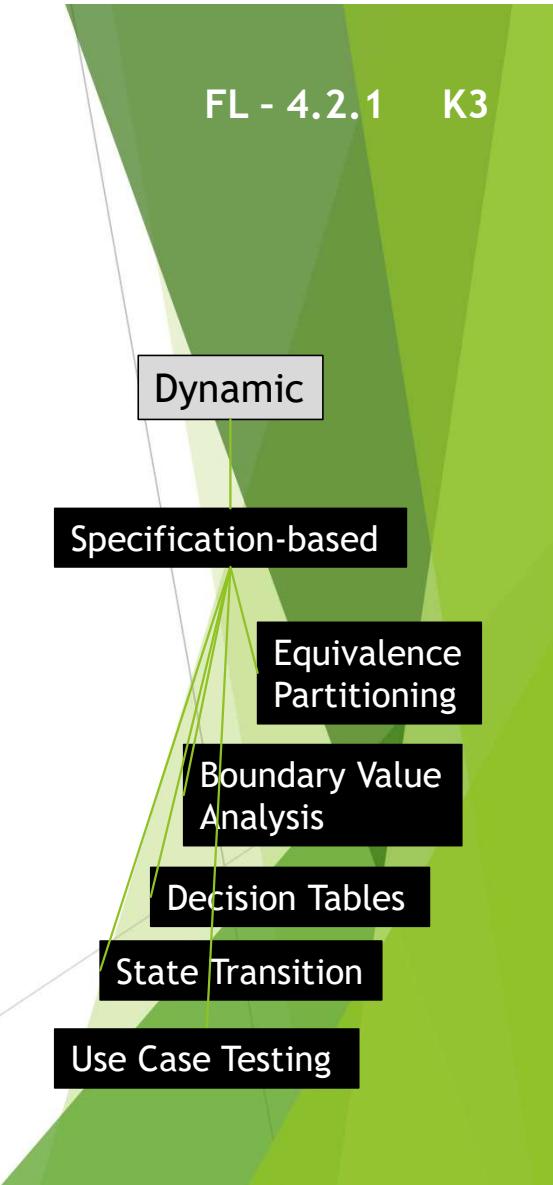
Decision Tables

State Transition

Use Case Testing

Multiple sets of Partitions

- ▶ Multiple sets of partitions are common in test objects
- ▶ A test case will then cover multiple partitions from different EP tables
- ▶ How then to measure coverage?
- ▶ Easiest method is ‘Each Choice coverage’
 - ▶ Test cases should exercise EACH partition from EACH set at least one
 - ▶ Combinations of partitions are not taken into account



Equivalence Partitioning - Example

- ▶ Savings Account in a bank - earns different interest percentages - depending on balance
 - ▶ Balance between 0 and 50 - 2 % interest
 - ▶ Balance more than 50 but less than 1000 - 3 % interest
 - ▶ Balance of 1000 and more - 5 % interest
- ▶ Result is 3 valid partitions and 1 invalid one - invalid one was not in the spec, but needs to be tested, so we have 4 tests

Invalid	Valid (2%)	Valid (3%)	Valid (5 %)
< 0.00	0.00 - 50.00	50.01 - 999.99	> 999.99

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

Equivalence Partitioning - Example (2)

- ▶ Equivalence partitioning can also be applied to the output values (%)
- ▶ Example of proper input values are -10.00, 25.00, 999.98 and 1456.55
- ▶ ‘Invalid’ partition - not one of the expected inputs for that field
- ▶ Don’t only do what is specified, but also think about things that haven’t been specified
- ▶ What if testing this scenario without applying EP?

Invalid	Valid (2%)	Valid (3%)	Valid (5 %)
< 0.00	0.00 - 50.00	50.01 - 999.99	> 999.99

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

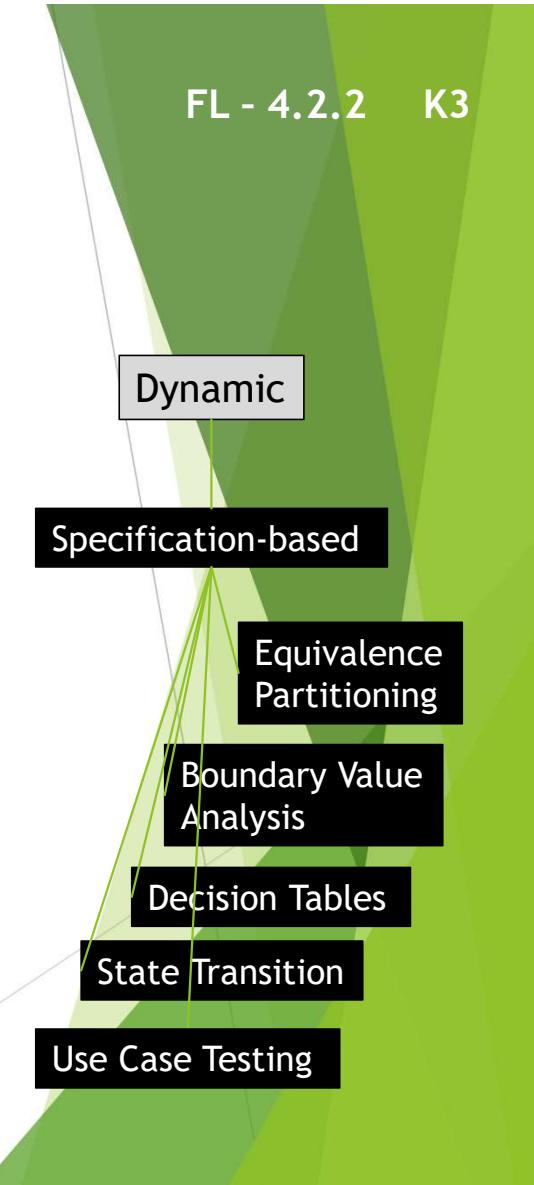


Black-box Test Techniques

Boundary Value Analysis

Boundary Value Analysis explained

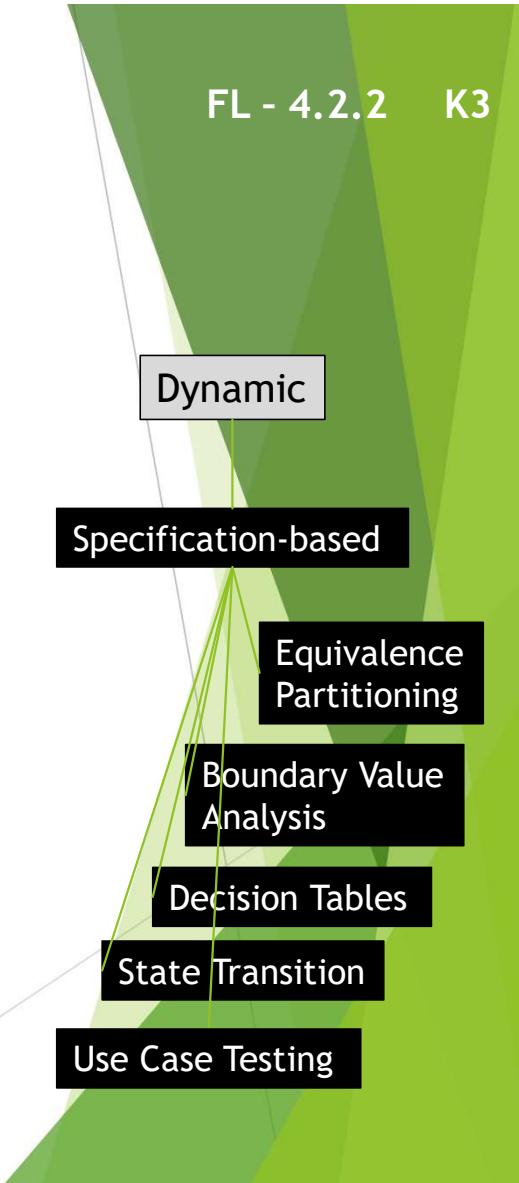
- ▶ Can be applied at all test levels
- ▶ Extension of EP, generally used to test requirements that call for a range of numbers
- ▶ Can ONLY be used when the partitions are ordered
- ▶ Exercising the boundaries of equivalence partitions - min and max values of a partition
- ▶ Behavior at the boundaries of equivalence partitions are **more likely** to be incorrect than within the partitions



Boundary Value Analysis and Coverage

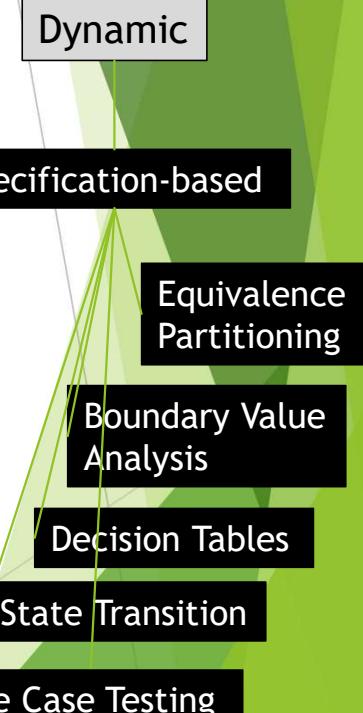
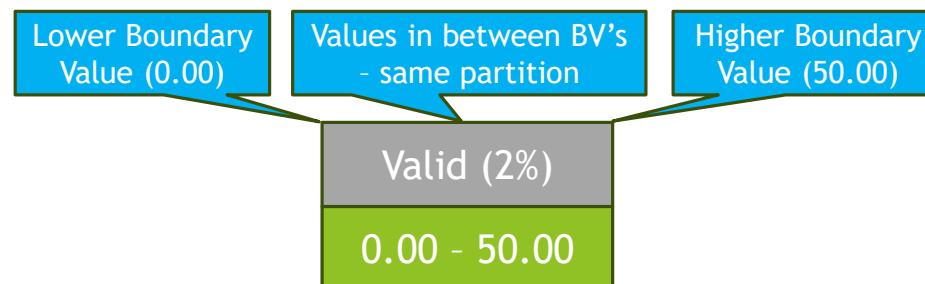
- ▶ 2 elements (values) belonging to the same partition - all values in between must also belong to that partition
- ▶ Boundary values are the MIN AND MAX values of Equivalence Partitions
- ▶ Typically, locations of defects found by BVA at misplaced boundaries
- ▶ Coverage is calculated as:

$$\frac{\text{Nr of coverage items tested}}{\text{Total nr of identified coverage items}} * 100 \%$$



Identifying Boundary Values

- ▶ Divide a set of test conditions into groups or sets for which the behavior is assumed to be the same
- ▶ Draw up your Equivalence Partition table
- ▶ Identify the minimum and maximum values of each partition
- ▶ These are your Boundary Values
- ▶ Possible to only have 1 Boundary Value in a partition



Boundary Value Analysis visualized

- ▶ Savings Account in a bank - earns different interest percentages - depending on balance
 - ▶ Balance between 0 and 50 - 2 % interest
 - ▶ Balance more than 50 but less than 1000 - 3 % interest
 - ▶ Balance of 1000 and more - 5 % interest
- ▶ Result is 3 valid partitions and 1 invalid one
- ▶ Let's identify our Boundary Values



Dynamic

Specification-based

Equivalence
Partitioning

Boundary
Value
Analysis

Decision Tables

State Transition

Use Case Testing



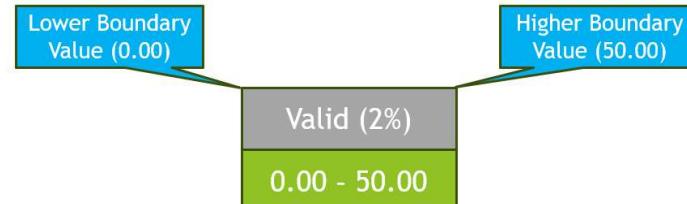
Black-box Test Techniques

2-Value Boundary Value Analysis

Boundary Value Analysis (2-value) - coverage

- ▶ Each Boundary Value has 2 coverage items
 - ▶ Boundary Value AND
 - ▶ The value of the closest neighbor belonging to the adjacent partition
- ▶ 100% 2-value BVA coverage - cover ALL coverage items (Boundary Values)
- ▶ Coverage is calculated as:

$$\frac{\text{Nr of boundary values tested}}{\text{Total nr of identified boundary values}} * 100 \%$$



Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

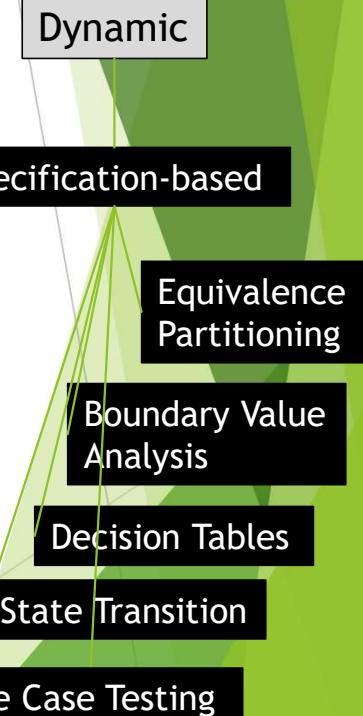
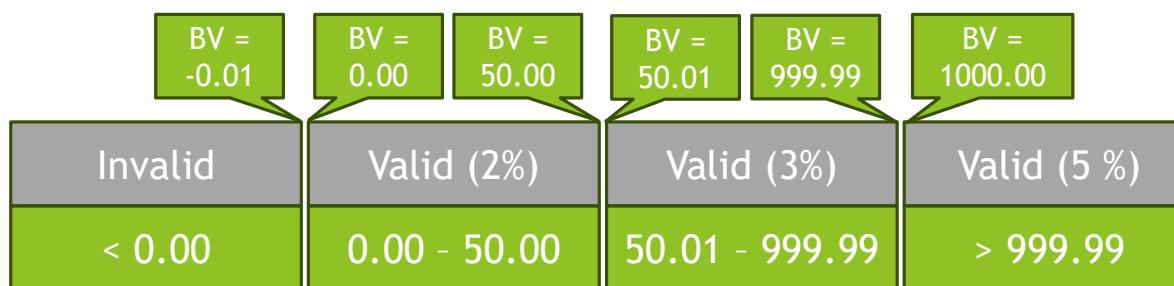
Decision Tables

State Transition

Use Case Testing

Boundary Value Analysis (2-value) applied

- ▶ Focused on testing at the **minimum** and **maximum** values of each partition
 - ▶ The min and max (boundary) values of each partition, and
 - ▶ The value of the closest neighbor belonging to the adjacent partition
- ▶ Proper input values on previous example are -0.01, 0.00, 50.00, 50.01, 999.99 and 1000.00





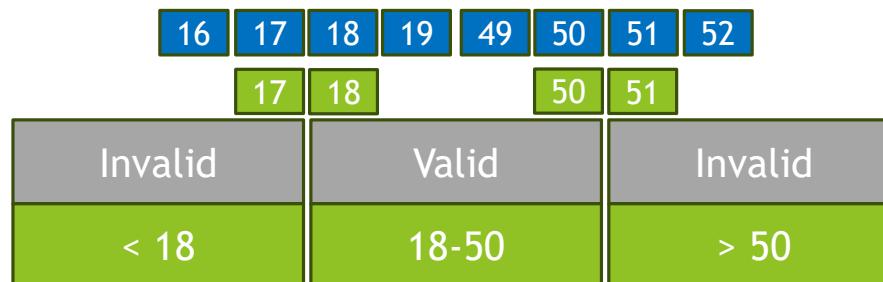
Black-box Test Techniques

3-Value Boundary Value Analysis

Boundary Value Analysis (3-value) - coverage

- ▶ Also called ‘Full Boundary Value Analysis’
- ▶ Each Boundary Value has 3 coverage items
 - ▶ Boundary Value AND
 - ▶ The values of BOTH its’ neighbors
- ▶ Part of the coverage items are NOT Boundary Values
- ▶ 100 % 3-value BVA coverage - cover ALL coverage items (Boundary Values and their neighbors)
- ▶ Coverage is calculated as:

$$\frac{\text{Nr of boundary values and their neighbors exercised}}{\text{Total nr of identified boundary values and their neighbors}} * 100 \%$$



Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

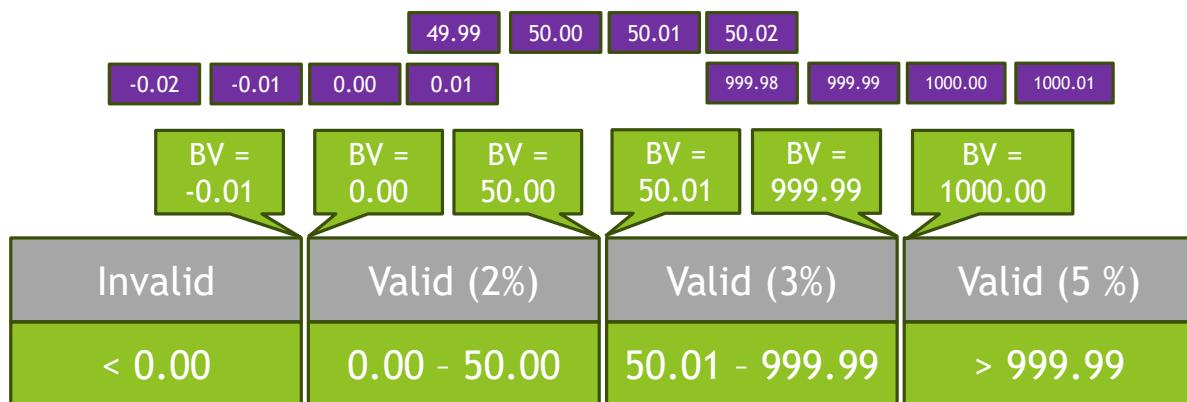
Decision Tables

State Transition

Use Case Testing

Boundary Value Analysis (3-value) - applied

- ▶ Focused on testing at the **boundaries** using the Boundary Values **PLUS** both one value on either side of each boundary (as close as you can get)
- ▶ Idea is to take:
 - ▶ The min and max (boundary) values of each partition, and
 - ▶ One value on each side of the boundary value as close to the boundary value as possible



Dynamic

Specification-based

Equivalence Partitioning

Boundary Value Analysis

Decision Tables

State Transition

Use Case Testing



Black-box Test Techniques

Equivalence Partitioning
and Boundary Value
Analysis combined

EP and BVA combined

- ▶ After having determined the test conditions (here with EP and BVA), the test cases can be designed
- ▶ Try to cover more test conditions with a single test case
- ▶ A balance should be found between having too many and too few test cases
- ▶ Test partitions and boundaries separately
- ▶ Which partitions and boundaries to exercise - depends on the goal and the time pressure
- ▶ Can be implemented on different kinds of input, etc

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing



Black-box Test Techniques

Decision Table Testing

Decision Table Testing

- ▶ Combinatorial test technique that can be applied at all test levels
- ▶ Useful to test implementation of system requirements that specify how different combinations of conditions resulting in different outcomes
- ▶ Effective way to record complex logic such as business rules
- ▶ Helps identify all the important combinations of conditions
- ▶ Technique is to:
 - ▶ Identify all inputs (conditions) and outputs (actions) and put them in a table as rows
 - ▶ List all True-False combinations for each of the conditions (**mostly Boolean**)
 - ▶ Identify the correct outcome for each combination - each column is a decision rule
 - ▶ Write tests to exercise each of the rules (columns) in the decision table at least once

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

Decision Table Testing (2)

- ▶ Systematic way to identify all combinations of conditions
- ▶ Find any gaps or contradictions in the requirements
- ▶ Exercising ALL decision rules may be very time consuming if there are a lot of conditions
- ▶ If a lot of combinations, or having multiple types of input that are not binary, use risk assessment or a minimized decision table to decide which rules to execute
- ▶ Table can be collapsed by deleting business rules (columns) that are not possible
- ▶ Normal notation is:
 - ▶ Conditions: T if true, F if not true (false), - if the condition is irrelevant, N/A indicates an infeasible condition for a given rule column
 - ▶ Actions: X if action should occur (or a value, color, etc), blank if the action should not occur (or -)

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

Decision Table Testing - Coverage

- ▶ Decision tables can be simplified and made smaller by:
 - ▶ Deleting columns with infeasible combinations of conditions
 - ▶ Merging columns to minimize the table (ONLY if some conditions do not affect the outcome)
- ▶ Coverage items are feasible combinations of conditions (columns)
- ▶ Each column should be exercised to get 100% Decision Table coverage
- ▶ Coverage is calculated as

$$\frac{\text{Nr of exercised columns}}{\text{Total nr of feasible columns}} * 100 \%$$

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

Decision table testing - Example

- ▶ Opening a credit card account
- ▶ New customer, 15% discount on purchases today
- ▶ Existing customer and loyalty card, 10% discount on purchases today
- ▶ If you have a coupon, 20% discount on purchases today (not in combination with the new customer discount)
- ▶ Discounts are added if applicable or if combination is invalid the highest discount is given

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

Decision table testing - Example (2)

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
New customer								
Loyalty card								
Coupon								
Actions								
Discount %								

Opening a credit card account

New customer, 15% discount on purchases today

Existing customer and loyalty card, 10% discount on purchases today

If you have a coupon, 20% discount on purchases today (not in combination with the new customer discount)

Discounts are added if applicable or if combination is invalid the highest discount is given

Let's try and draw up a filled in decision table

Decision table testing - Example (3)

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
New customer	T	T	T	T	F	F	F	F
Loyalty card	T	T	F	F	T	T	F	F
Coupon	T	F	T	F	T	F	T	F
Actions								
Discount %			20	15	30	10	20	0

Opening a credit card account

New customer, 15% discount on purchases today

Existing customer and loyalty card, 10% discount on purchases today

If you have a coupon, 20% discount on purchases today (not in combination with the new customer discount)

Discounts are added if applicable or if combination is invalid the highest discount is given

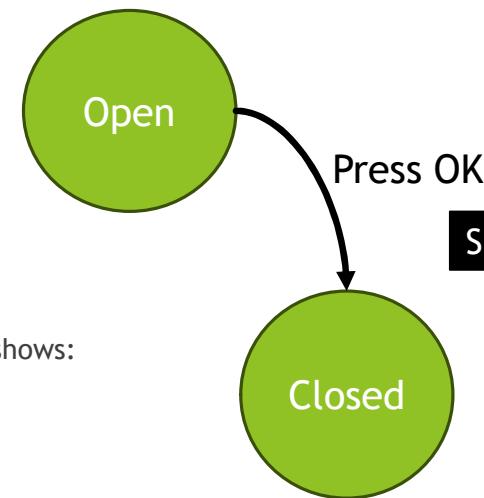


Black-box Test Techniques

State Transition Testing

State transition testing

- ▶ Used when an aspect of the system is in a ‘finite state machine’
 - ▶ Limited number of states
 - ▶ State is determined by the rules of the machine
 - ▶ Different response on an event depending on the current conditions and previous history
- ▶ Test cases can be designed to execute valid and invalid state transitions
- ▶ Behavior in a system is shown in a **state transition diagram model**, which shows:
 - ▶ Possible states and valid transitions
 - ▶ Events that cause or result in a transition from one state to another
 - ▶ Actions that result from a transition
- ▶ If one event from the same state can result in 2 or more different transitions, the event may be qualified by a **guard condition** (precondition for an event), standard syntax is ‘event [guard condition] / action’
- ▶ Transitions are assumed to be instantaneous
- ▶ Transitions may result in the software taking action
- ▶ An event can cause a different action from a different state



Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

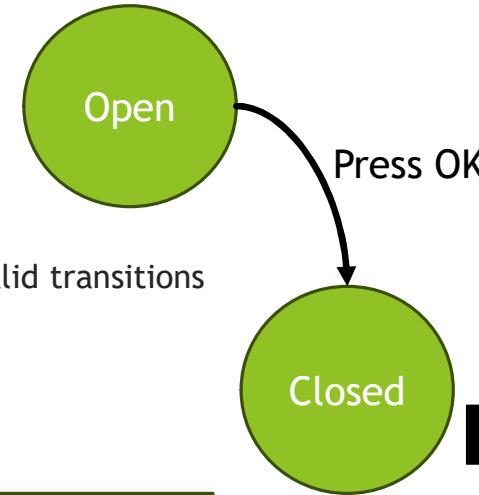
State Transition

Use Case Testing

State transition testing (2)

- ▶ State transition tables are often used to identify valid and invalid transitions
- ▶ Tests can be designed to:
 - ▶ Cover a typical sequence of states
 - ▶ Exercise specific sequences of transitions
 - ▶ Exercise all states
 - ▶ Exercise all valid transitions
 - ▶ Exercise all transitions
 - ▶ Test invalid transitions
- ▶ Rows represent states, columns represent events (with or without guard conditions)
- ▶ Cell entries represent transitions and contain the target state and possible actions
- ▶ If a cell is empty, that is an invalid transition

	Event
State	Press OK
Open	Closed
Closed	-



FL - 4.2.4 K3

Dynamic

Specification-based

Equivalence Partitioning

Boundary Value Analysis

Decision Tables

State Transition

Use Case Testing

State transition testing (3)

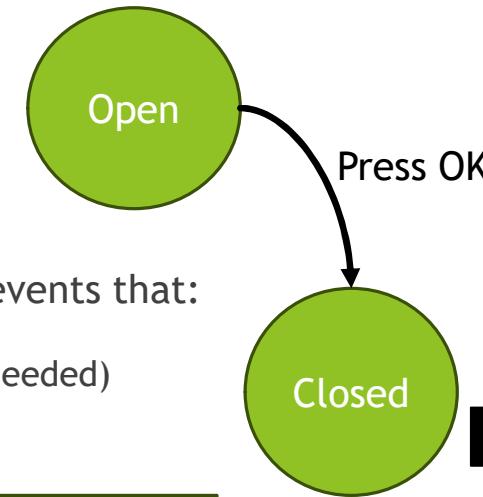
- ▶ Test cases are usually represented as a sequence of events that:
 - ▶ Results in a sequence of state changes (and actions if needed)
 - ▶ Usually cover several transitions between states

TC - Open, Press OK, Closed

OR

TC - Open, Closed

	Event
State	Press OK
Open	Closed
Closed	-



FL - 4.2.4 K3

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

State transition testing - Coverage

3 versions of coverage criteria:

- ▶ All states coverage
 - ▶ Coverage item are the states
 - ▶ 100% all states coverage - all states should be visited
 - ▶ Nr of visited states / total nr of states * 100%
- ▶ Valid Transitions Coverage (aka 0-switch coverage)
 - ▶ Coverage items are single valid transitions
 - ▶ 100% valid transition coverage - exercise all valid transitions
 - ▶ Nr of exercised valid transitions / total nr of valid transitions * 100%

	Event
State	Press OK
Open	Closed
Closed	-

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

Decision Tables

State Transition

Use Case Testing

State transition testing - Coverage

- ▶ All Transitions Coverage
 - ▶ Coverage items are ALL transitions (valid and invalid)
 - ▶ 100% all transition coverage:
 - ▶ Exercise all valid transitions
 - ▶ Try to exercise all invalid transitions
 - ▶ Test only one invalid transition per test case:
 - ▶ To avoid fault masking
 - ▶ Nr of exercised valid and invalid transitions / total nr of valid and invalid transitions * 100%
 - ▶ Often a minimum requirement for mission and safety-critical software
- ▶ 100% All Transitions Coverage guarantees 100% Valid Transitions coverage AND 100% All States Coverage
- ▶ 100% Valid Transitions coverage guarantees 100% All States Coverage
- ▶ Full Valid Transitions Coverage is most widely used
- ▶ All states coverage is weakest as typically it does not cover all transitions

	Event
State	Press OK
Open	Closed
Closed	-

Dynamic

Specification-based

Equivalence
Partitioning

Boundary Value
Analysis

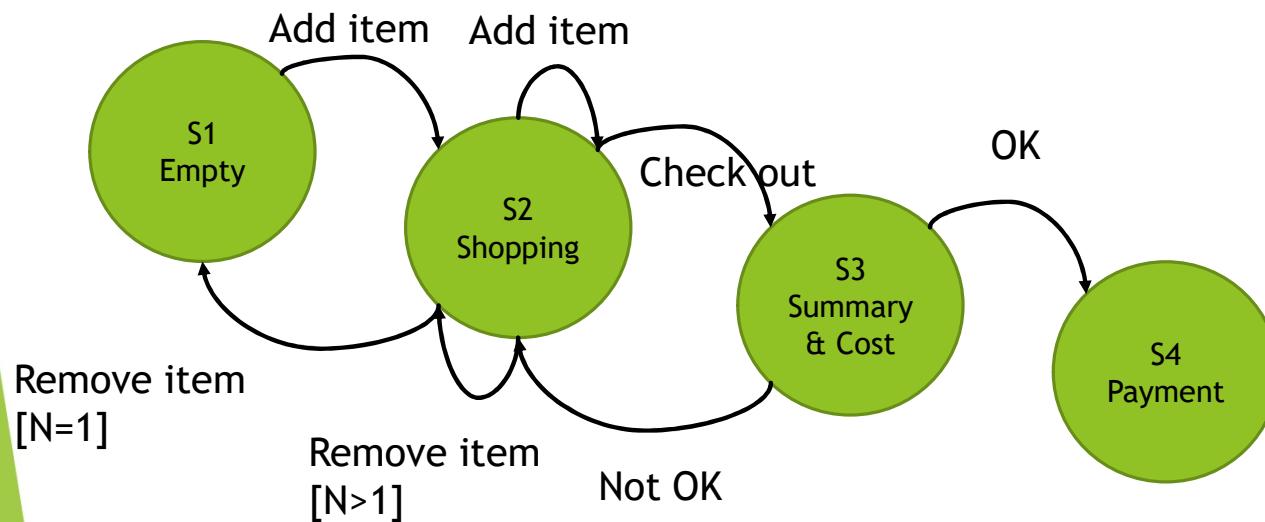
Decision Tables

State Transition

Use Case Testing

State transition testing - Example

- ▶ Website shopping cart starts empty - purchases are selected and added to the cart. Items can also be removed from the cart. On checkout, a summary of basket and cost is shown and customer indicates whether this is OK or not. If all is ok, then leave the summary display and go to the payment system. Otherwise go back to shopping.



Dynamic

Specification-based

Equivalence Partitioning

Boundary Value Analysis

Decision Tables

State Transition

Use Case Testing

State transition testing - Example (2)

State	Event					
	Add Item	Remove Item [N>1]	Remove Item [N=1]	Check out	Not OK	OK
S1 Empty	S2	-	-	-	-	-
S2 Shopping	S2	S2	S1	S3	-	-
S3 Summary	-	-	-	-	S2	S4
S4 Payment	-	-	-	-	-	-

All of the boxes that contain '-' are invalid. So examples of invalid transitions are:

- Try to add an item from the summary & cost state
- Try to remove an item from an empty shopping cart
- Try to enter OK while in the shopping state or when having an empty basket

Dynamic

Specification-based

Equivalence Partitioning

Boundary Value Analysis

Decision Tables

State Transition

Use Case Testing

State transition testing - Example (3)

- ▶ A test to cover all valid transitions would be:

State	Event
S1	Add item
S2	Remove Item [N=1]
S1	Add item
S2	Add item
S2	Remove Item [N>1]
S2	Check out
S3	Not OK
S2	Check out
S3	OK
S4	

State	Event					
	Add Item	Remove Item [N>1]	Remove Item [N=1]	Check out	Not OK	OK
S1 Empty	S2	-	-	-	-	-
S2 Shopping	S2	S2	S1	S3	-	-
S3 Summary	-	-	-	-	S2	S4
S4 Payment	-	-	-	-	-	-

All states coverage = $4 / 4 * 100 = 100 \%$

Valid transitions coverage = $7 / 7 * 100 = 100 \%$

All transitions coverage = $7 / 24 * 100 = 29.17 \%$

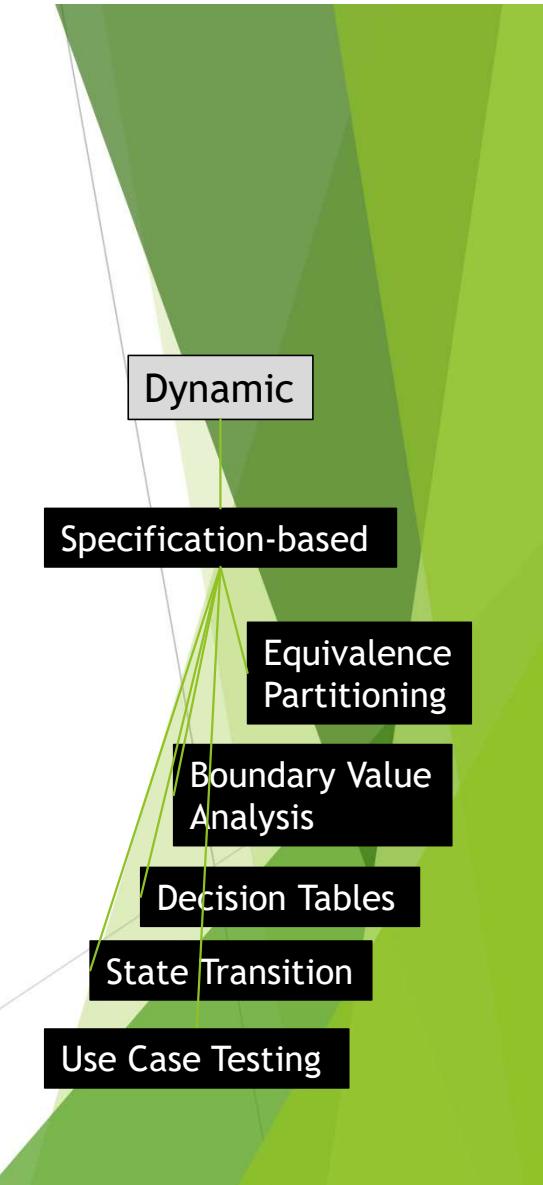


Black-box Test Techniques

Use Case Testing

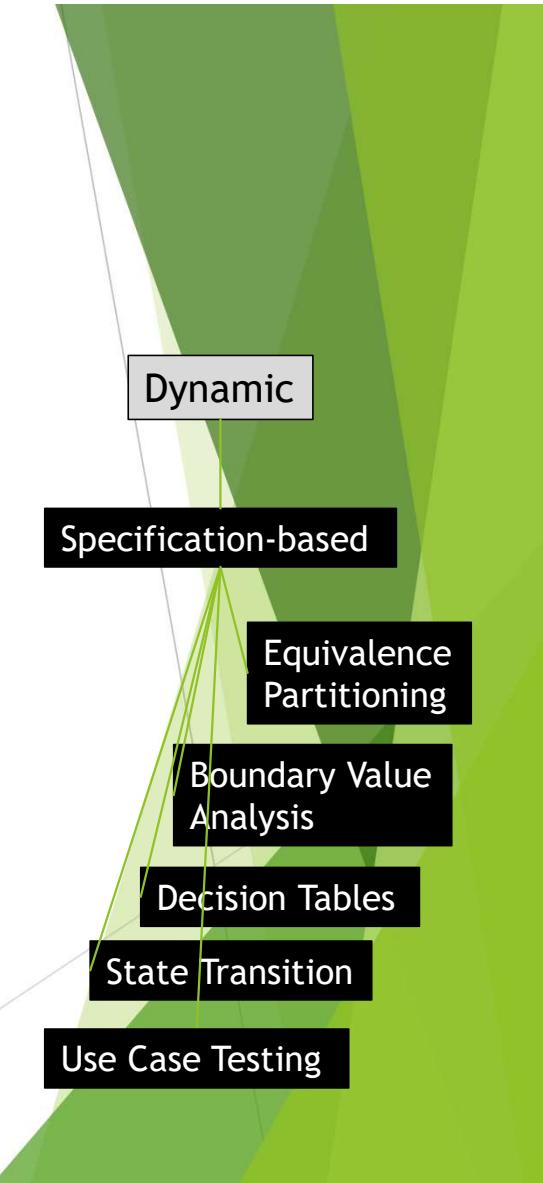
Use Case Testing

- ▶ Identify test cases that exercise the whole system on transaction by transaction basis from start to finish (scenarios of use cases)
- ▶ Use case - a description of the use of the system by an actor (user, external hardware, other components or systems)
- ▶ Use cases are associated with actors and subjects (component or system to which the use case is applied)
- ▶ Each use case specifies some behavior that a subject can perform with one or more actors
- ▶ Use case can be described by:
 - ▶ Interactions
 - ▶ Activities
 - ▶ Preconditions
 - ▶ Postconditions
- ▶ Interactions between actors and the subject may result in changes to the state of the subject



Use Case Testing

- ▶ Interactions can be represented using workflows, activity diagrams, business process models
- ▶ Mostly at the system and acceptance levels
- ▶ As most likely user is used, good at finding defects in the real world use of the system
- ▶ Tests are designed to exercise the defined behaviors:
 - ▶ Basic
 - ▶ Exceptional
 - ▶ Alternative
 - ▶ Error handling
- ▶ Coverage is calculated as
$$\frac{\text{Nr of use case behaviors tested}}{\text{Total nr of use case behaviors}} * 100 \%$$



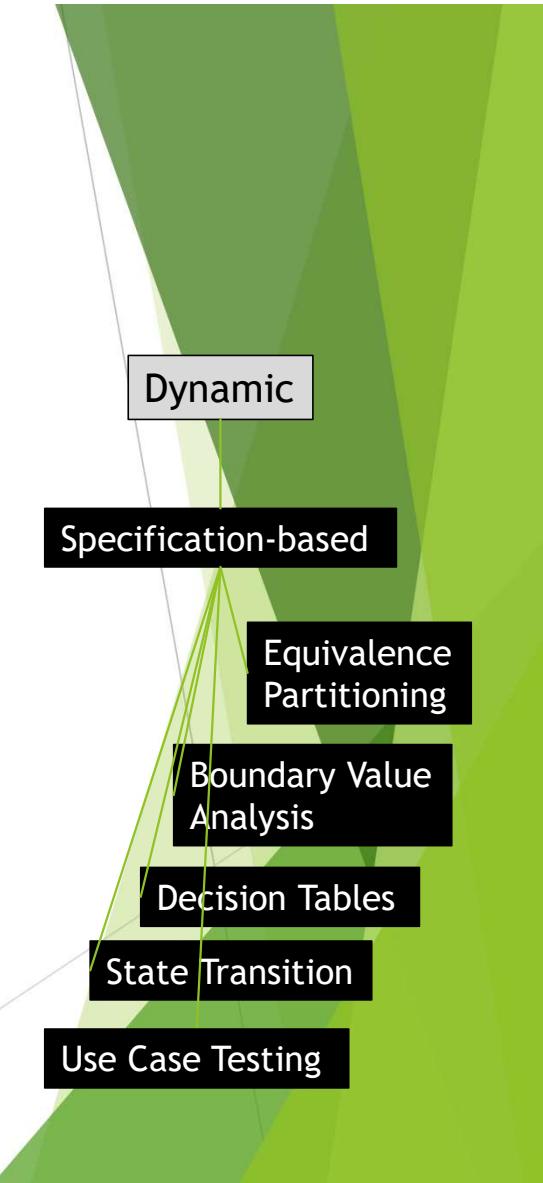
Use Case Testing - Example

- ▶ A little extension from our ATM example we used in the State Transition testing

Use Case:

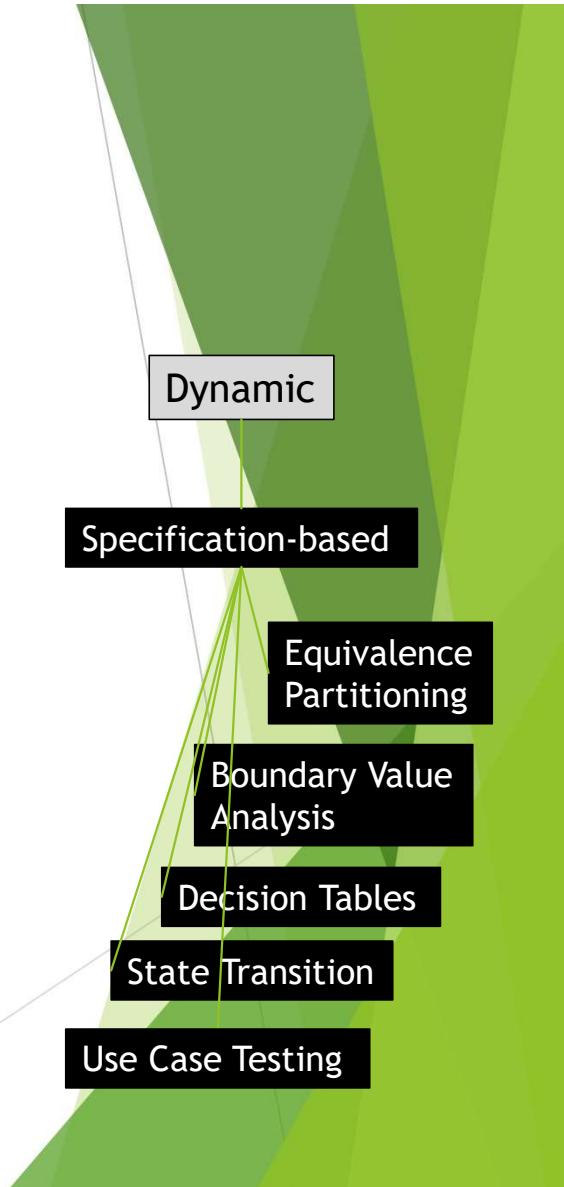
User Inserts the card, system validates the card and asks for the PIN.

Actor enters PIN and System validates the PIN. If valid, account access is granted. If the card is not valid, a message is shown and the card is rejected. After 3 incorrect PIN attempts, the card is eaten.



Use Case Testing - Example

	Step	Description
Main Success Scenario A: Actor S: System	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows account access
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for retry (twice)
	4b	PIN invalid 3 times S: Eat card and exit

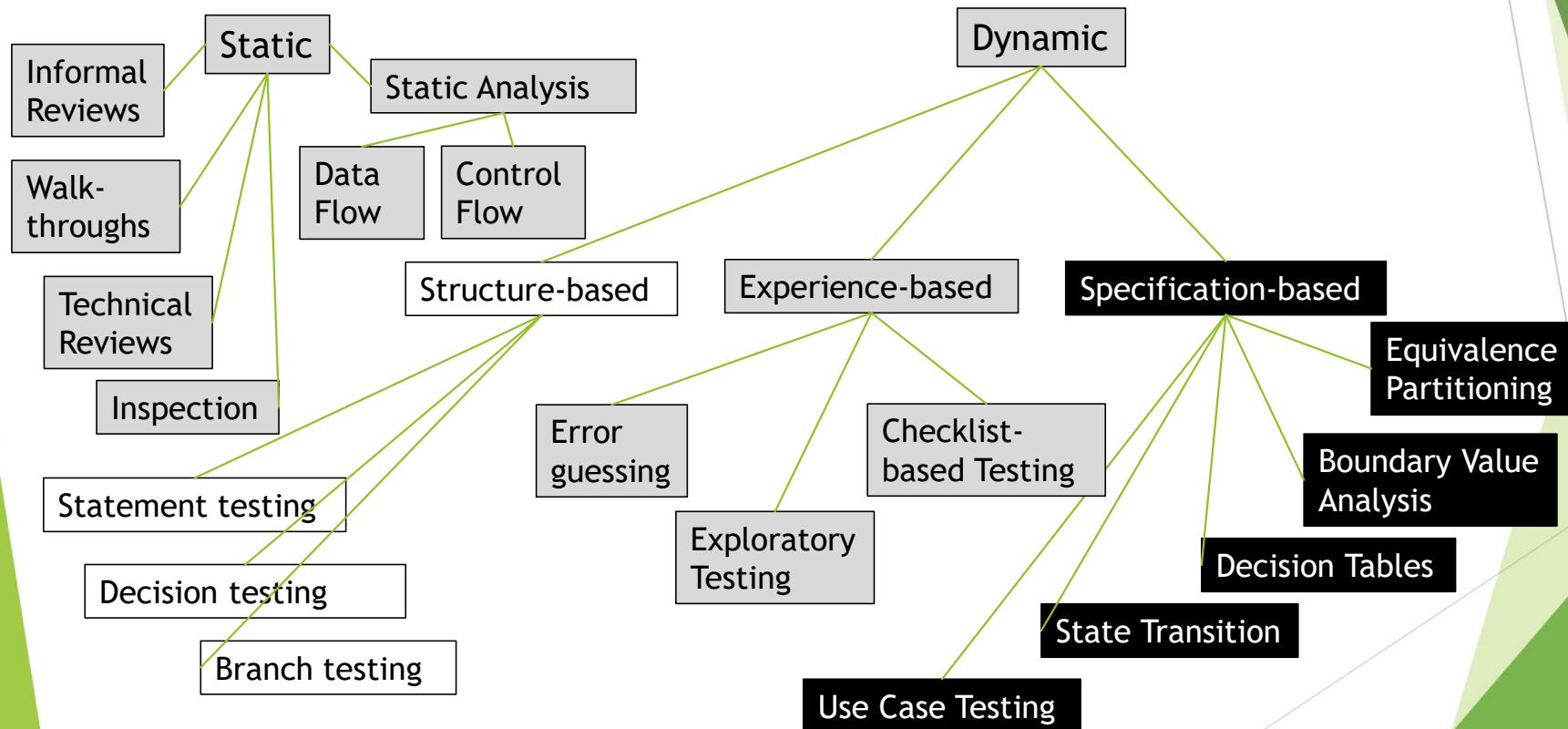




White-box Test Techniques

Introduction

Testing Techniques - visualized



White-box Test Techniques

- ▶ Aka structure-based techniques
- ▶ Based on an analysis of the architecture, detailed design, internal structure, code and processing **within** the test object
- ▶ Test cases are dependent on the **internal** structure and processing within the test object
- ▶ Software design is the basis for test cases, so test cases can not be created before the design or implementation of the test object



FL - 4.3.1 K2

Structure-based

Statement testing

Decision testing

Branch testing

White-box Test Techniques - Characteristics

- ▶ Test conditions, test cases and test data are derived from a test basis that may include
 - ▶ Code
 - ▶ Software architecture
 - ▶ Detailed design
 - ▶ Any other information source regarding the software structure
- ▶ Specifications are often used as an additional source of information to determine the expected outcome of test cases
- ▶ **Coverage** is measured based on the items tested **within a selected structure** (i.e. code or interfaces)

Structure-based

Statement testing

Decision testing

Branch testing

White-box Test Techniques - Characteristics (2)

- ▶ Based on how the software is constructed and designed, internal structure of the software is used to derive test cases
- ▶ Mainly on **component** level, but also on integration and system/ acceptance level
- ▶ Serves 2 purposes:
 - ▶ Test coverage **measurement** (from specification-based techniques)
 - ▶ Structural **test case design** (to increase test coverage)
- ▶ Basic coverage measure is:
$$\frac{\text{Nr of coverage items exercised}}{\text{Total nr of coverage items}} * 100 \%$$
- ▶ 100 % coverage does NOT equal 100 % tested
- ▶ Drawback is that code coverage measurement ONLY covers what HAS been written, the structure that is already there
 - ▶ Specified function not implemented - Specification-based techniques
 - ▶ Specification missing a function - Experience-based techniques



Structure-based

Statement testing

Decision testing

Branch testing

Coverage

- ▶ Can be measured on all levels:
 - ▶ System / Acceptance levels
 - ▶ Requirements, menu options, screens, database structural elements, files, etc
 - ▶ Integration level
 - ▶ API testing, Interfaces, interactions, call coverage of modules, objects or procedures, etc
 - ▶ Component level
 - ▶ Code coverage such as statement, decision, branch, path coverage
- ▶ Sometimes use coverage not related to the code (neuron coverage in neural network testing for example)
- ▶ Coverage can be measured for the any test techniques
 - ▶ Black-box test techniques
 - ▶ White-box test techniques
 - ▶ Experience-based test techniques

Structure-based

Statement testing

Decision testing

Branch testing

How far should you go?

- ▶ Depends on:

Organization

Industry

Risk

Test Object

- ▶ More rigorous test techniques and code coverage in:
 - ▶ Safety-critical environments
 - ▶ Mission-critical environments
 - ▶ High integrity environments

Structure-based

Statement testing

Decision testing

Branch testing



White-box Test Techniques

Statement Testing & Coverage

Statement testing & coverage

- ▶ Coverage items are executable statements
- ▶ Executable statements perform computations and control the output of the program
- ▶ Design test cases that exercise statement until an acceptable level of coverage

Example

```
program AbsoluteValueExample
program ExecutableExample
    implicit none

    integer :: x, y, z

    x = 10
    y = 5
    z = x + y

    write(*, *)
    "The sum of x and y is:", z

end program ExecutableExample
```

Structure-based

Statement testing

Decision testing

Branch testing

Statement testing & coverage - Example

- ▶ Exercises the executable statements in the code
- ▶ Statement Coverage is the percentage of executable statements that have been executed by a test suite is calculated by:

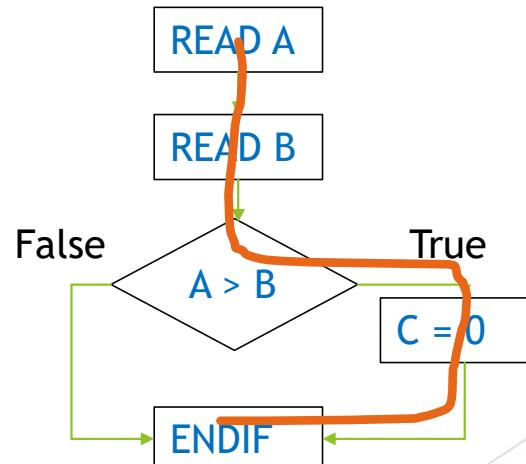
$$\frac{\text{Nr of statements executed}}{\text{Total nr of executable statements}} * 100 \%$$

- ▶ Principle works as follows:

Code:

```
READ A
READ B
IF A > B THEN C = 0
ENDIF
```

- ▶ How many testcases are needed to achieve 100 % statement coverage?



Structure-based

Statement testing

Decision testing

Branch testing

Statement testing & coverage - points of attention

- ▶ 100% Statement Coverage means:
 - ▶ All executable statements in the code have been exercised
 - ▶ Each statement with a defect has been executed
 - ▶ The defective statement MAY cause a failure and show there is a defect

BUT

- ▶ Defects are not detected in all cases - Data dependent defects (division by '0' for example)
- ▶ ONLY the statements are tested, not the logic (both sides of an IF statement for example)
- ▶ 100% Statement Coverage does NOT mean all decisions or branches are tested

Structure-based

Statement testing

Decision testing

Branch testing

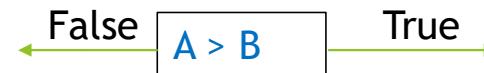


White-box Test Techniques

Decision Testing & Coverage

Decision testing & coverage

- ▶ Decision is an IF, loop control (DO-WHILE, REPEAT-UNTIL) or a CASE statement with two or more possible outcomes (True - False)
- ▶ A decision is both sides of a statement
- ▶ Decision testing exercises the decisions in the code and tests the executed code based on decision outcomes - being **conditional branches**
- ▶ Decision and branch coverage are NOT the same thing
 - ▶ Decision testing - considers ONLY CONDITIONAL branches in the code
 - ▶ Branch testing - considers BOTH CONDITIONAL AND UNCONDITIONAL branches in the code
- ▶ 100% Decision Coverage implies 100% statement coverage
- ▶ The percentage of decision outcomes that have been executed by a test suite



$$\frac{\text{Nr of decision outcomes executed}}{\text{Total nr of decision outcomes}} * 100 \%$$

Structure-based

Statement testing

Decision testing

Branch testing

Decision testing & coverage - Example

- Principle works as follows:

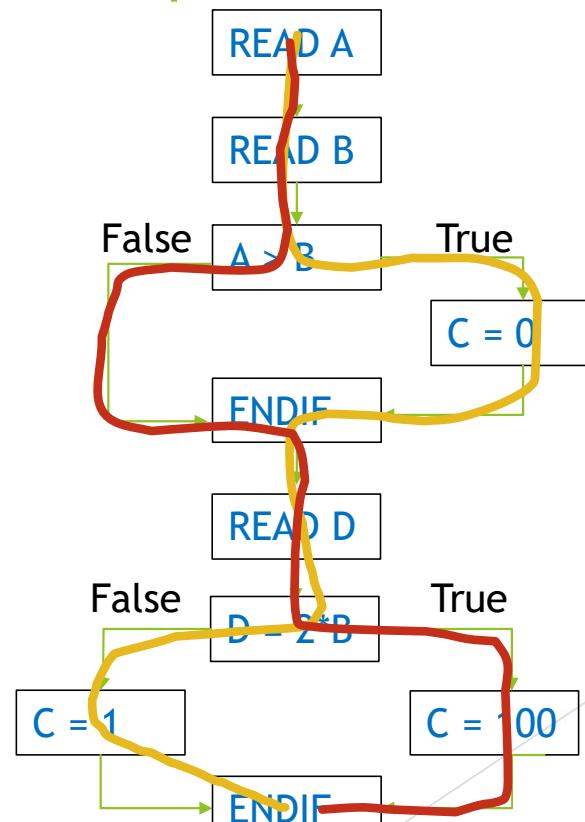
Code:

```

READ A
READ B
IF A > B THEN C = 0
ENDIF
READ D
IF D = 2*B THEN C = 100
ELSE C = 1
ENDIF

```

- What is the decision coverage with A = 12, B = 10, D = 5?
 - Answer: 50 % - 2 of the 4 decision outcomes
- How many testcases are needed to achieve 100 % decision coverage?
 - 2 (extra could be A = 9, B = 10, D = 20)



Structure-based

Statement testing

Decision testing

Branch testing

The value of Statement and Decision coverage

- ▶ **100 % Statement Coverage:**
 - ▶ Ensures all executable statements have been tested at least once
 - ▶ Does NOT ensure all decision logic has been tested
 - ▶ Provides less coverage than Decision testing
 - ▶ Helps to find defects in code that was not exercised by other tests
- ▶ **100 % Decision Coverage:**
 - ▶ Ensures all decision outcomes (all conditional branches) AND all executable statements
 - ▶ Includes all True and False outcomes, even if implied
 - ▶ Helps to find defects in code where other tests have not taken all outcomes into account
 - ▶ $100\% \text{ Decision Coverage} = 100\% \text{ Statement Coverage}$ - not vice versa

Structure-based

Statement testing

Decision testing

Branch testing

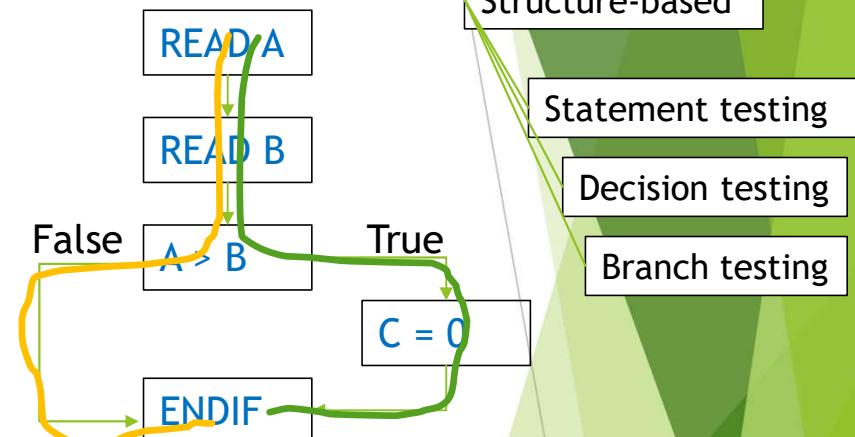


White-box Test Techniques

Branch Testing & Coverage

Branch testing & coverage

- ▶ A branch is transfer between two nodes in the control flow
- ▶ This transfer shows possible sequences in which statements are executed
- ▶ Each branch can be
 - ▶ Unconditional (i.e. straight line code) or
 - ▶ Conditional (i.e. a decision outcome, 'IF, THEN')
- ▶ Coverage items are branches
- ▶ Aim is to design tests to exercise branches in the code until an acceptable level of coverage is achieved



Branch testing & coverage (2)

- ▶ Branch testing focuses on the percentage of branches that have been executed by a test suite
- ▶ Difference with Decision testing is that where:
 - ▶ Decision testing focus ONLY on the conditional branches
 - ▶ Branch testing focuses on BOTH conditional and unconditional branches
- ▶ In practise Decision Testing and Branch Testing mostly give the same results
- ▶ Calculation of Branch Coverage:

$$\frac{\text{Nr of branches exercised by the test cases}}{\text{Total nr of branches}} * 100\%$$

Structure-based

Statement testing

Decision testing

Branch testing

Branch testing & coverage - Example

- Principle works as follows:

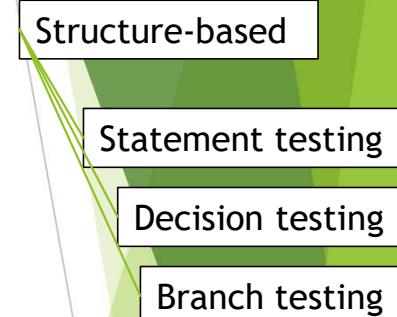
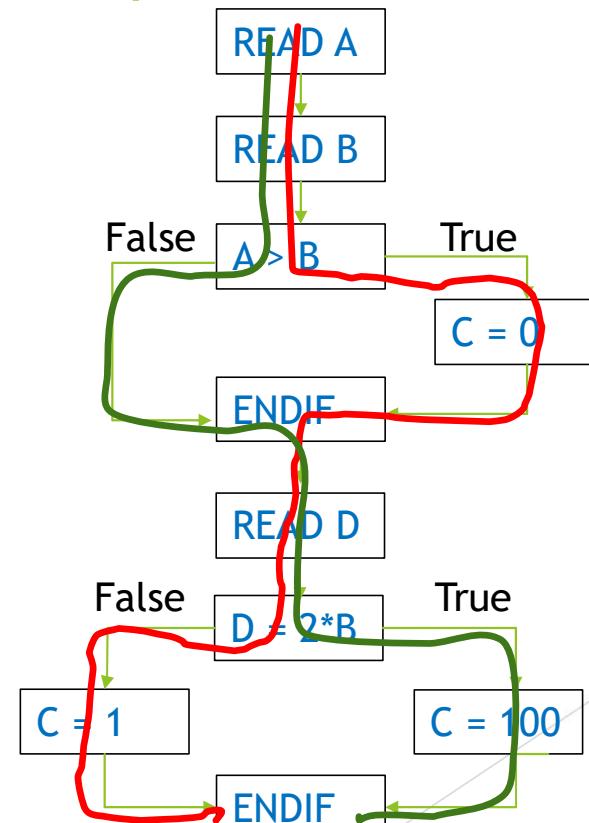
Code:

```

READ A
READ B
IF A > B THEN C = 0
ENDIF
READ D
IF D = 2*B THEN C = 100
ELSE C = 1
ENDIF

```

- What is the branch coverage with A = 12, B = 10, D = 5?
 - Answer: 50 % - 1 of the 2 branches
- How many testcases are needed to achieve 100 % branch coverage?
 - 2 (extra could be A = 9, B = 10, D = 20)



The value of Branch testing

- ▶ **100 % Branch Coverage:**
 - ▶ Ensures all branches in the code are exercised by tests
 - ▶ Conditional branches - typically
 - ▶ Decision outcomes (True - False from an 'IF' statement)
 - ▶ Outcome from a switch/case statement
 - ▶ Decision to exit or continue in a loop
 - ▶ Unconditional branches - Do NOT depend on a decision made in a statement
 - ▶ Defects will not be found in ALL cases - Branch Coverage is not the same as path coverage
 - ▶ $100\% \text{ Branch Coverage} = 100\% \text{ Decision Coverage} = 100\% \text{ Statement Coverage}$ - not vice versa

Structure-based

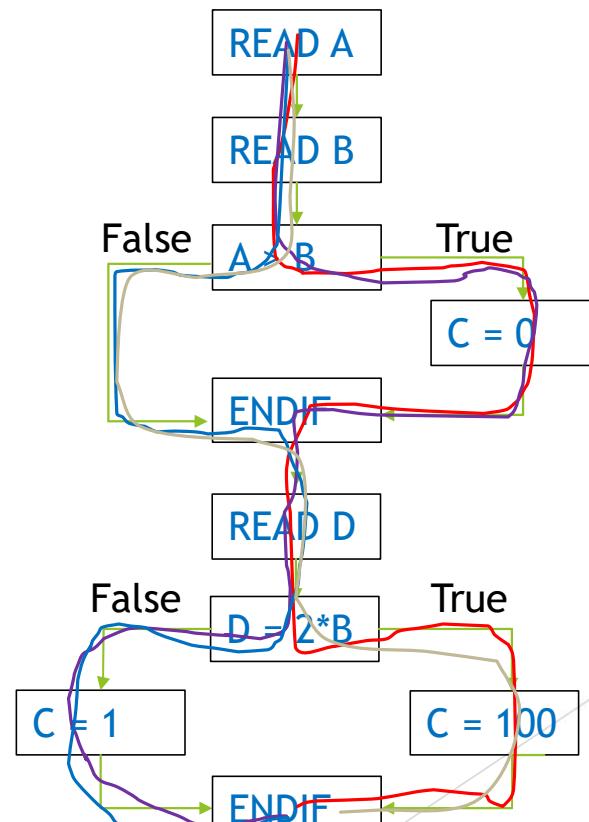
Statement testing

Decision testing

Branch testing

Path Testing & Coverage

- ▶ Taking branch testing one step further
- ▶ Does not only execute each path, but ALSO each combination between different statements
- ▶ 100 % Path Coverage = 100 % Branch Coverage
 $= 100\% \text{ Decision Coverage} = 100 \% \text{ Statement Coverage}$ - not vice versa
- ▶ In our example from before, 2 test cases will not cover it. We need to cover all combinations between our True - False statements.
- ▶ Result is 4 test cases needed



Structure-based

Statement testing

Decision testing

Branch testing

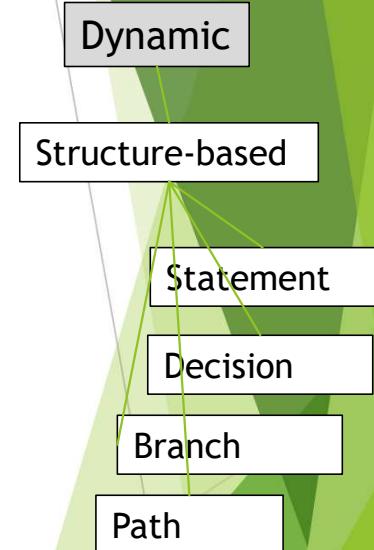


White-box Test Techniques

The Value of White-box Testing

Why White-box Testing?

- ▶ Provides an objective measurement of code coverage
- ▶ Provides information needed to create additional tests to **increase** the coverage and increasing the confidence in the code
- ▶ The entire software implementation is considered, defects can be found even if the software specs are:
 - ▶ Vague
 - ▶ Outdated
 - ▶ Incomplete
- ▶ However, missing requirements will not be found by white-box testing
- ▶ Can also be used in static testing:
 - ▶ Dry runs of the code
 - ▶ Reviewing:
 - ▶ Code that is not ready for execution, pseudo-code, high-level or top-down logic that can be modeled with a Control Flow Graph.





Experience-based Test Techniques

Introduction

Experience-bases Techniques - Characteristics

- ▶ Test conditions, test cases and test data are derived from a test basis that may include:
 - ▶ Knowledge and expertise of testers
 - ▶ Knowledge and expertise of developers
 - ▶ Knowledge and expertise of users
 - ▶ Knowledge and expertise of other stakeholders
- ▶ Knowledge includes:
 - ▶ The expected use of the software
 - ▶ Its environment
 - ▶ Likely defects and the distribution of defects



Dynamic

Experience-based

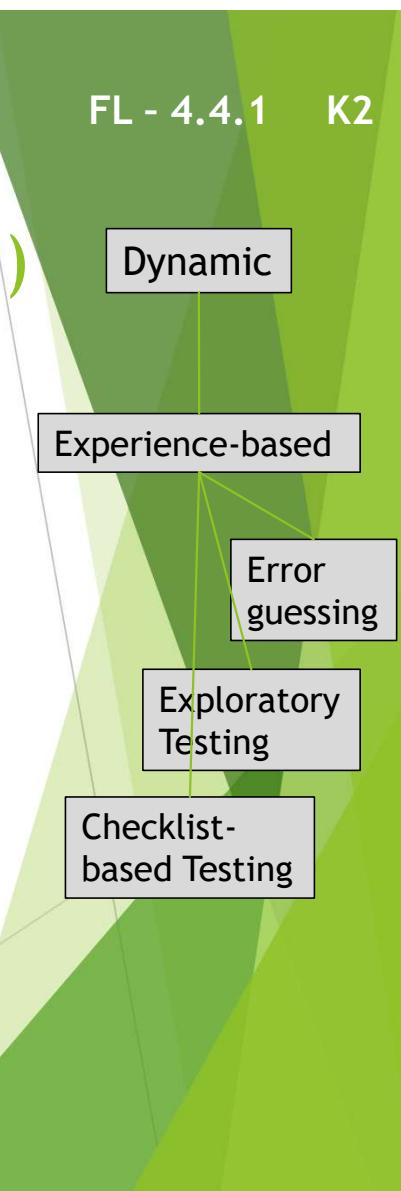
Error
guessing

Exploratory
Testing

Checklist-
based Testing

Experience-bases Techniques - Characteristics (2)

- ▶ Based on knowledge, experience, imagination and intuition
- ▶ Some defects are hard to find using more systematic approaches
- ▶ Should in general be used as a complement to more formal techniques
- ▶ Effectiveness and coverage purely depends on the testers' approach and experience
- ▶ Coverage can be difficult to assess and may not be measurable





Experience-based Test Techniques

Error Guessing

Error guessing

A technique that anticipates the occurrence of mistakes, defects and failures, based on the knowledge of the tester:

- ▶ How the application worked in the past
- ▶ What types of mistakes the developers tend to make
- ▶ The type of defects that result from these errors made by developers
- ▶ Failures that have occurred in other applications
- ▶ No rules, tester is encouraged to think of situations the software will not be able to cope with:
 - ▶ Division by zero
 - ▶ Empty files
 - ▶ Wrong kind of data or input
 - ▶ 'That should not happen' or 'There is no risk' is a trigger to test that condition as assumption is the mother...

FL - 4.4.1 K2

Dynamic

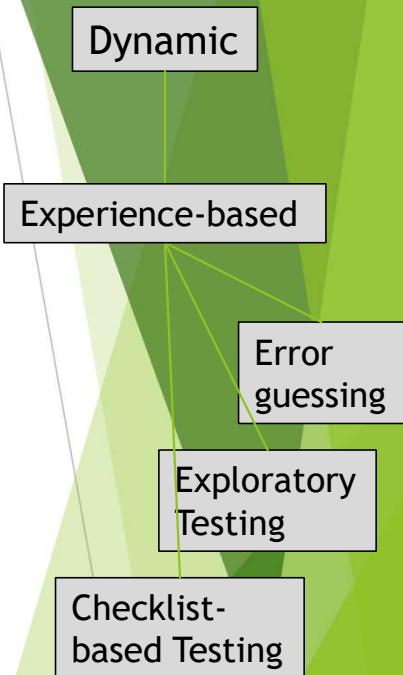
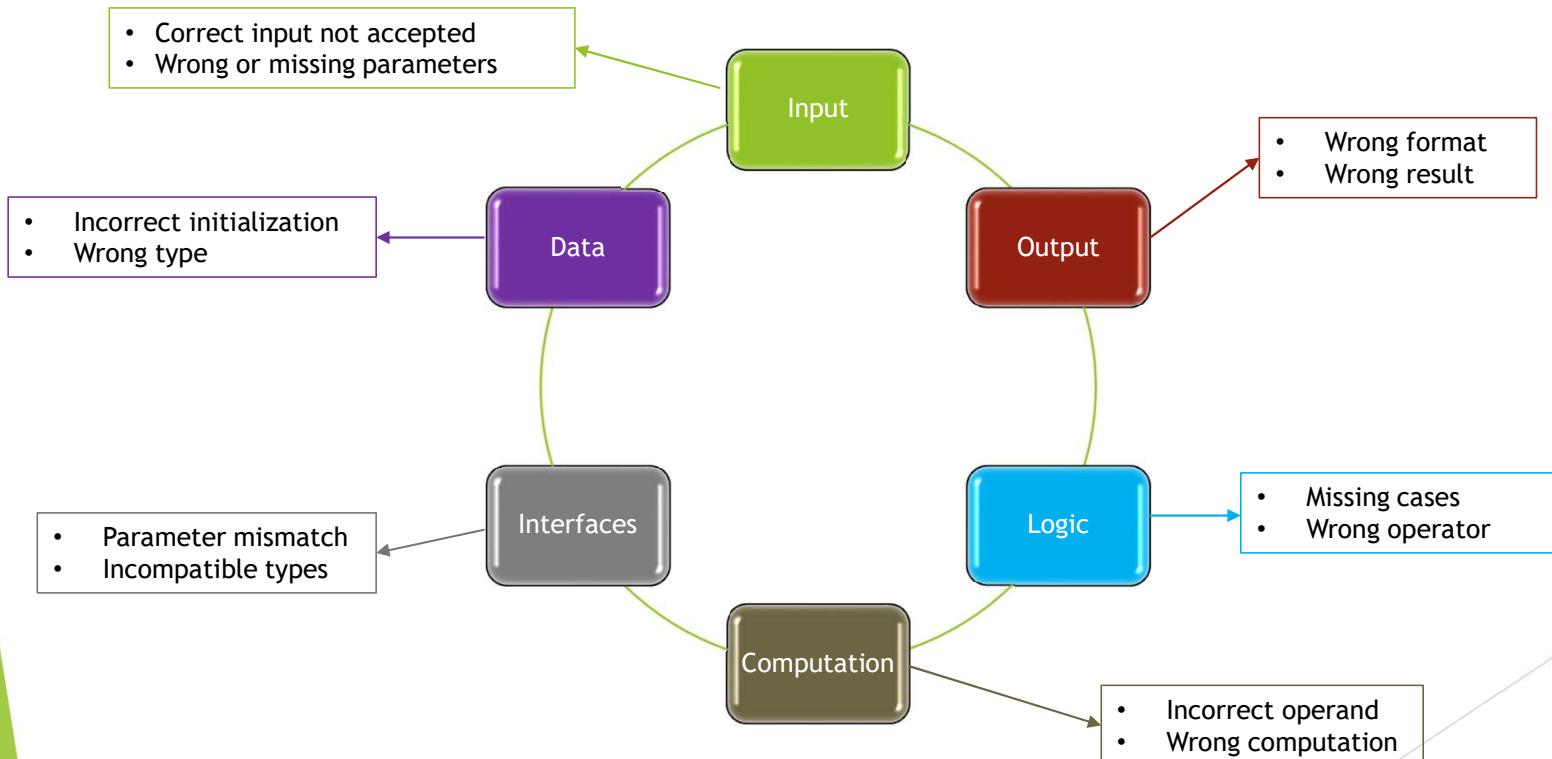
Experience-based

Error
guessing

Exploratory
Testing

Checklist-
based Testing

Error guessing - typical errors, defects and failures



Error guessing - Examples

- ▶ New date field:
 - ▶ Feb 29, European date format, US date format
- ▶ Division by zero
- ▶ Submitting a form without entering values
- ▶ Entering invalid values
- ▶ Replacing @ and dot(.) in email addresses
(eg: wouter.vrijen.protestsolutions@net)
- ▶ Last names with non alphabetical characters (eg: ronald.o'neil@testing.com)
- ▶ Ignoring the default maximum and minimum characters to be entered in some fields

FL - 4.4.1 K2

Dynamic

Experience-based

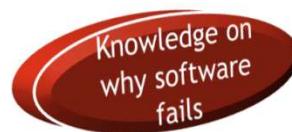
Error
guessing

Exploratory
Testing

Checklist-
based Testing

Fault attacks

- ▶ Methodical and focused way of error guessing
- ▶ List possible errors, defects and failures based on experience
- ▶ These lists can be based on:



- ▶ Design tests that will:
 - ▶ Identify associated defects associated with the errors
 - ▶ Expose the defects or
 - ▶ Cause the failures

FL - 4.4.1 K2

Dynamic

Experience-based

Error
guessing

Exploratory
Testing

Checklist-
based Testing

Fault attacks - Examples

Modify or corrupt data stored or processed by the software

Data Integrity Testing

Typical
Fault
Attacks

Operating systems, browsers, devices, or third-party components

Compatibility Testing

Introduce concurrent access or race conditions in multi-threaded or distributed systems

Concurrency Testing

Input Validation Testing

Entering special characters, long strings, unexpected data formats

Exception Handling Testing

Dividing by zero, accessing null references, exceeding array bounds

Configuration Testing

Changing settings related to file paths, network connections, or system resources

FL - 4.4.1 K2

Dynamic

Experience-based

Error guessing

Exploratory Testing

Checklist-based Testing

The process of error guessing

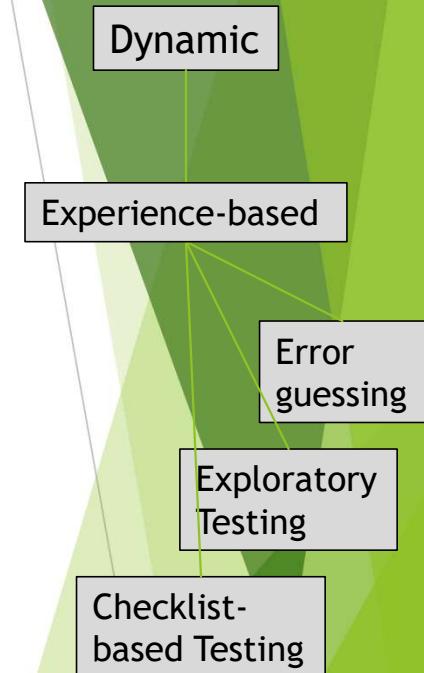
Scenario

Imagine you are testing a web application's login functionality. The application allows users to log in with their username and password to access their account dashboard.

1. Identification of Potential Errors

Begin by identifying potential errors or faults that could occur in the login functionality based on your experience and knowledge of common issues in authentication systems. Some potential errors might include:

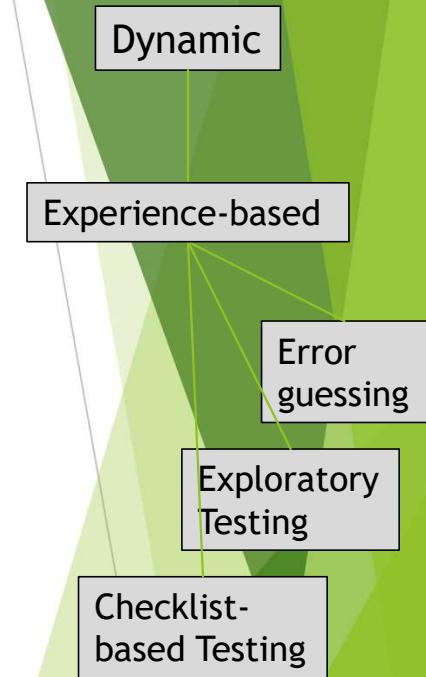
- ▶ Invalid username/password combination
- ▶ Empty username or password fields
- ▶ Username or password containing special characters
- ▶ Long username or password exceeding character limits
- ▶ SQL injection attacks by entering special characters in the username or password fields



The process of error guessing (2)

2. Based on the identified potential errors, design test cases to systematically test each scenario. For example:

- Test Case 1: Attempt to login with a valid username and an incorrect password.
- Test Case 2: Attempt to login with a valid username and a valid password.
- Test Case 3: Attempt to login with an empty username and a valid password.
- Test Case 4: Attempt to login with a valid username and an empty password.
- Test Case 5: Attempt to login with a username containing special characters.
- Test Case 6: Attempt to login with a password containing special characters.
- Test Case 7: Attempt to login with a long username (exceeding character limit).
- Test Case 8: Attempt to login with a long password (exceeding character limit).
- Test Case 9: Attempt to login with SQL injection attempt in the username or password fields.



The process of error guessing (3)

3. Execute Test Cases

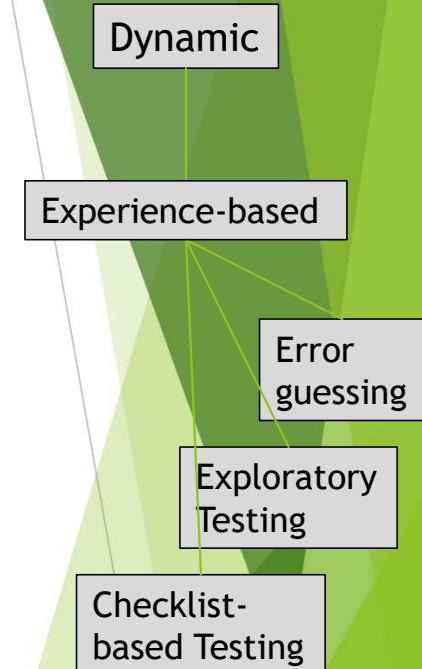
Execute each test case against the login functionality of the web application. Record the actual results and any observed deviations from expected behavior.

4. Analyze Results

Analyze the results of each test case to identify any errors or unexpected behavior encountered during the testing process. Pay attention to any discrepancies between the expected and actual outcomes.

5. Report and Document Findings

Document the findings, including any errors or faults identified during the testing process. Report the details of each issue, including steps to reproduce, observed behavior, and potential impact on the application's functionality or security.



The process of error guessing (4)

6. Retest and Verify Fixes

After developers address the reported issues, retest the affected areas to ensure that the fixes have been successfully implemented and that no new errors have been introduced.

This way, testers can methodically and structurally identify and address potential errors or faults in software applications, helping to improve the overall quality, reliability, and security of the software.

FL - 4.4.1 K2

Dynamic

Experience-based

Error
guessing

Exploratory
Testing

Checklist-
based Testing



Experience-based Test Techniques

Exploratory Testing

Exploratory testing

- ▶ Informal test technique
- ▶ During test execution, the tester designs, executes, logs and evaluates tests
- ▶ Is used to:
 - ▶ Learn more about a component or system
 - ▶ Explore the test object with more focused tests
 - ▶ Create tests for untested areas
 - ▶ Identify and exercise coverage items
- ▶ Can be conducted using session-based testing:
 - ▶ Defined timebox
 - ▶ Tester uses a test charter containing test objectives to guide the testing
 - ▶ Test objectives may be treated as high level test conditions
 - ▶ Tester may use test session sheets to document the steps and the discoveries
 - ▶ Often concluded with a debriefing

Dynamic

Experience-based

Error
guessing

Exploratory
Testing

Checklist-
based Testing

Exploratory testing



Dynamic

Experience-based

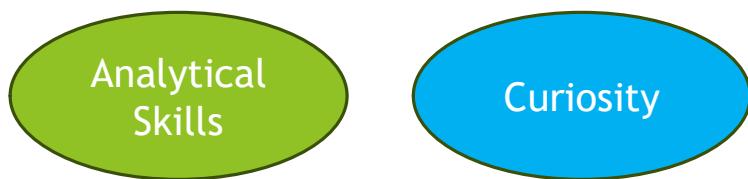
Error
guessing

**Exploratory
Testing**

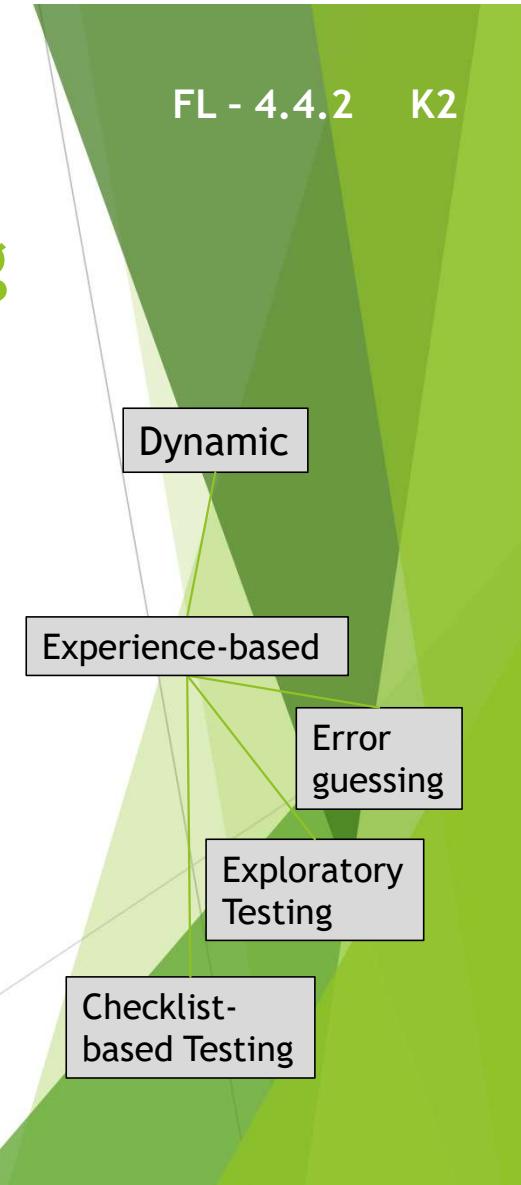
Checklist-
based Testing

When and how to use exploratory testing

- ▶ Most useful when there are:
 - ▶ Few or inadequate specifications
 - ▶ Severe time pressure
 - ▶ As complement to more formal techniques
- ▶ Strongly associated with reactive testing strategies
- ▶ Effectiveness grows with the knowledge, skills and experience of the tester



- ▶ Can include the use of black-box, white-box or other experience-based techniques



Exploratory testing - Example

- ▶ Shopping website - what do you expect?
- ▶ During this process, you might learn certain things
- ▶ As a tester, you need to:
 - ▶ verify whether a system is working as expected
 - ▶ check if that system is not behaving in a way which is not expected.
- ▶ Success factors are:
 - ▶ Your mission should be clear.
 - ▶ Create notes and report on what you are doing and how a system is behaving
 - ▶ Use your experience, knowledge and skills
 - ▶ Learn, observe and then come up with new test cases.



Dynamic

Experience-based

Error
guessing

Exploratory
Testing

Checklist-
based Testing

Myths about Exploratory Testing

- ▶ Exploratory Testing does not need planning or documentation
- ▶ Exploratory testing is the same as Ad-hoc testing
- ▶ Exploratory testing is not effective on complex systems
- ▶ Choose Exploratory Testing **OR** scripted testing



Dynamic

Experience-based

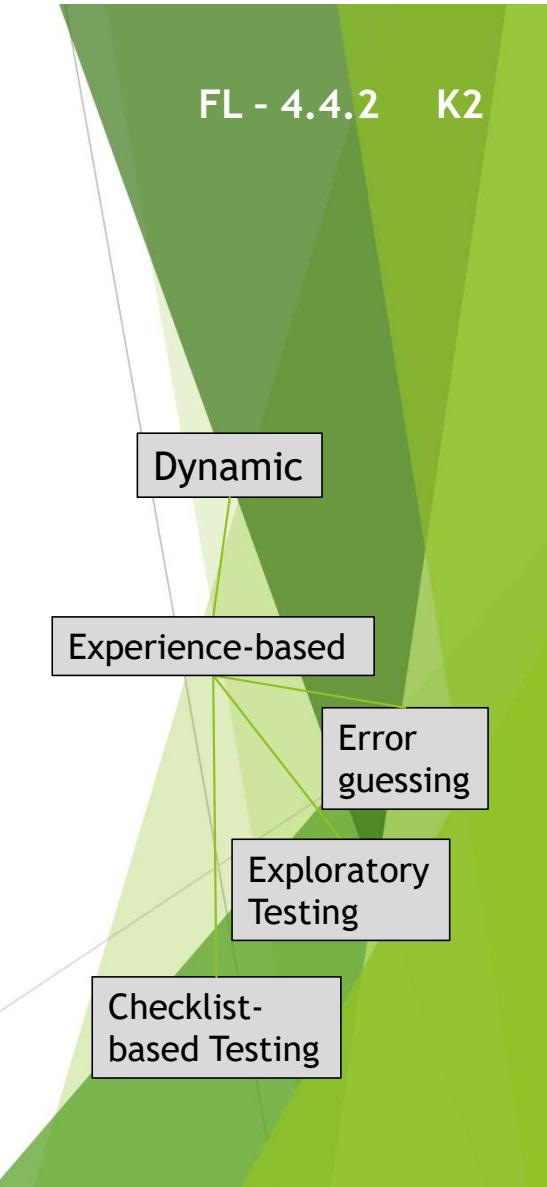
Error
guessing

Exploratory
Testing

Checklist-
based Testing

Test Automation in Exploratory Testing

- ▶ Can NOT be replaced by test automation
- ▶ You CAN NOT automate:
 - ▶ Creativity
 - ▶ Human curiosity
 - ▶ Random inventions
- ▶ Combine Exploratory Testing and test automation
- ▶ You CAN automate:
 - ▶ The creation of random test data
 - ▶ The execution of functional tests
 - ▶ Output logging and reporting



Common pitfalls in Exploratory Testing

- ▶ Lack of Test Coverage
- ▶ Limited Time and Resources
- ▶ Documentation and Reporting
- ▶ Adapting to Agile
- ▶ Subjectivity in Testing
- ▶ Skill and Knowledge Gaps
- ▶ Defect Reproducibility
- ▶ Managing Test Data
- ▶ Collaboration and Communication



Dynamic

Experience-based

Error
guessing

Exploratory
Testing

Checklist-
based Testing



Experience-based Test Techniques

Checklist-based Testing

Checklist-based testing

- ▶ Testers design, implement and execute tests to cover test conditions found in a checklist
- ▶ To create the checklist, the tester may:
 - ▶ Create a new checklist
 - ▶ Amend an existing checklist
 - ▶ Use an existing checklist without changes
- ▶ Checklists are based on:



Experience

Knowledge about
importance for
the user

Understanding
about why and
how software fails

Dynamic

Experience-based

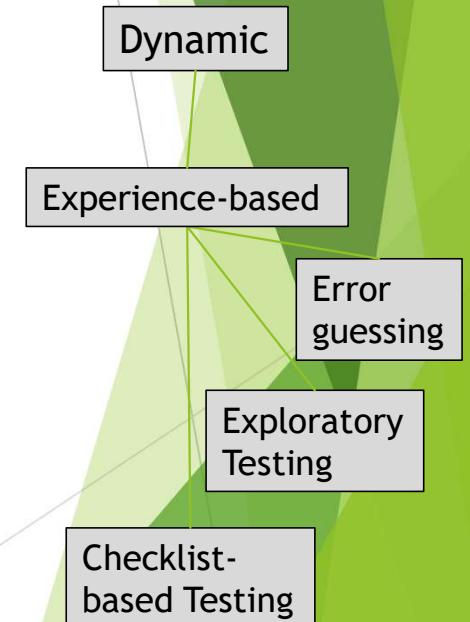
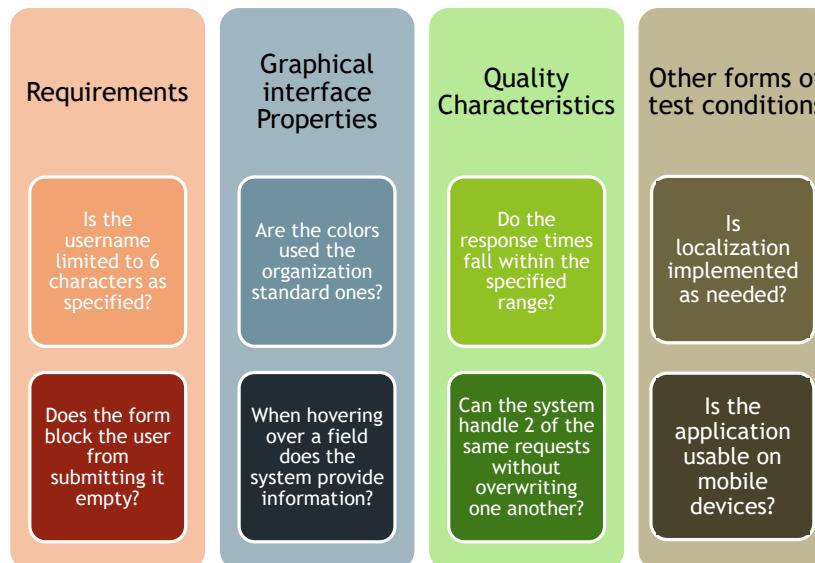
Error
guessing

Exploratory
Testing

Checklist-
based Testing

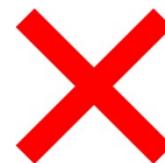
Checklist-based testing (2)

- ▶ Checklist items are often phrased as questions
- ▶ Each item should be separately ‘checkable’
- ▶ Checklist items may refer to:



Checklist-based testing (3)

- ▶ Checklists can support different test types (Functional, Non-Functional)
- ▶ If there are no detailed test cases, checklists can provide guidelines and consistency
- ▶ If checklists are high level, variability will most likely occur, resulting in:
 - ▶ Greater coverage
 - ▶ Less repeatability
- ▶ Checklist should NOT contain:
 - ▶ Items that can be checked automatically
 - ▶ Items that should be entry or exit criteria
 - ▶ Items that are too general



Dynamic

Experience-based

Error
guessing

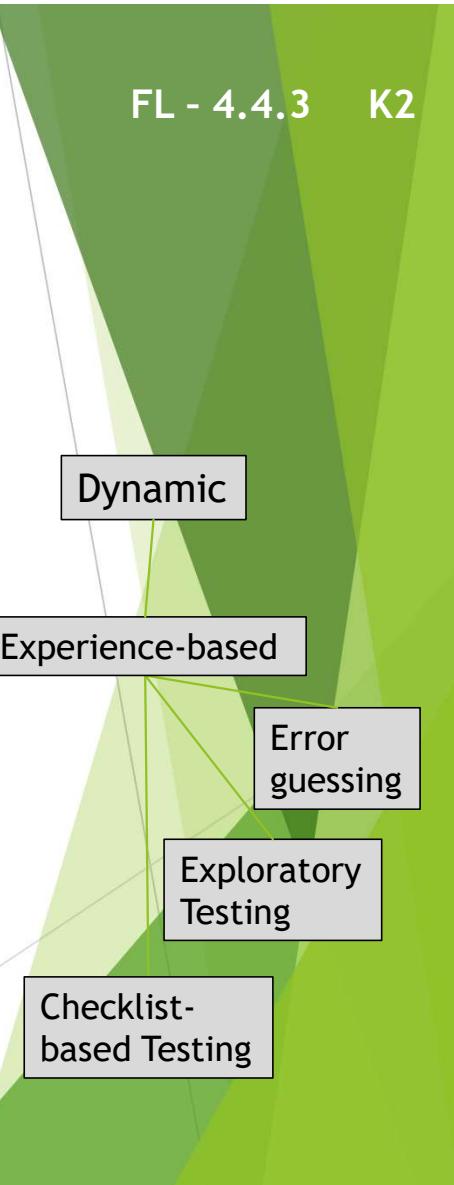
Exploratory
Testing

Checklist-
based Testing

Checklist-based testing - Examples

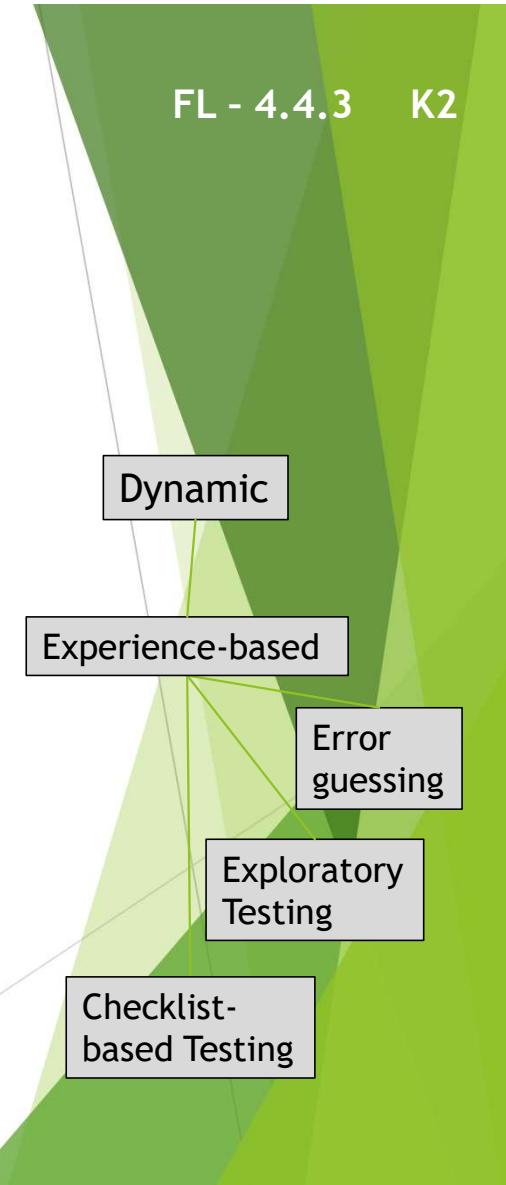
- ▶ A checklist to make sure none of the software modules are forgotten
- ▶ A checklist to make sure that none of the non-functional quality characteristics are forgotten:
 - ▶ Performance & stress
 - ▶ Usability
 - ▶ Portability
- ▶ A checklist to make sure all parts of a workflow have been tested:
 - ▶ Login, open record, close record, create record, save record, edit record, close record, delete record

With more experience the generic checklists can become more specific and clearer. They are guidelines rather than test cases.



Checklist-based testing - Benefits

- ▶ Structured approach to testing
- ▶ Promote consistency and repeatability
- ▶ Improve communication between team members
- ▶ Helps new testers test new products more confidently and more efficiently



Checklist-based testing - Risks

- ▶ Checklist become gradually less effective - developers learn from their mistakes
- ▶ Constant maintenance is needed:
 - ▶ To add newly found defects
 - ▶ To update based on defect analysis
- ▶ Checklists easily become too long
- ▶ Interpretation is subjective



Dynamic

Experience-based

Error
guessing

Exploratory
Testing

Checklist-
based Testing



Collaboration-based Test Approaches





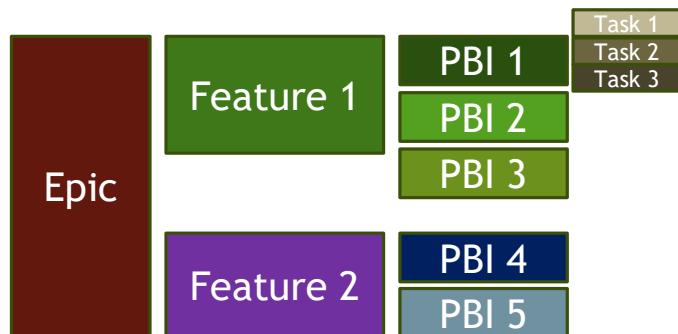
Collaborative User Story Writing

User Stories

Definition

'A feature that will be valuable to either a user or purchaser of a system or software'

- ▶ *Should be written in natural language, no jargon*
- ▶ *Should not be written too complex*
- ▶ *Are often broken up into smaller stories*



User Story Writing - 3C Concept



Card - Simple card or other medium accurately describing the user story.



Conversation - Explains how the software will be used and can be documented or verbal. Begins during release-planning and continue while the story is scheduled to be picked up.



Confirmation - Acceptance criteria are used to confirm whether a story is done and may concern multiple user stories. Tests should include positive and negative scenarios. Various participants play the role of tester (dev, specialists, testers, etc)

User Story Writing - Issues

- ▶ Poor specifications (user stories) often result in failure
- ▶ Bad specs can result from:
 - ▶ Lack of insight of the user on what is needed
 - ▶ No global vision for the system
 - ▶ Redundant features
 - ▶ Contradictory features
 - ▶ Miscommunications
 - ▶ Misinterpretations



How can you build a solution to a problem when the problem is not clear?

Collaborative User Story Writing

- ▶ Software development heavily depends on user stories
- ▶ User stories are captured and acceptance criteria defined from different perspectives:
 - ▶ Developers
 - ▶ Testers
 - ▶ Business representatives
- ▶ Shared vision through frequent informal reviews
- ▶ User stories should address functional and non-functional requirements
- ▶ Acceptance criteria define when a feature is done



As a <ROLE> I want <GOAL TO BE ACCOMPLISHED> so that I can
<RESULTING BUSINESS VALUE FOR THE ROLE>

Followed by the ‘Acceptance Criteria’

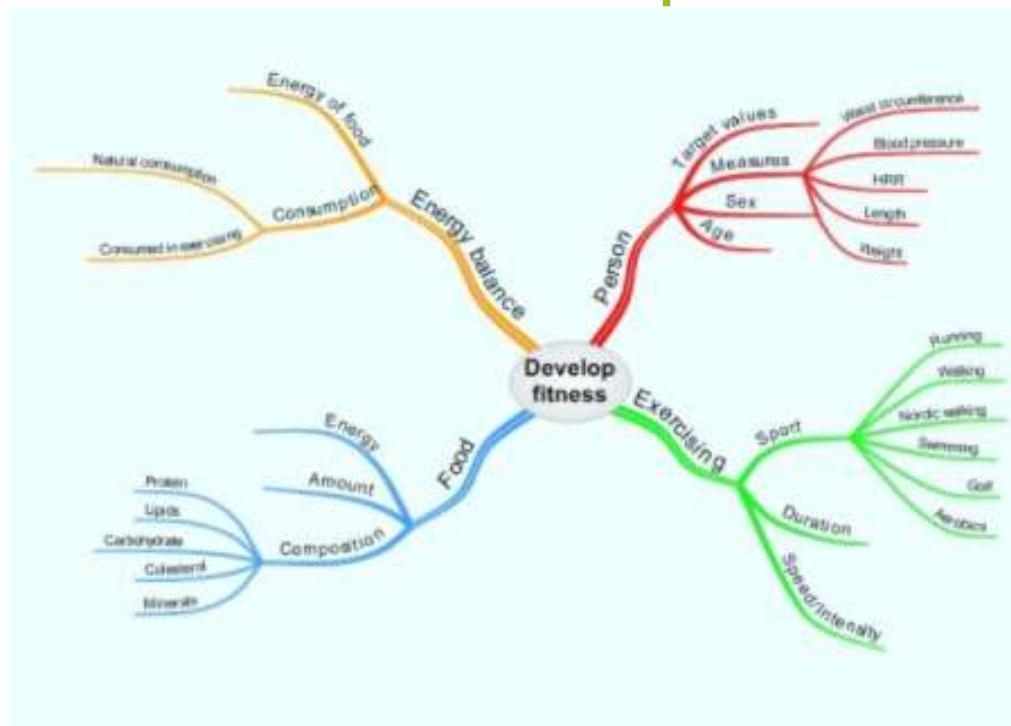
User Story Creation - Tester perspective

- ▶ Identifying missing details
- ▶ Identifying non-functional requirements
- ▶ Asking open-ended questions to the business representatives about the story
- ▶ Proposing ways to test the story
- ▶ Confirming the acceptance criteria



User Story Creation - Techniques

- ▶ Brainstorming
- ▶ Mind mapping
- ▶ INVEST
 - ▶ Independent
 - ▶ Negotiable
 - ▶ Valuable
 - ▶ Estimable
 - ▶ Small
 - ▶ Testable





Writing Acceptance Criteria

User Story Creation - 3C Concept



Card - Simple card accurately describing the user story.



Conversation - Explains how the software will be used and can be documented or verbal. Begins during release-planning and continue while the story is scheduled to be picked up.



Confirmation - Acceptance criteria are used to confirm whether a story is done and may concern multiple user stories. Tests should include positive and negative scenarios. Various participants play the role of tester (dev, specialists, testers, etc)

What are Acceptance Criteria

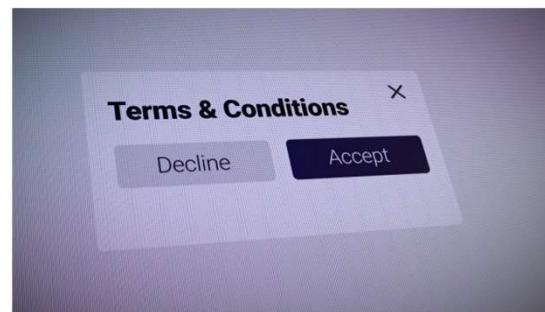
- ▶ The conditions that the implementation of a user story is correct
- ▶ Define when a user story must be accepted by the stakeholders
- ▶ Test Conditions to be exercised by tests
- ▶ Acceptance criteria should cover:
 - ▶ Functional characteristics AND
 - ▶ Non-Functional characteristics



Why do we need acceptance criteria?

Acceptance Criteria:

- ▶ Define the scope of the User Story
- ▶ Reach consensus among the stakeholders
- ▶ Describe both positive AND negative scenarios
- ▶ Serve as a basis for the user story acceptance testing
- ▶ Allow accurate planning and estimation



How to write Acceptance Criteria?

- ▶ Acceptance criteria should be:
 - ▶ Well-defined
 - ▶ Unambiguous

- ▶ Formats to write Acceptance Criteria:
 - ▶ Scenario-oriented
 - ▶ Rule-oriented



Scenario-oriented Acceptance Criteria

- ▶ Aims to describe how a function or task operates within the context of a user scenario
- ▶ Puts focus:
 - ▶ On the expected workflow of the user
 - ▶ And on the navigation of the application
- ▶ Often using the BDD format:
 - ▶ Given (precondition)
 - ▶ When (action)
 - ▶ Then (result)

Scenario-oriented Acceptance Criteria

User Story: As a bank card user, I want to be able to request cash from my account at an ATM so that I will be able to receive available money from my account quickly and in different places.

Scenario: Requesting the cash from an overdrawn (negative balance) account

Given The account is overdrawn

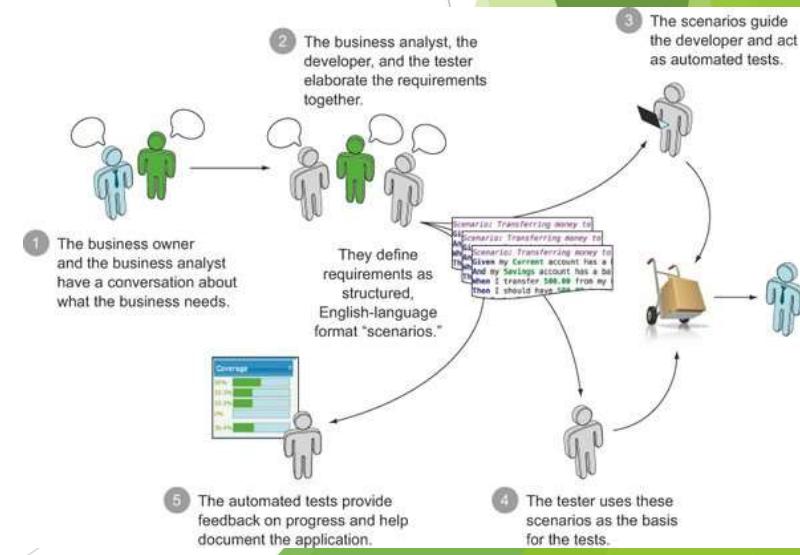
And The card is valid

When The bank card user requests cash from an ATM

Then Ensure the cash is not dispensed

Behavior-Driven Development

- ▶ BDD helps developers collaborate with other stakeholders to define accurate tests **focused on business needs**
- ▶ Allows a developer to focus on testing the code based on the expected behaviour
- ▶ As tests are based on the exhibited behaviour, the tests are easier to understand
- ▶ Specific frameworks to define acceptance criteria based on the ‘given / when / then’ format:
 - ▶ Given some initial context,
 - ▶ When an event occurs,
 - ▶ Then ensure some outcomes.
- ▶ From these requirements, the framework generates code (test classes) that dev can use to create test cases



Rule-oriented Acceptance Criteria

- ▶ Define the scope and functionality of a user story
- ▶ Uses a list of:
 - ▶ Functional and
 - ▶ Behavioral parameters
- ▶ Format looks more like ‘Waterfall’ scenarios - each function listed individually
- ▶ This can be in the form of a:
 - ▶ Bullet point verification list
 - ▶ Tabulated form of input-output mapping

Rule-oriented Acceptance Criteria

User Story: As a patient, I want to be able to schedule appointments with doctors in the healthcare app with specific rules and constraints to ensure efficient scheduling and to avoid conflicts

Rule # 1

Patients should be able to request appointments with specific doctors from within the app

Rule # 2

Appointments requests must include the preferred date and time

Rule # 3

The app should check the availability of the doctor for the requested date and time

Rule # 4

If the doctor is unavailable, the system should suggest alternative slots or ask the patient to choose a different doctor

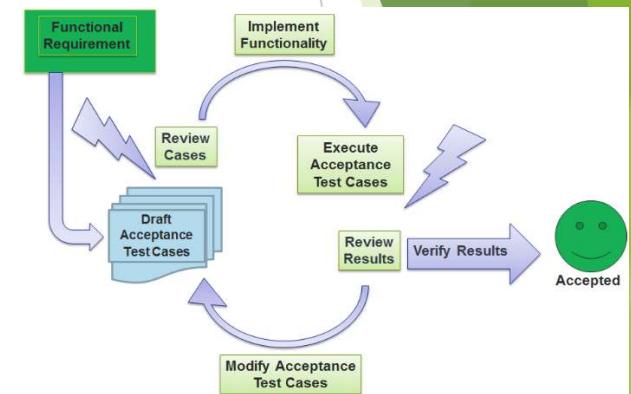


Use Acceptance Test-Driven
Development to derive test
cases

Acceptance Test-Driven Development

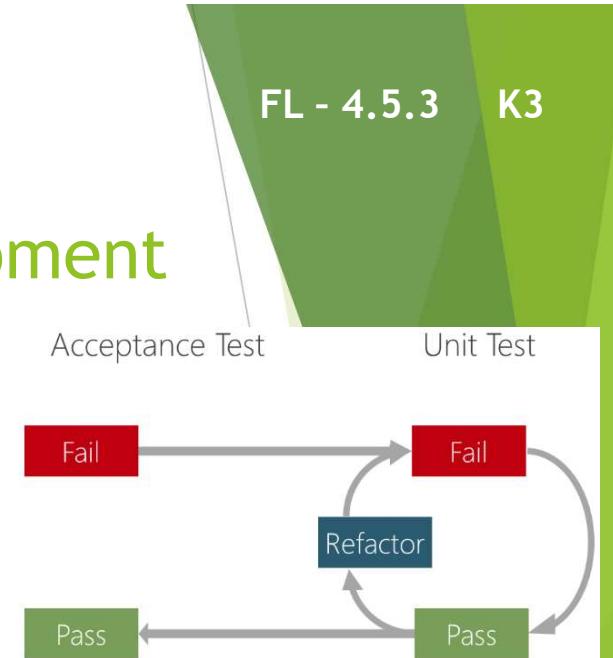
- ▶ Acceptance criteria and tests are defined during the creation of the user stories
- ▶ Encourages collaboration amongst the business, developer and tester
- ▶ Every stakeholder should understand how the software component has to behave and what is needed to ensure this behaviour
- ▶ ATDD creates reusable tests for regression testing
- ▶ Tools support creation and execution of tests (often in the CI process)
- ▶ Tools can connect to data and service layers of the application (enabling execution on system or acceptance levels)
- ▶ ATDD allows quick resolution of defects and validation of feature behaviour
- ▶ Helps determine if the acceptance criteria are met for the feature

Is the code doing what it is supposed to do?



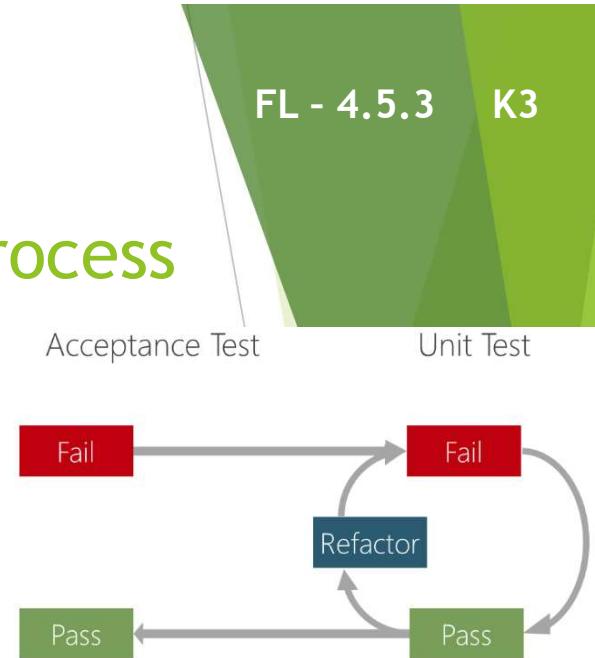
Applying Acceptance Test-Driven Development

- ▶ Test-first approach
- ▶ Tests (manual or automated) are written before coding
- ▶ Test cases are written by the team:
 - ▶ Developer
 - ▶ Tester
 - ▶ Business representative



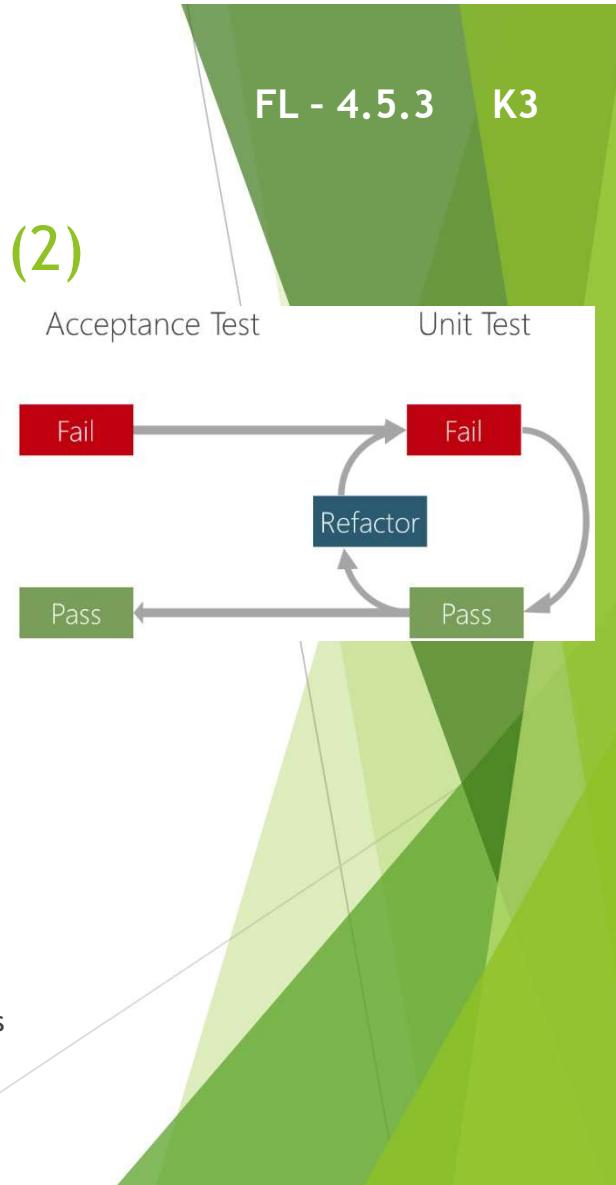
Acceptance Test-Driven Development - Process

1. First a specification workshop takes place
 - ▶ Workshop is a collaboration between developers, testers and the business
 - ▶ User stories are:
 - ▶ Analysed
 - ▶ Discussed
 - ▶ Written
 - ▶ Goal of the workshop is to fix any of the below in the user stories
 - ▶ Incompleteness
 - ▶ Ambiguities
 - ▶ Defects



Acceptance Test-Driven Development - Process (2)

2. Tests are created
 - ▶ By the team or
 - ▶ By the tester
 - ▶ An independent person (e.g. a business representative) will validate the tests
 - ▶ The tests are based on the Acceptance Criteria
 - ▶ Tests are examples that:
 - ▶ Describe the specific characteristics of the user story
 - ▶ Will help the team to implement the user story correctly
 - ▶ Start with basic examples and open questions
 - ▶ The tests can be created using black-box, white-box and experience-based test techniques



Acceptance Test-Driven Development - Tests

1. Positive Tests

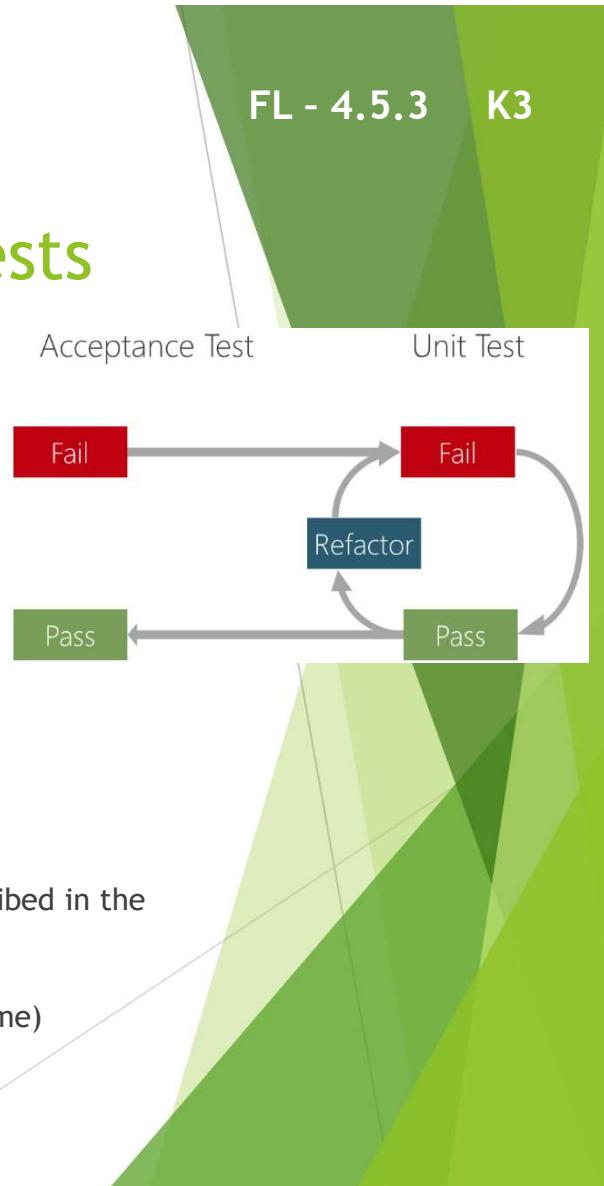
- ▶ Confirm the correct behaviour without exception or error condition
- ▶ Entail the sequence of activities executed if all goes as expected

2. Negative Path tests

3. Non-Functional attributes (e.g. performance, usability, etc)

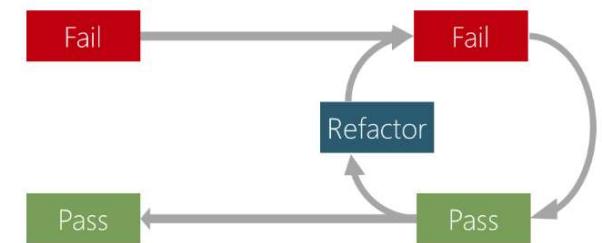
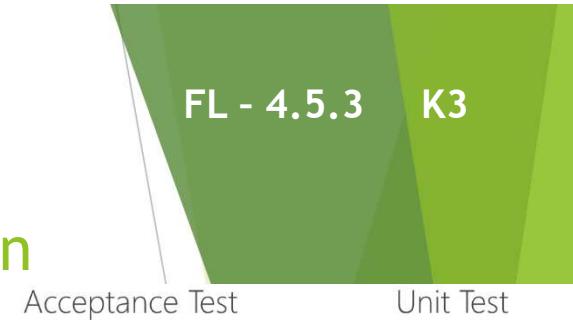
► The tests should:

- ▶ Cover all characteristics of the user story
- ▶ NOT add to the user story (Acceptance Criteria may specify some of the issues described in the User Story)
- ▶ Deal with unique characteristics of the user story (no two tests/examples for the same)



Acceptance Test-Driven Development - Automation

- ▶ Test Cases can be captured in a format supported by a test automation framework
- ▶ In that way
 - ▶ The developer implements the feature described by a user story
 - ▶ The developer can automate the test cases **at the same time**
 - ▶ Like this, the Acceptance Tests become executable requirements

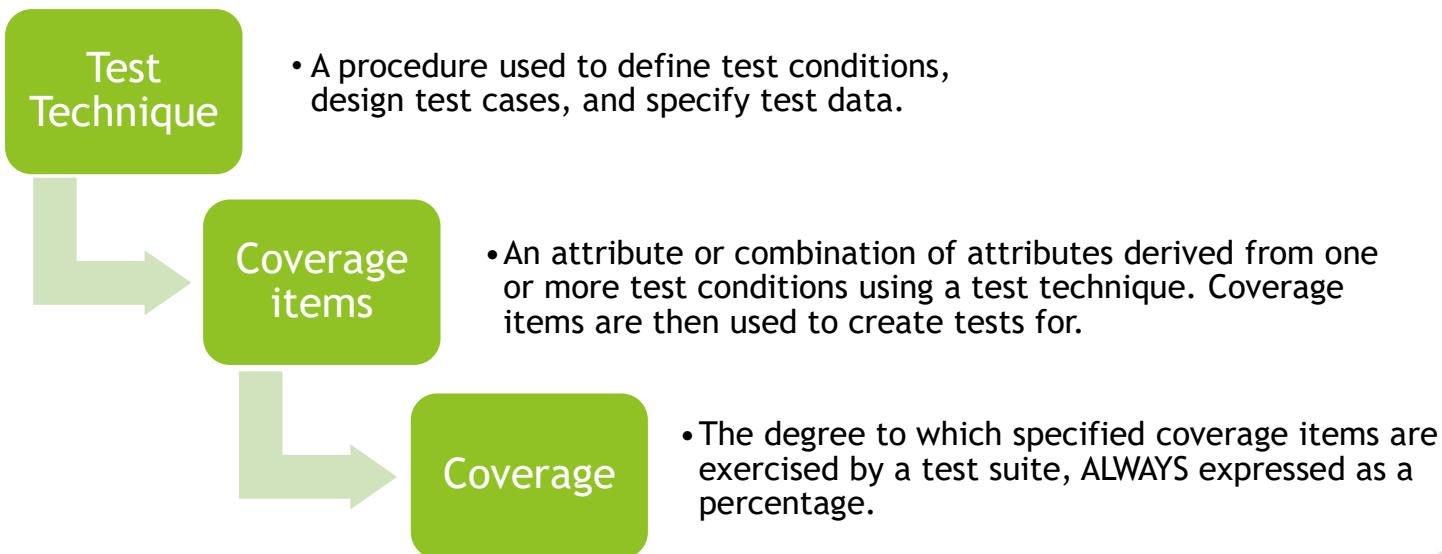




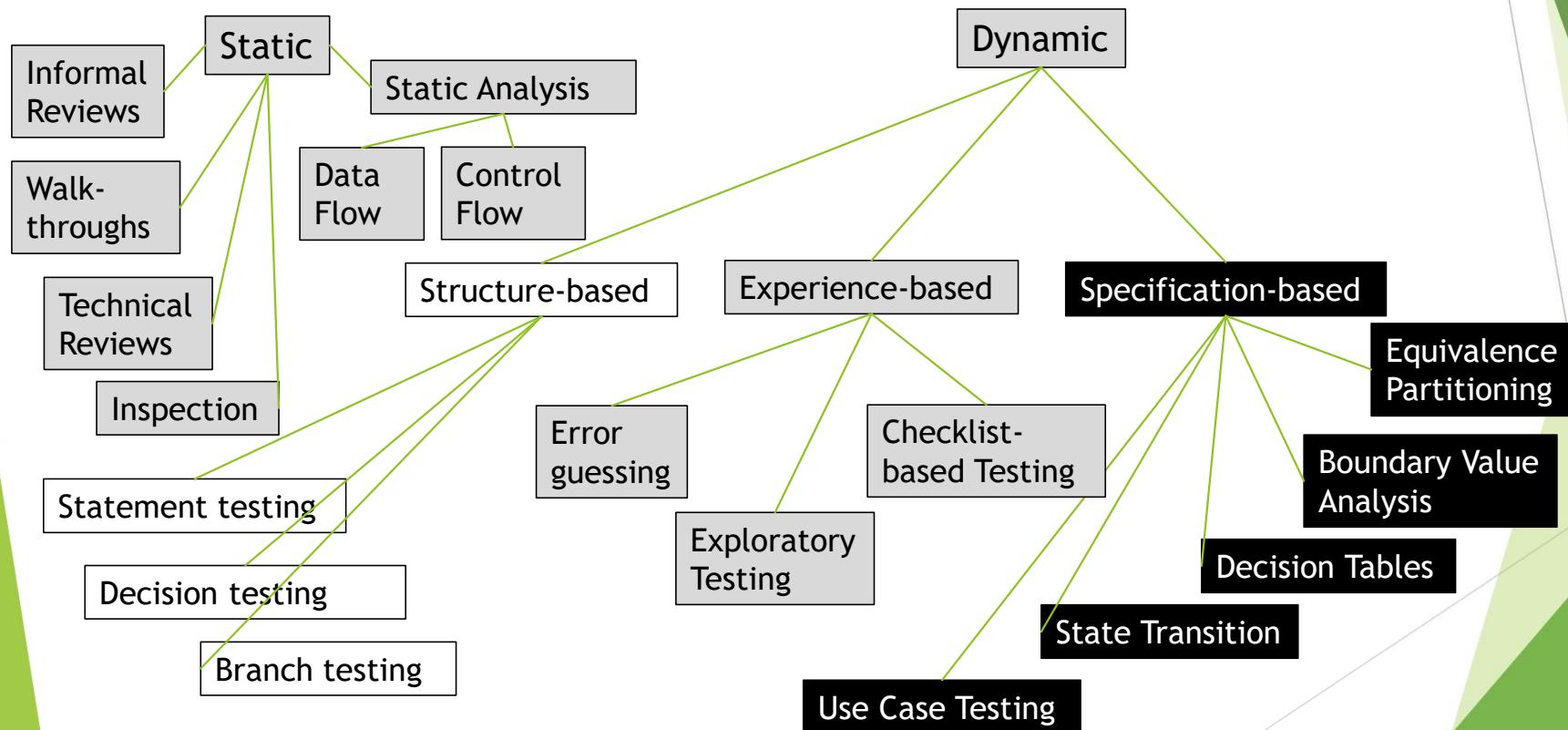
Test Analysis and Design

Summary - Keywords Explained

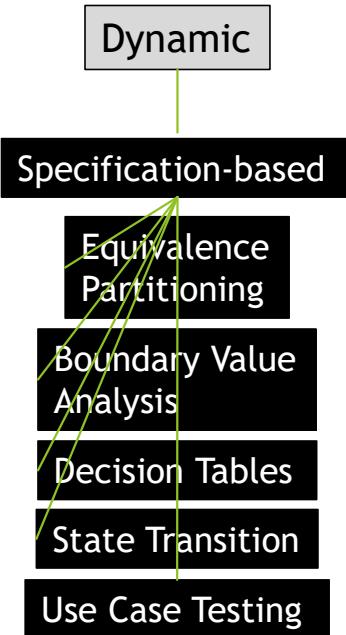
Test Analysis and Design Process



Test Techniques - visualized



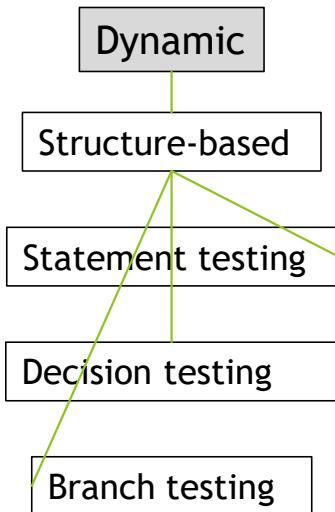
Black-Box test techniques



Several Black-Box (specification-based) test techniques have been discussed in this course:

- ▶ Equivalence partitioning
 - ▶ Test conditions are equivalence partitions exercised by one representative member of each partition.
- ▶ Boundary Value Analysis
 - ▶ Test cases are designed based on boundary values.
- ▶ Decision table testing
 - ▶ Test cases are designed to exercise the combinations of conditions and the resulting actions shown in a decision table.
- ▶ State transition testing
 - ▶ Test cases are designed to exercise elements of a state transition model.
- ▶ Use case testing
 - ▶ Test cases are designed to exercise use case behaviors.

White-Box test techniques

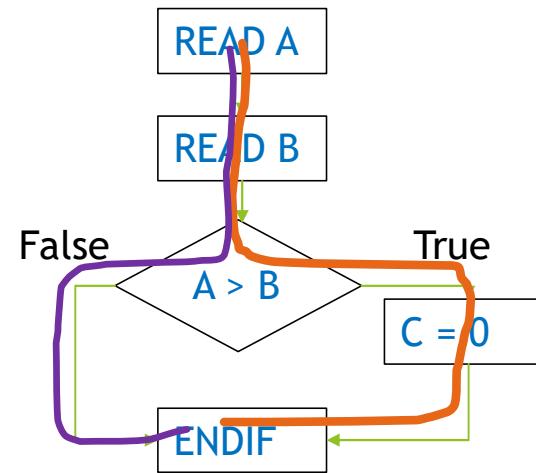


Several White-Box (structure-based) test techniques have been discussed in this course:

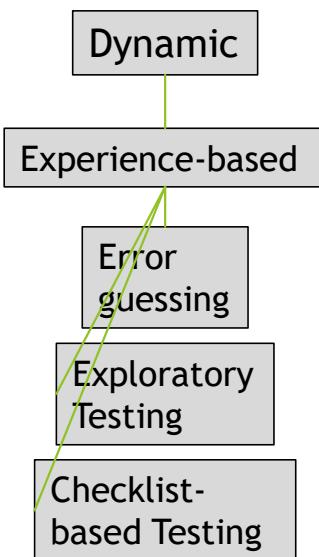
- ▶ Statement Testing
 - ▶ In which test cases are designed to execute statements
- ▶ Decision testing
 - ▶ In which test cases are designed to execute decision outcomes
- ▶ Branch testing
 - ▶ In which the test conditions are branches

Coverage in White-Box testing

- ▶ Statement Coverage
 - ▶ The coverage of executable statements.
- ▶ Decision Coverage
 - ▶ The coverage of decision outcomes.
- ▶ Branch Coverage
 - ▶ The coverage of branches in a control flow graph.



Experience-based test techniques



Several Experience-based test techniques have been discussed in this course:

- ▶ **Error Guessing**
 - ▶ A test technique in which tests are derived on the basis of the tester's knowledge of past failures, or general knowledge of failure modes.
- ▶ **Exploratory Testing**
 - ▶ An approach to testing in which the testers dynamically design and execute tests based on their knowledge, exploration of the test item and the results of previous tests.
- ▶ **Checklist Based testing**
 - ▶ An experience-based test technique in which test cases are designed to exercise the items of a checklist.

Collaboration-based Test Approaches



‘An approach to testing that focuses on defect avoidance by collaborating among stakeholders.’

Think for example of collaborative user story writing.

- ▶ **Acceptance Criteria** are defined which are
 - ▶ The criteria that a component or system must satisfy in order to be accepted by a user, customer, or other authorized entity.
- ▶ Acceptance criteria can be written in a lot of formats, the most common ones being:
 - ▶ Scenario-based or
 - ▶ Rule-based
- ▶ Acceptance criteria are used in **Acceptance-Test Driven Development** for the creation of Acceptance Test Cases.
- ▶ Acceptance Test Driven Development is a collaboration-based test-first approach that defines acceptance tests in the stakeholders' domain language.



Managing the Test Activities

Introduction

Managing the Test Activities

5.1 Test Planning

- ▶ FL-5.1.1 (K2) Exemplify the purpose and content of a test plan
- ▶ FL-5.1.2 (K1) Recognize how a tester adds value to iteration and release planning
- ▶ FL-5.1.3 (K2) Compare and contrast entry criteria and exit criteria
- ▶ FL-5.1.4 (K3) Use estimation techniques to calculate the required test effort
- ▶ FL-5.1.5 (K3) Apply test case prioritization
- ▶ FL-5.1.6 (K1) Recall the concepts of the test pyramid
- ▶ FL-5.1.7 (K2) Summarize the testing quadrants and their relationships with test levels and test types

5.2 Risk Management

- ▶ FL-5.2.1 (K1) Identify risk level by using risk likelihood and risk impact
- ▶ FL-5.2.2 (K2) Distinguish between project risks and product risks
- ▶ FL-5.2.3 (K2) Explain how product risk analysis may influence thoroughness and scope of testing
- ▶ FL-5.2.4 (K2) Explain what measures can be taken in response to analyzed product risks

Managing the Test Activities

5.3 Test Monitoring, Test Control and Test Completion

- ▶ FL-5.3.1 (K1) Recall metrics used for testing
- ▶ FL-5.3.2 (K2) Summarize the purposes, content, and audiences for test reports
- ▶ FL-5.3.3 (K2) Exemplify how to communicate the status of testing

5.4 Configuration Management

- ▶ FL-5.4.1 (K2) Summarize how configuration management supports testing

5.5 Defect Management

- ▶ FL-5.5.1 (K3) Prepare a defect report



Test Planning

Purpose & Content of a Test Plan

What is a Test Plan?

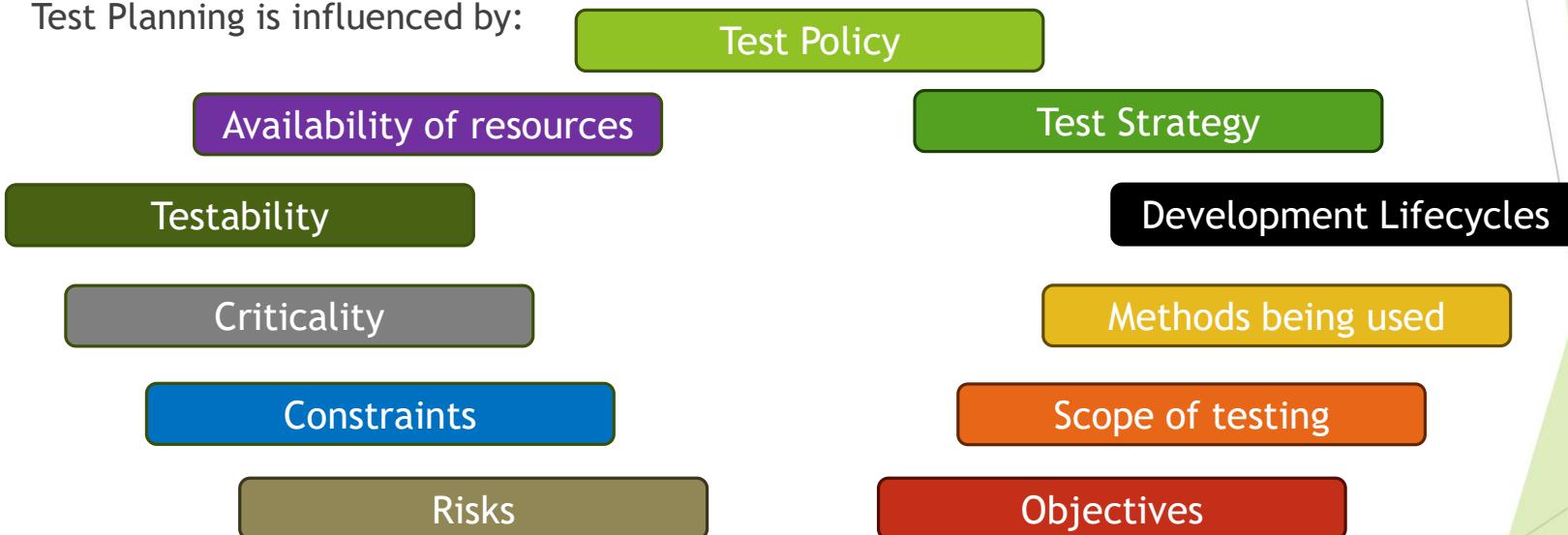


The purpose of a Test Plan

- ▶ Documents the means and the schedule for achieving the test objectives (how and when)
- ▶ Helps to ensure that the performed test activities will meet the established criteria
- ▶ Serves as a means of communication with team members and other stakeholders
- ▶ Demonstrates that testing will be in line with the test policy and test strategy (or explains where it will not be in line)

The Test Planning Process

Test Planning is influenced by:



The Test Planning Process (2)

- ▶ Typical planning activities are:
 - ▶ Determining the scope, objectives and risk of testing
 - ▶ Defining the overall approach of testing
 - ▶ Integrating and coordinating the testing activities into the SLC
 - ▶ Deciding on what, who, how, when
 - ▶ Scheduling test analysis, design, implementation, execution, evaluation and assigning resources
 - ▶ Selecting metrics for test monitoring and control
 - ▶ Budgeting for test activities
 - ▶ Determining the level of detail and structure for test documentation (templates, etc)

The benefits of Test Planning

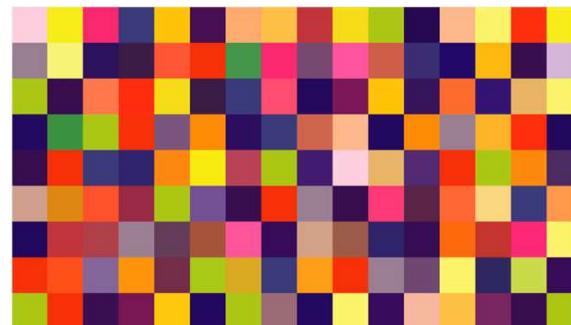
- ▶ Test Planning:
 - ▶ Guides the thinking process of the testers
 - ▶ Forces the testers to think about and confront future challenges



- ▶ The process of preparing a Test Plan helps in thinking through the efforts needed to achieve the objectives of the test project

Test Planning points of attention

- ▶ Test planning is a continuous activity and is performed throughout the product's lifecycle
- ▶ As the project progresses, more detail can be added into the test plan
- ▶ Feedback from test activities should be used to recognize changing risks and adjust the planning
- ▶ Planning may be documented in a master test plan and in separate test plans per test level or test type



Typical Contents of a Test Plan

- ▶ Context of testing:
 - ▶ Scope
 - ▶ Test objectives
 - ▶ Constraints
 - ▶ Test basis
- ▶ Assumptions and constraints
- ▶ Stakeholders
 - ▶ Roles
 - ▶ Responsibilities
 - ▶ Relevance to testing
 - ▶ Hiring and training needs
- ▶ Communication
 - ▶ Forms and frequency
 - ▶ Documentation templates
- ▶ Risk register
 - ▶ Project risks
 - ▶ Product risks
- ▶ Test approach
 - ▶ Test levels and test types
 - ▶ Test techniques
 - ▶ Test deliverables
 - ▶ Entry and Exit criteria
 - ▶ Independence of testing
- ▶ Metrics to be collected
- ▶ Test data requirements
- ▶ Test environment requirements
- ▶ Deviations from strategy or policy
- ▶ Budget and Schedule

Test Plan - ISO/IEC/IEEE 29119 Standard

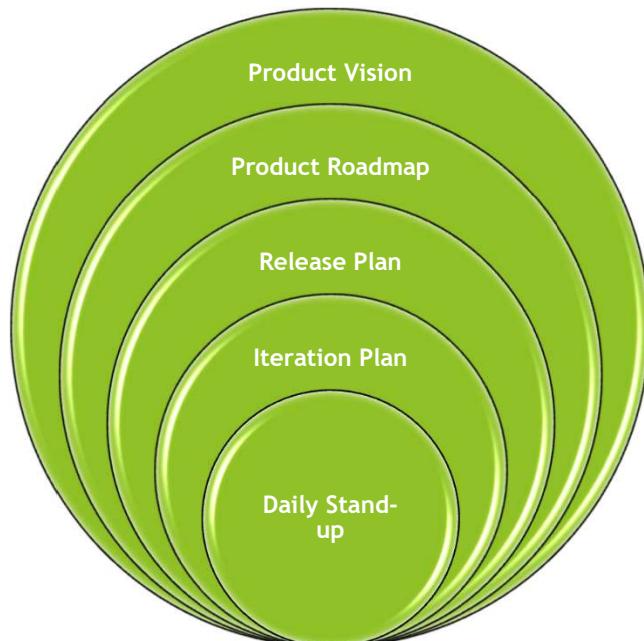
- ▶ Context of testing
 - ▶ Type of Test Plan
 - ▶ Test Items
 - ▶ Test Scope
 - ▶ Assumptions and Constraints
 - ▶ Stakeholders
- ▶ Testing Lines of Communication
- ▶ Risk Register (Risks)
- ▶ Testing activities and estimates
- ▶ Staffing
 - ▶ Roles, tasks, responsibilities
 - ▶ Training needs
 - ▶ Hiring needs
- ▶ Schedule
- ▶ Test Strategy (of the project)
 - ▶ Test sub-Processes
 - ▶ Test Deliverables
 - ▶ Test Design Techniques
 - ▶ Test completion criteria
 - ▶ Metrics to be collected
 - ▶ Test Data Requirements
 - ▶ Test Environment Requirements
 - ▶ Retesting and regression testing
 - ▶ Suspension and resumption criteria
 - ▶ Deviations from the organizational test strategy



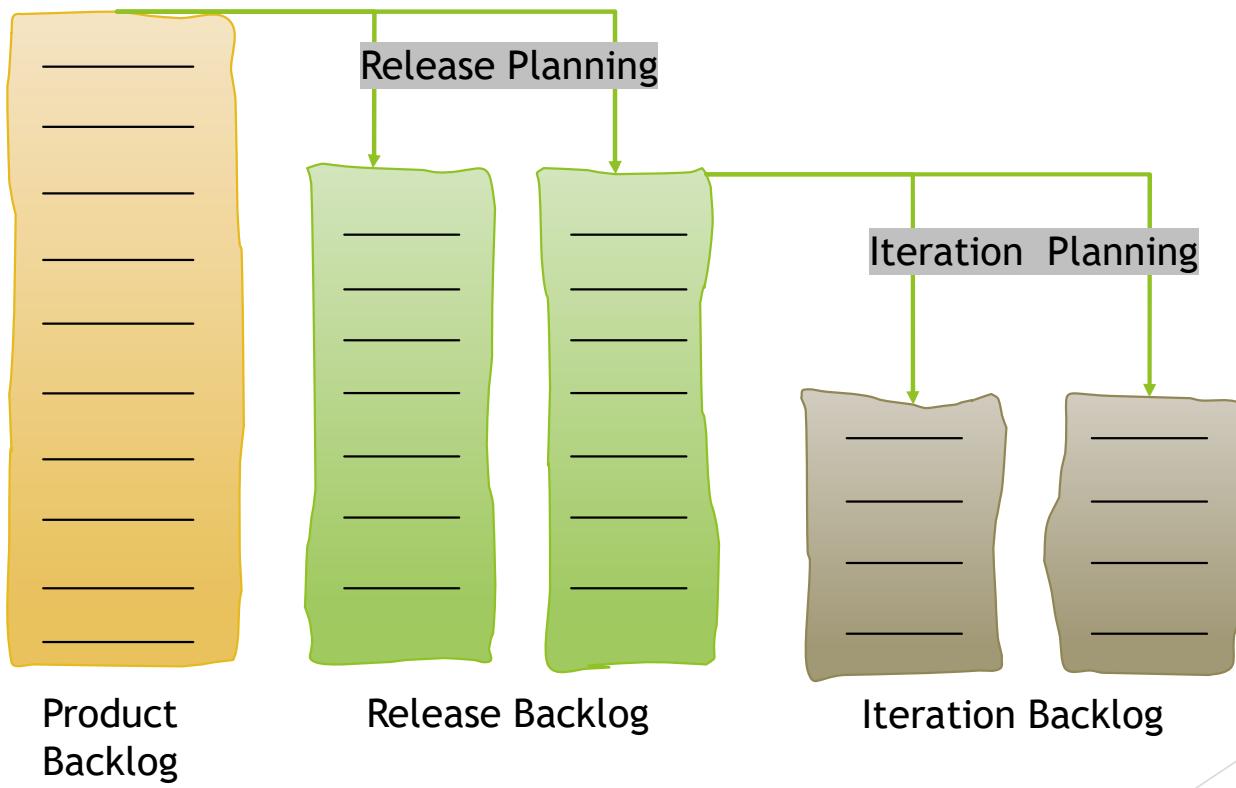
Tester's Contribution to Iteration and Release Planning

Release and Iteration Planning

- ▶ Planning is an ongoing activity, in whichever lifecycle
- ▶ In Iterative lifecycles, two kinds of planning occur:
 - ▶ Release Planning
 - ▶ Iteration Planning

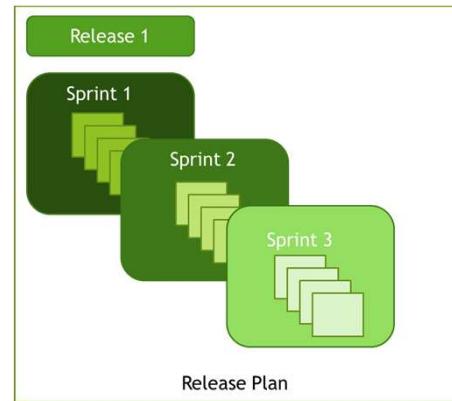


Release and Iteration Planning



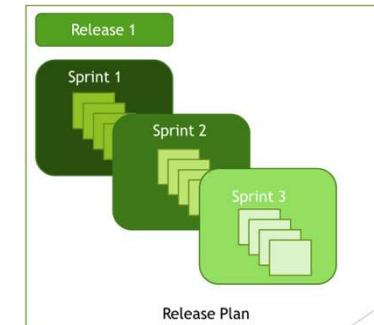
Release Planning

- ▶ Looks ahead to the release of the product
- ▶ Often several months ahead of the project-start
- ▶ Defines and redefines the product backlog
- ▶ May involve refining larger user stories into smaller ones
- ▶ Provides the basis for a test approach and test planning for all iterations
- ▶ Release plans are high level



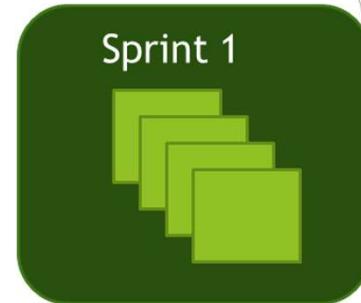
Release Planning - the process

- ▶ Business representatives, together with the team, establish and prioritize the user stories for the release
- ▶ Project and quality risks are identified and high-level effort estimation is performed
- ▶ Testers are involved in release planning and particularly add value in:
 - ▶ Writing testable user stories, including acceptance criteria
 - ▶ Participating in project and quality risk analyses
 - ▶ Estimating testing effort associated with the user stories
 - ▶ Determining the test approach
 - ▶ Planning the testing for the release



Iteration Planning

- ▶ Takes place before each iteration
- ▶ Looks ahead to the end of a single iteration
- ▶ Is concerned with the iteration backlog



Iteration Planning - the process

- ▶ Team selects user stories from the prioritized release backlog
- ▶ Team elaborates the user stories
- ▶ Risk analysis for the user stories is performed
- ▶ Work needed for each user story is estimated
- ▶ All needs to be clear to the team or the team can refuse to accept it and take another item based on the priority
- ▶ Business representatives should answer any questions of the team about the user stories to clarify:
 - ▶ What should be implemented
 - ▶ How it should be tested



Iteration Planning - The process (2)

- ▶ Number of stories selected is based on the story size and the velocity of the team
- ▶ After the iteration content is final, the stories are broken up into tasks
- ▶ Testers are involved in iteration planning and particularly add value in:
 - ▶ Participating in the detailed risk analysis of user stories
 - ▶ Determining the testability of the user stories
 - ▶ Creating acceptance tests for the user stories
 - ▶ Breaking down user stories into tasks (particularly testing tasks)
 - ▶ Estimating testing effort for all testing tasks
 - ▶ Identifying and refining functional and non-functional aspects of the test object



Test Planning

Entry & Exit Criteria

Entry vs Exit Criteria

Entry Criteria: 'define the preconditions for starting a certain activity.'

Aka '**Definition of Ready**'

- ▶ *Entry Criteria not being defined or not met will make the activity:*

More Difficult

Time
consuming

More Costly

Riskier

Exit Criteria: 'define what must be achieved to declare an activity completed'

Aka '**Definition of Done**'

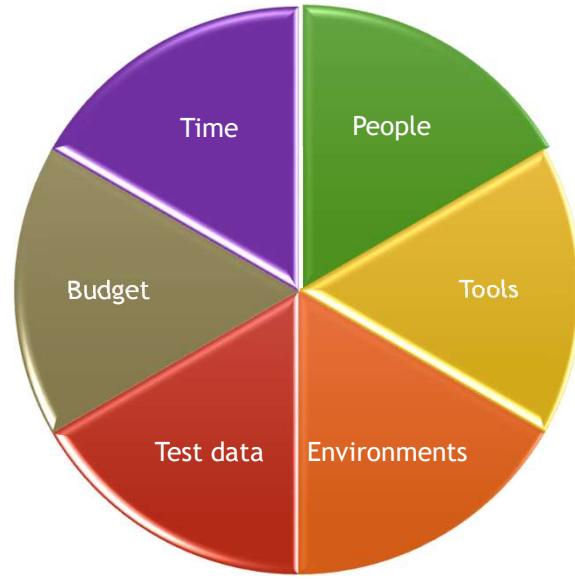
Both Entry and Exit criteria should be defined for:

- ▶ Each test level
- ▶ Each test type
- ▶ Differ based on the test objectives

Typical Entry Criteria (Definition of Ready)

When to safely start a test level or test phase?

- ▶ Availability of Resources
- ▶ Availability of Testware
- ▶ Initial quality level of the test object satisfied



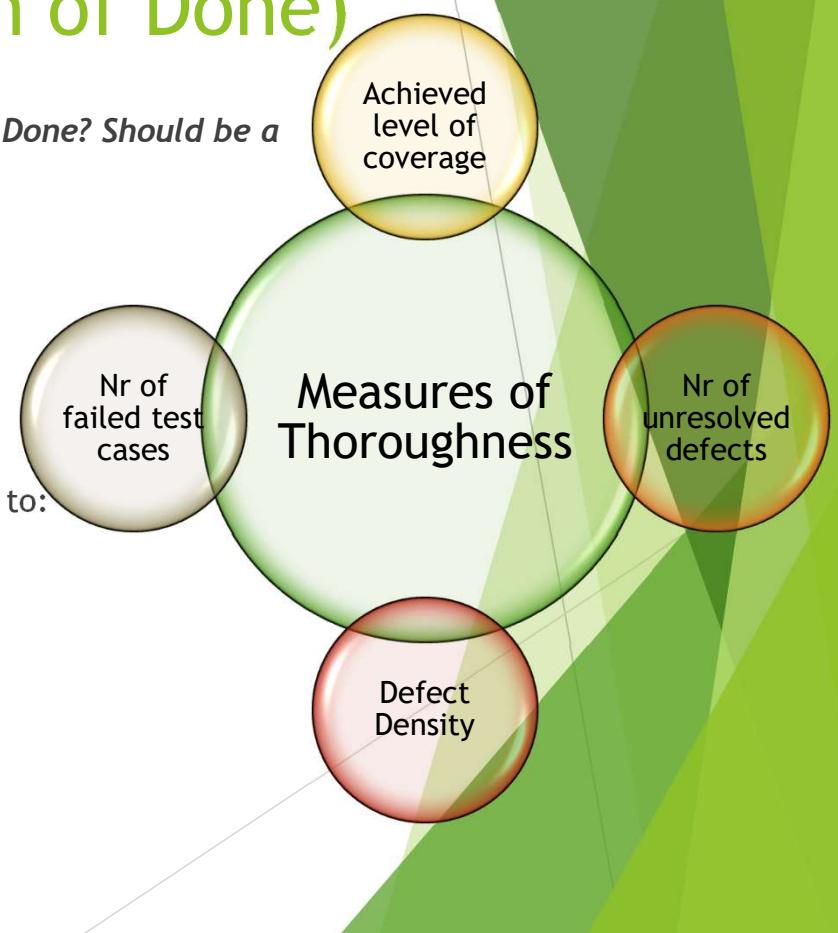
Typical Exit Criteria (Definition of Done)

When to safely stop testing, when to declare a certain test activity Done? Should be a balance of quality, budget, schedule and feature considerations.

- ▶ Measures of thoroughness
- ▶ Completion Criteria

Even without the exit criteria satisfied, test activities can also stop due to:

- ▶ Testing budget being expended
- ▶ Scheduled time being complete
- ▶ Pressure to bring the product to market



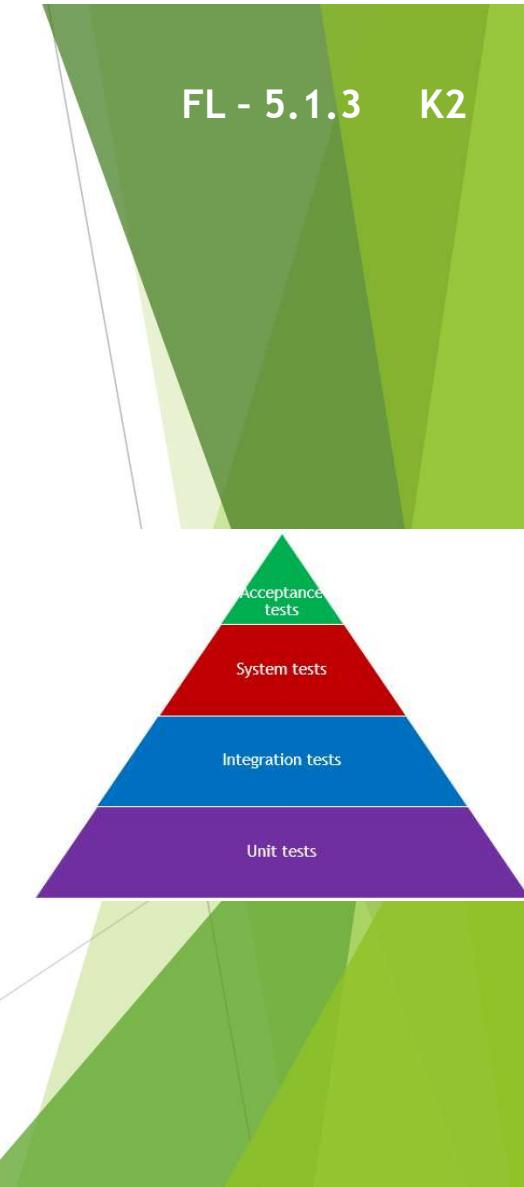
Benefits and pitfalls



- ▶ Checklist that guides discussion, estimation and design
- ▶ Limits the cost of rework
- ▶ Limits the risk of misunderstanding
- ▶ Can become the subject rather than a tool
- ▶ The list easily grows too big
- ▶ If not committed to it is not effective

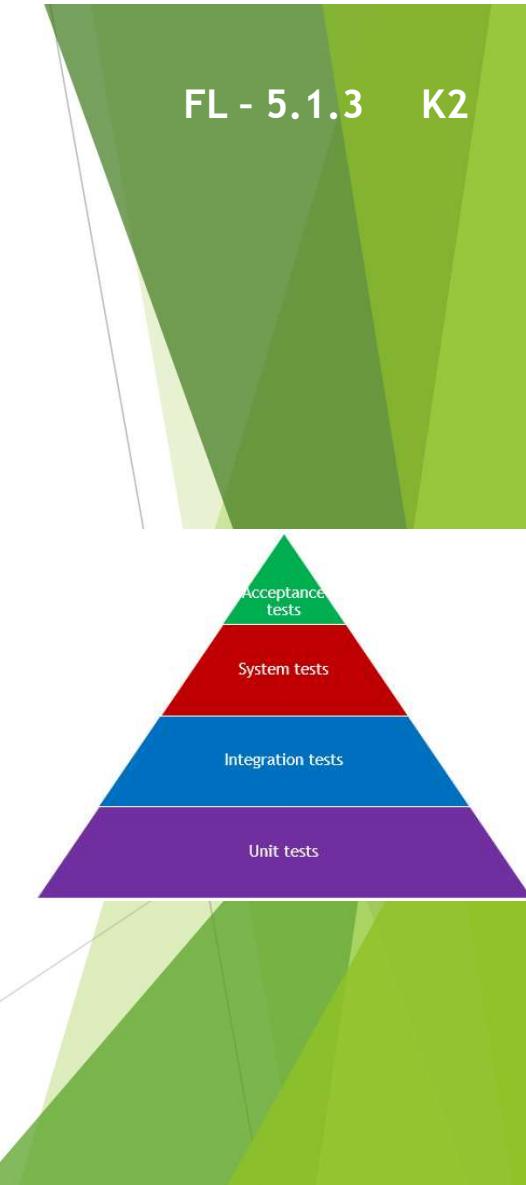
Definition of Done - Test Levels - Unit Testing

- ▶ 100% decision coverage where possible
- ▶ Static analysis performed on all code
- ▶ No unresolved major defects (ranked based on priority and severity)
- ▶ No known unacceptable technical debt remaining in the design and the code
- ▶ All code, unit tests, and unit test results reviewed
- ▶ All unit tests automated
- ▶ Important characteristics are within agreed limits (e.g., performance)



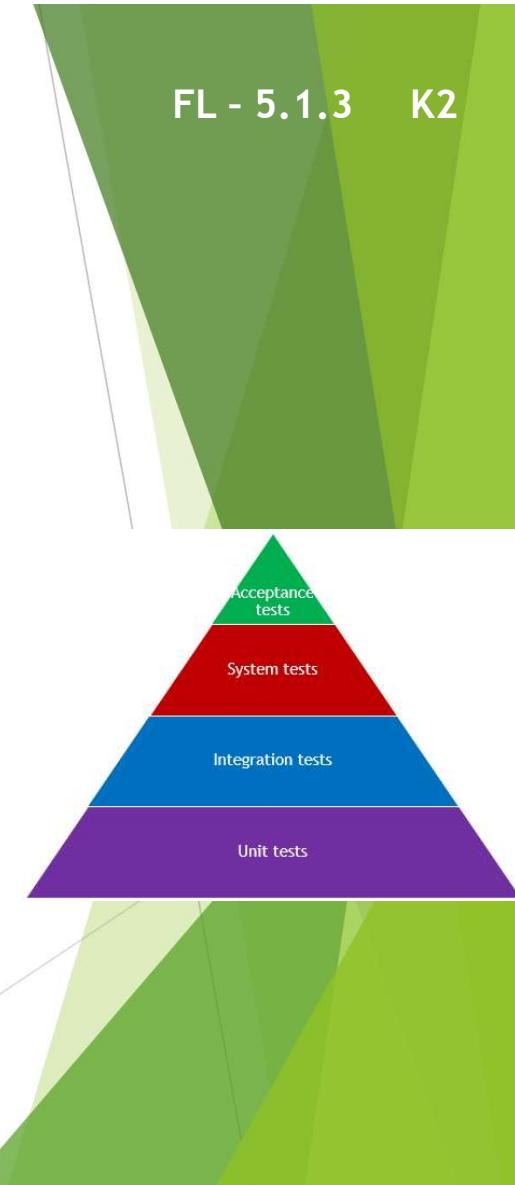
Definition of Done - Test Levels - Integration Testing

- ▶ All functional requirements tested, including both positive and negative tests, with the number of tests based on size, complexity, and risks
- ▶ All interfaces between units tested
- ▶ All quality risks covered according to the agreed extent of testing
- ▶ No unresolved major defects (prioritized according to risk and importance)
- ▶ All defects found are reported
- ▶ All regression tests automated, where possible, with all automated tests stored in a common repository



Definition of Done - Test Levels - System Testing

- ▶ End-to-end tests of user stories, features, and functions
- ▶ All user personas covered
- ▶ The most important quality characteristics of the system covered (e.g., performance, robustness, reliability)
- ▶ Testing done in a production-like environment(s)
- ▶ All quality risks covered according to the agreed extent of testing
- ▶ All regression tests automated, where possible, with all automated tests stored in a common repository
- ▶ All defects found are reported and possibly fixed
- ▶ No unresolved major defects (prioritized according to risk and importance)



Definition of Done - User Story

- ▶ The user stories selected for the iteration are complete, understood by the team, and have detailed, testable acceptance criteria
- ▶ All the elements of the user story are specified, reviewed (including the user story acceptance tests) and have been completed
- ▶ Tasks necessary to implement and test the selected user stories have been identified and estimated by the team



Definition of Done - Feature

- ▶ All user stories, with acceptance criteria, are defined and approved by the customer
- ▶ The design is complete, with no known technical debt
- ▶ The code is complete, with no known technical debt or unfinished refactoring
- ▶ Unit tests have been performed and have achieved the defined level of coverage
- ▶ Integration tests and system tests for the feature have been performed according to the defined coverage criteria
- ▶ No major defects remain to be corrected
- ▶ Feature documentation is complete, which may include release notes, user manuals, and on-line help functions

Definition of Done - Iteration

- ▶ All features for the iteration are ready and individually tested according to the feature level criteria
- ▶ Any non-critical defects that cannot be fixed within the constraints of the iteration added to the product backlog and prioritized
- ▶ Integration of all features for the iteration completed and tested
- ▶ Documentation written, reviewed, and approved

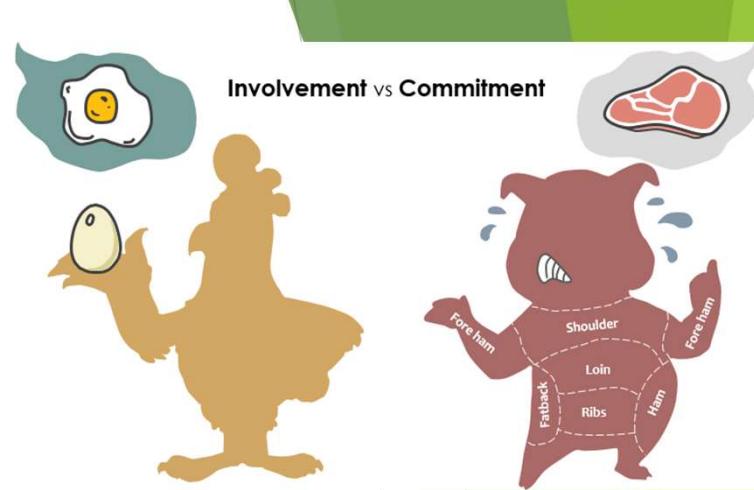
Now the software is potentially releasable



Definition of Done - Release

- ▶ Coverage
- ▶ Quality
- ▶ Time
- ▶ Cost





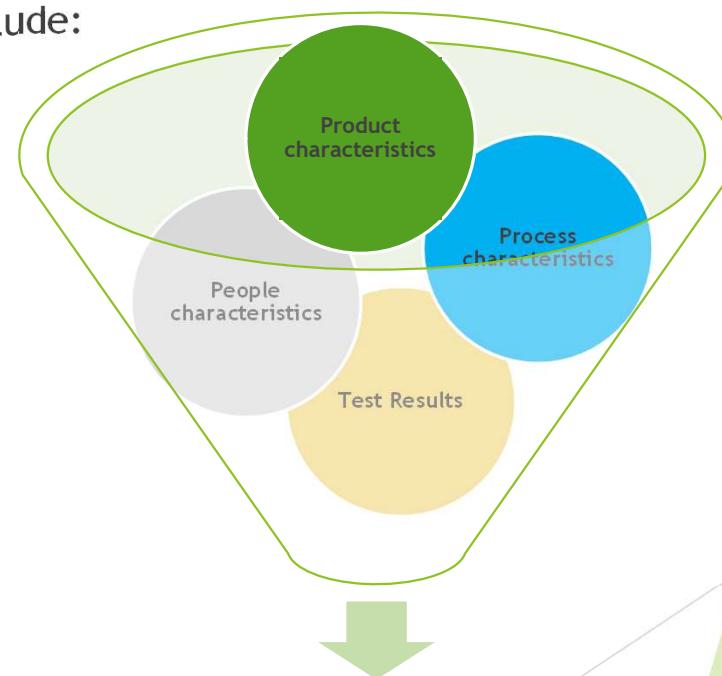
Estimating Test Effort

Test Effort Estimation

- ▶ Predicting the amount of test-related work needed to meet the test objectives for a:
 - ▶ Test Project
 - ▶ Release
 - ▶ Iteration
 - ▶ etc
- ▶ Stakeholders should know that:
 - ▶ The **estimate** is based on a nr of assumptions and is subject to estimation errors
 - ▶ Estimating for **small** tasks is more accurate than for large tasks
 - ▶ Too large tasks should be **broken up** into smaller tasks, that can be estimated more accurately

Test Effort Estimation - influencing factors

- ▶ Factors that influence the test effort include:
 - ▶ Product characteristics
 - ▶ Development process characteristics
 - ▶ People characteristics
 - ▶ Test results



Test Effort

Product Characteristics influencing test effort

- ▶ Risks associated with the product
- ▶ Quality of the test basis
- ▶ Size of the product
- ▶ Complexity of the product domain
- ▶ Requirements for quality characteristics (performance, usability, etc)
- ▶ Required level of detail for test documentation
- ▶ Requirements for legal and regulatory requirements



Development Process Characteristics influencing test effort

- ▶ Stability and maturity of the organization
- ▶ Development model in use
- ▶ Test approach
- ▶ Tools used
- ▶ Test process
- ▶ Time pressure



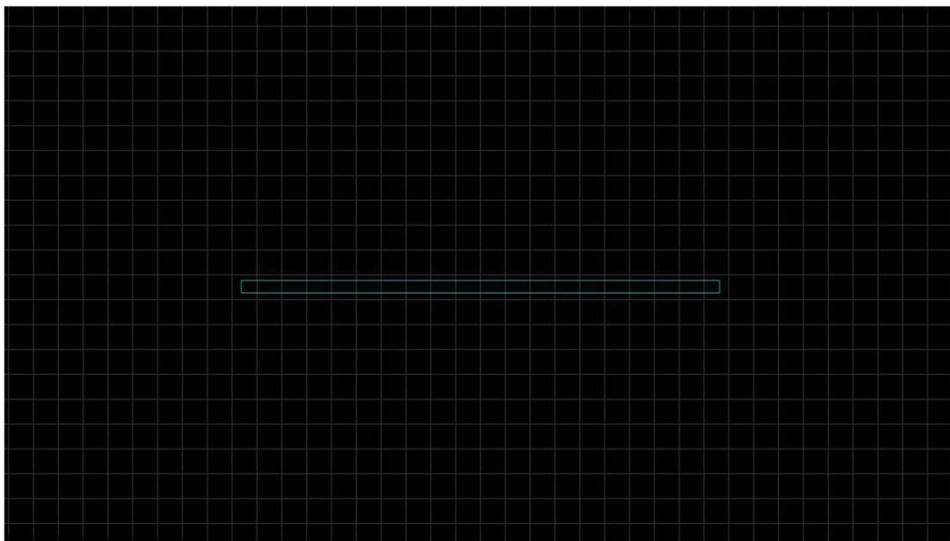
People Characteristics influencing test effort

- ▶ Skills of the people involved
- ▶ Experience of the people involved (domain knowledge)
- ▶ Team cohesion and leadership



Test Results influencing test effort

- ▶ Number and severity of defects found
- ▶ Amount of rework required



Test estimation categories

- ▶ **Expert-based approach** - estimating based on the experience of the owners of the testing tasks or by the experts and experienced staff members. Examples here are:
 - ▶ Wideband Delphi estimation
 - ▶ Planning poker in Agile
- ▶ **Metrics-based approach** - estimating based on metrics of former or similar projects or based on typical values:
 - ▶ Dev-QA ratio
 - ▶ Project size and complexity
 - ▶ Defect removal models
 - ▶ Burndown charts and velocity



Test estimation Techniques

- ▶ Estimation based on ratios (metrics-based)
- ▶ Extrapolation (metrics based)
- ▶ Wideband Delphi (iterative, expert-based)
- ▶ Three-point estimation (expert-based)





Test Estimation based on Ratios

Estimation based on ratios

- ▶ Metrics-based technique
- ▶ Based on metrics from previous projects in the organization
- ▶ Ratios from the own organization are the best source in the estimation process as it:
 - ▶ Takes into account the development process characteristics
 - ▶ Takes into account the people characteristics
- ▶ Use these metrics to calculate ‘standard’ ratios for similar projects
- ▶ Use this standard ratio to estimate the test effort for the new project

Estimation based on ratios - Example

- ▶ A standard calculated ratio is calculated from past similar projects
- ▶ Development-to-test-effort was 5:3
- ▶ Current estimated development effort is 1000 person-days
- ▶ How much is the test effort?
 - ▶ $1000 / 5 = 200$
 - ▶ $200 * 3 = 600$
 - ▶ The test effort is estimated to be 600 person-days





Test Estimation - Extrapolation

Extrapolation



‘The action of estimating or concluding something by assuming that existing trends will continue or a current method will remain applicable.’

- ▶ Metrics-based technique
- ▶ Measuring and gathering data as **early** as possible in the **current** project
- ▶ When there are enough measurements, effort for the remaining work can be estimated by extrapolating the data
- ▶ Very suitable for iterative SDLC models

FL - 5.1.4 K3

Estimation based on ratios - Example

- ▶ A project has started 3 sprints ago; each sprint taking 2 weeks
- ▶ The velocity (delivery capacity) in story points of the team has been the below, divided into development and test:

	Sprint 1	Sprint 2	Sprint 3	Sprint 4?
Dev	13	18	17	16
Test	3	6	6	5

- ▶ Based on this, calculate the dev AND test velocity for the next sprint using extrapolation
 - ▶ Dev: $(13+18+17)/3 = 16$ story points
 - ▶ Test: $(3+6+6)/3 = 5$ story points

Estimation based on ratios - Example 2

- ▶ Let's say the high-level estimate is that in these 3 sprints about 10% of the application is done, how long do you estimate the entire project to take?

	Sprint 1	Sprint 2	Sprint 3	Sprint N
Dev	13	18	17	16
Test	3	6	6	5

- ▶ 10% of the application has taken 63 story points
- ▶ 100% of the application is estimated on 630 story points
- ▶ 90% is still to be done, estimated to take $((630 / 100) * 90\%) = 567$ story point
- ▶ 567 story points is estimated to take $(567 / 21 \text{ story points per sprint}) = 27$ sprints, so 54 weeks



Test Estimation - Wideband Delphi

Wideband Delphi

- ▶ Iterative, expert-based technique
- ▶ Experts make experience-based estimations
- ▶ Each expert estimates the effort in isolation
- ▶ Results are collected and if there are unacceptable deviations the estimates are discussed
- ▶ After the discussion each expert (in isolation) should make a new estimate based on the feedback
- ▶ This is repeated until there is consensus

Estimating Testing Effort - Planning Poker explained

- ▶ Takes place in Release Planning
- ▶ Feature is discussed by the team (developers, testers, business)
- ▶ Business answers questions from the team
- ▶ For estimation it is great there is availability of:
 - ▶ Risk Level
 - ▶ Priority
- ▶ In Planning poker,
 - ▶ Business representative presents a user story
 - ▶ After all questions and discussions, the estimation can begin
 - ▶ Each estimator (team member) privately selects a card to represent the estimate
 - ▶ All cards are revealed at the same time
 - ▶ If all have the same, that becomes the estimate, otherwise the poker continues until agreement is reached - this should be done by communication - so discuss the lowest and highest effort given and try to see whether some people in the team missed something or everyone else missed something.



Estimating Testing Effort - Planning Poker explained (2)

- ▶ Each estimator has a deck of cards with values similar to the Fibonacci sequence (0,1,2,3,5,8,13,21,34,55,89...), OR shirt sizes, OR jelly beans, OR.....
- ▶ Fibonacci - recommended as uncertainty grows exponentially with the size of the story
- ▶ Values represent story points, effort days or other units in which the team estimates
- ▶ Estimate too high, story unclear or needs to be broken down into smaller stories
- ▶ Agreement can be reached by:
 - ▶ Consensus
 - ▶ Applying rules (use highest, average, etc)

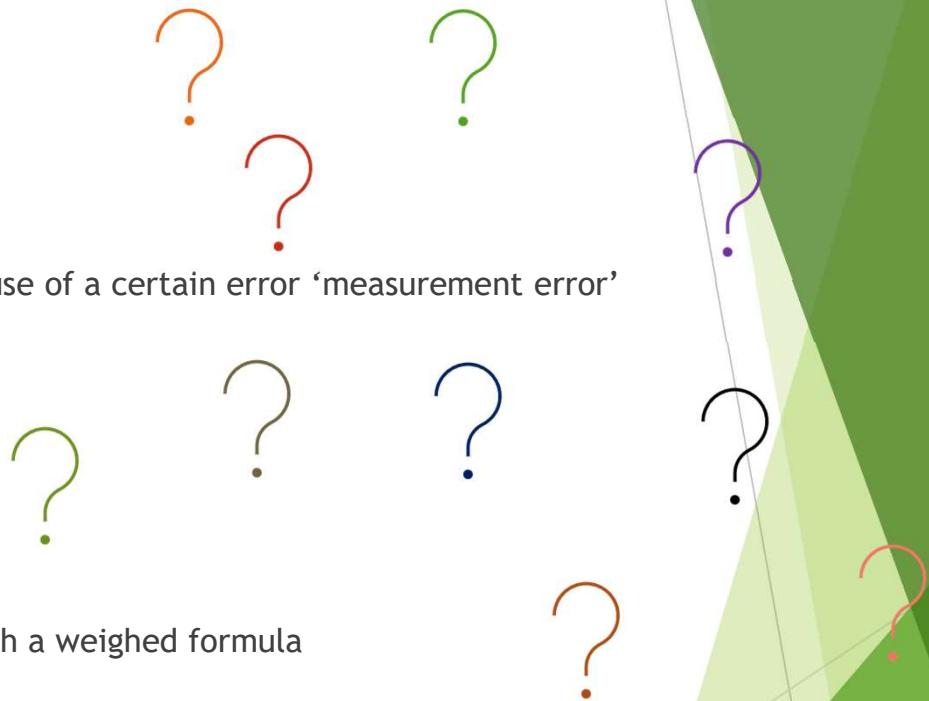




Test Estimation - Three-point Estimation

Three-point estimation

- ▶ Expert-based technique
- ▶ The outcome is an estimate that lies in a range because of a certain error ‘measurement error’
- ▶ Three estimates are made by experts
 - ▶ The most optimistic estimation
 - ▶ The most likely estimation
 - ▶ The most pessimistic estimation
- ▶ Calculate the estimate of these three estimations with a weighed formula
- ▶ Calculate the standard deviation, or measurement error
- ▶ The result is the range between the calculated estimate minus the deviation and the calculated estimate plus the deviation



Three-point estimation - Example

- ▶ Three estimates are made by experts
 - ▶ The most optimistic estimation (a) is 12 person-days
 - ▶ The most likely estimation (m) is 14 person-days
 - ▶ The most pessimistic estimation (b) is 18 person-days
- ▶ The estimate (E) is calculated as ' $E = (a+(4*m)+b) / 6$ '
- ▶ The most likely estimate is weighed 4 times and the others only once.
- ▶ The estimate E is $(12+(4*14)+18) / 6 = 88 / 6 = \textcolor{red}{14.33}$
- ▶ The standard deviation or measurement error (SD) is calculated as $SD = (b-a) / 6$
- ▶ $SD = (18-12) / 6 = \textcolor{red}{1}$
- ▶ Meaning the final estimate in person days is between $(14.33-1=)$ **13.33** and $(14.33+1=)$ **15.33**

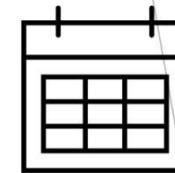




Test Case prioritization

Test Execution Schedule

- ▶ Test cases and test procedures have been created and assembled into test suites
- ▶ Test suites are then arranged in a test execution schedule
- ▶ Test execution schedule should take into account priority, dependencies, confirmation tests, regression tests, the most efficient sequence and available resources
- ▶ Ideally the sequencing is done only on priority but this is not possible if there are dependencies:
 - ▶ On lower priority test cases
 - ▶ Across test cases
- ▶ Confirmation and regression tests should also be prioritized based on importance of rapid feedback, but dependencies may apply
- ▶ Various sequences of tests can have different levels of efficiency, so trade-offs between efficiency and prioritization are needed



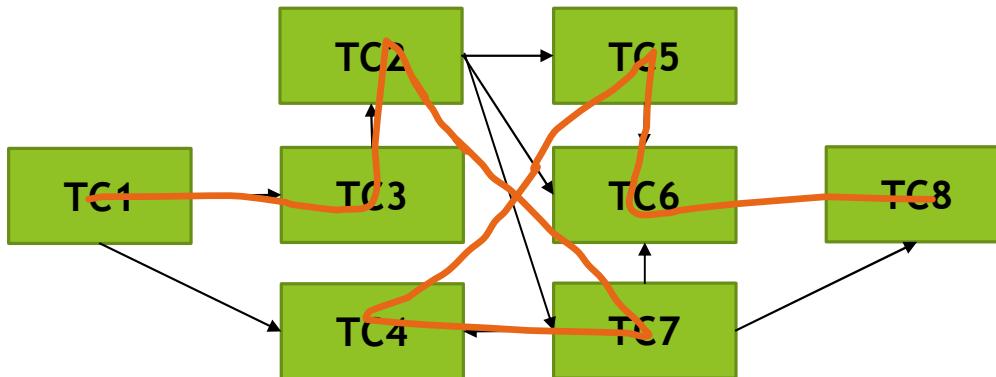
How to Prioritize Test Cases

- ▶ Risk-based prioritization
 - ▶ Based on the results from Risk Analysis
- ▶ Coverage-based prioritization
 - ▶ Based on the coverage
 - ▶ Test cases achieving the highest coverage are executed first
 - ▶ There are different variants, one being ‘Additional Coverage Prioritization’
 - ▶ First execute the test case achieving the highest coverage
 - ▶ Each test after that is the one that achieves the highest additional coverage
- ▶ Requirements-based prioritization
 - ▶ Based on the priority of the **requirements**
 - ▶ Test cases covering the highest priority requirements are executed first



Test Execution Schedule - Example

- ▶ There are 8 test cases, TC1 has the highest priority, TC8 the lowest, with their dependencies (logical or technical)
- ▶ Arrow from TC1 to TC4 means that TC4 is dependent on TC1
- ▶ Please create the most efficient test execution schedule while taking the dependencies into account

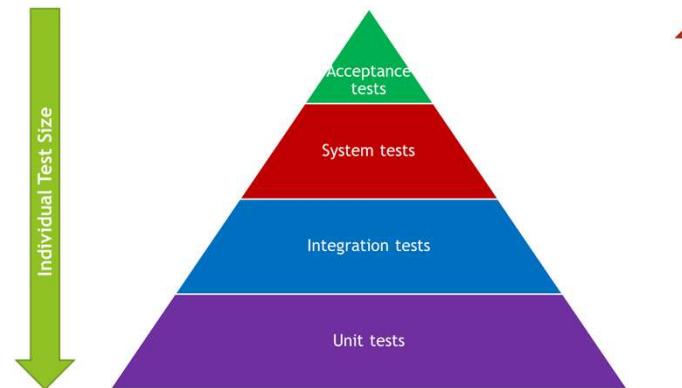




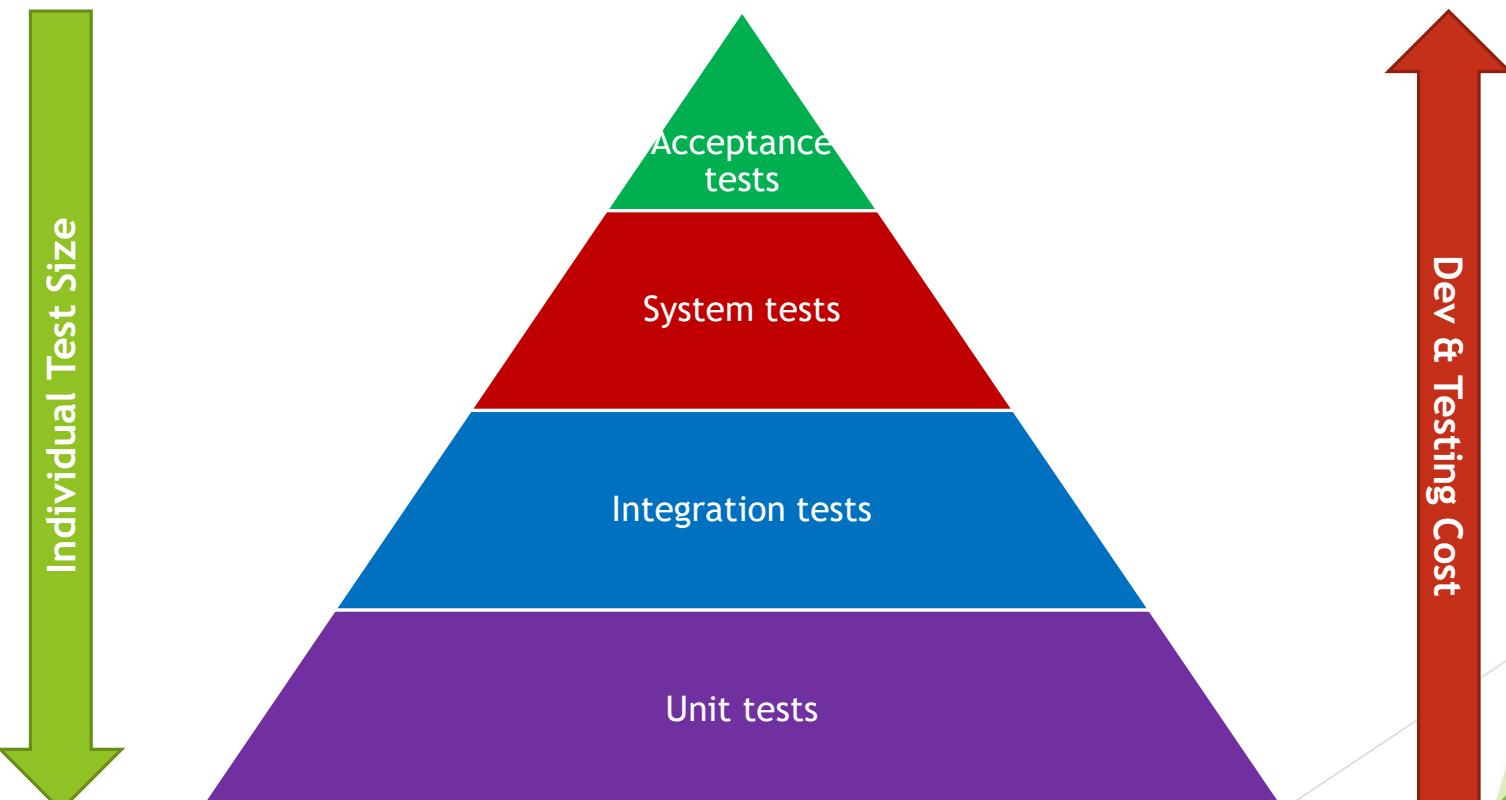
The Test Pyramid

The Test Pyramid

- ▶ Layers of the pyramid represent groups of tests
- ▶ Different Goals are supported by different levels of testing and test automation
- ▶ Supports the team in:
 - ▶ Test Automation
 - ▶ Test Effort Allocation
- ▶ The higher the layer
 - ▶ The lower the test granularity
 - ▶ The lower the test isolation - the larger the test
 - ▶ The higher the test execution time

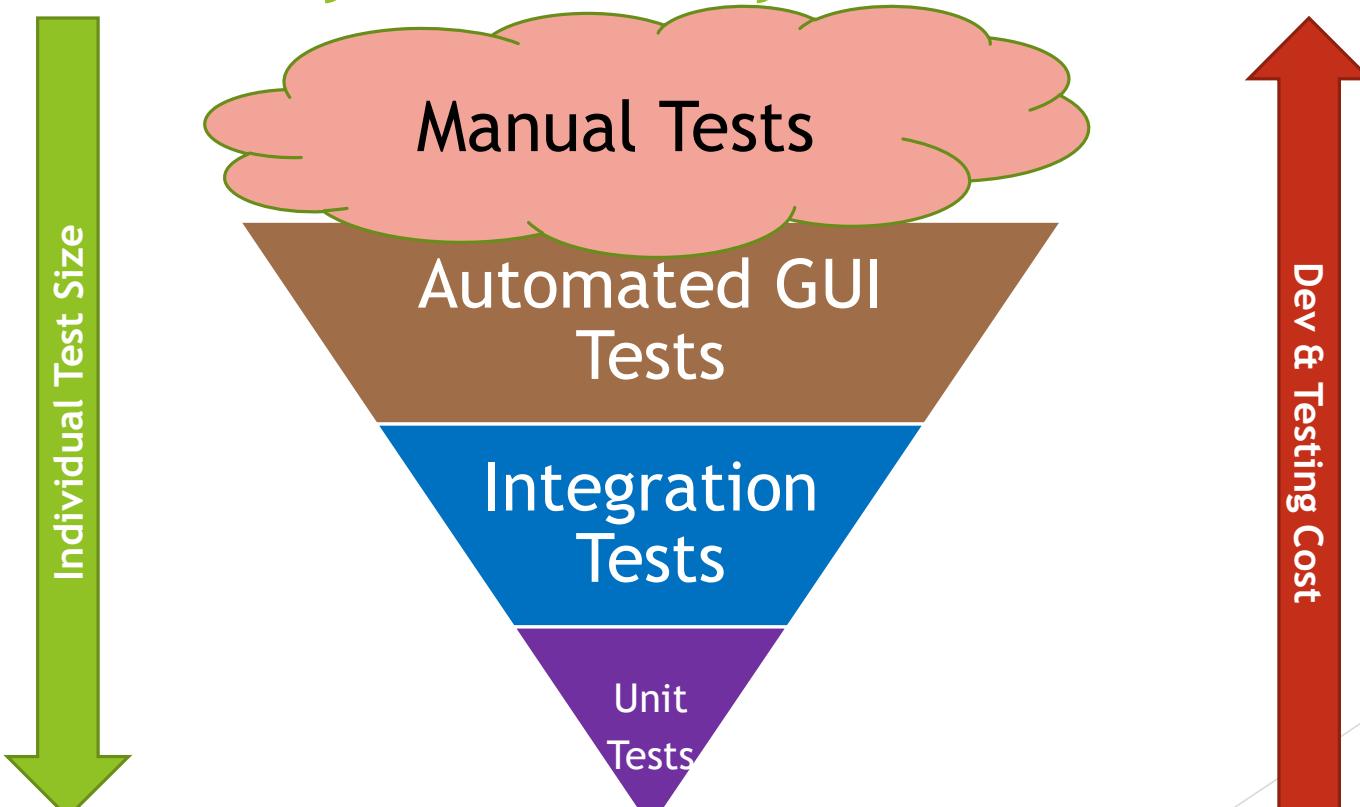


The Test Pyramid



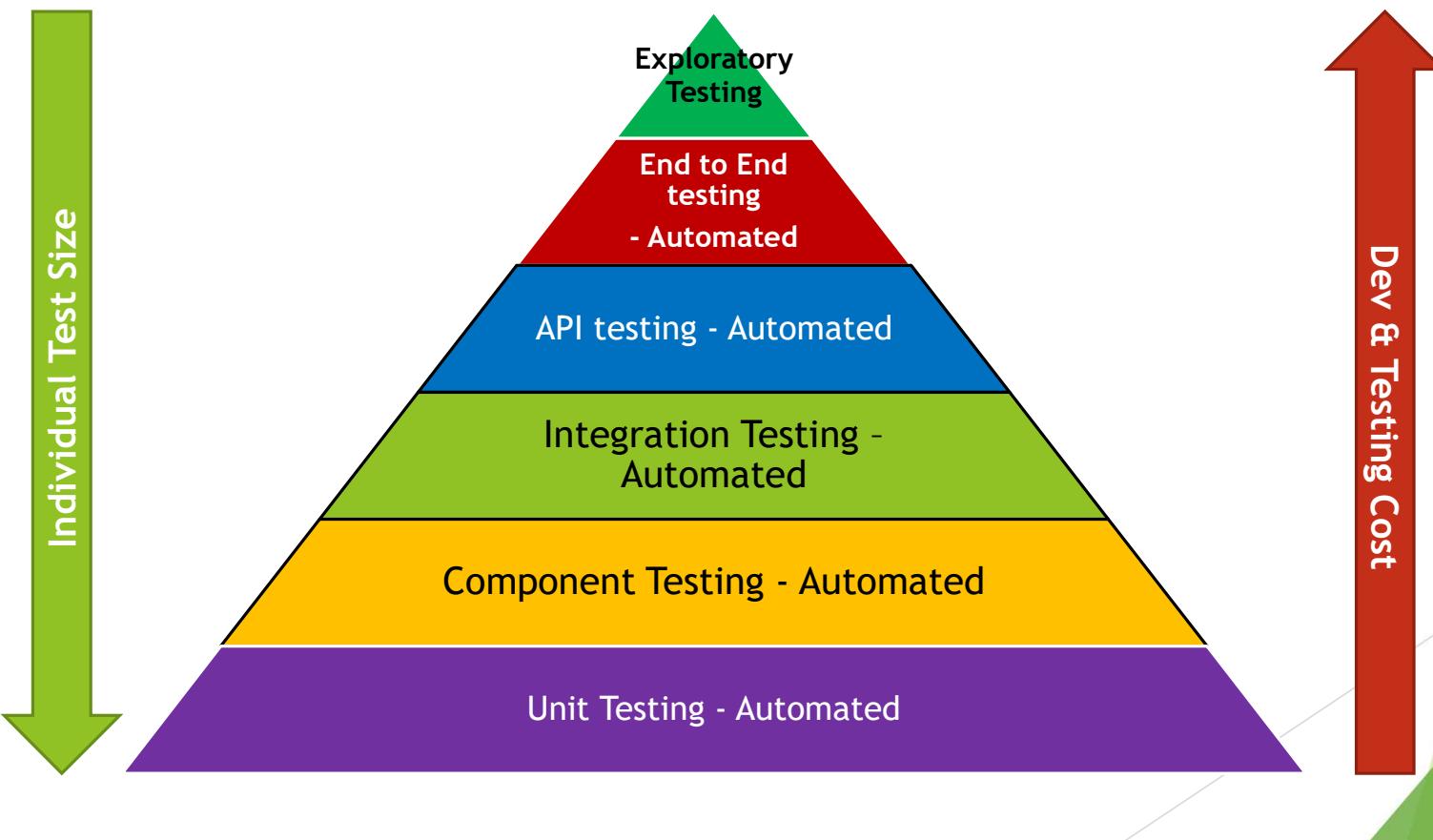
FL - 5.1.6 K1

The Test Pyramid - Why?



FL - 5.1.6 K1

The Test Pyramid - Keep thinking!!!





Testing Quadrants

Testing Quadrants

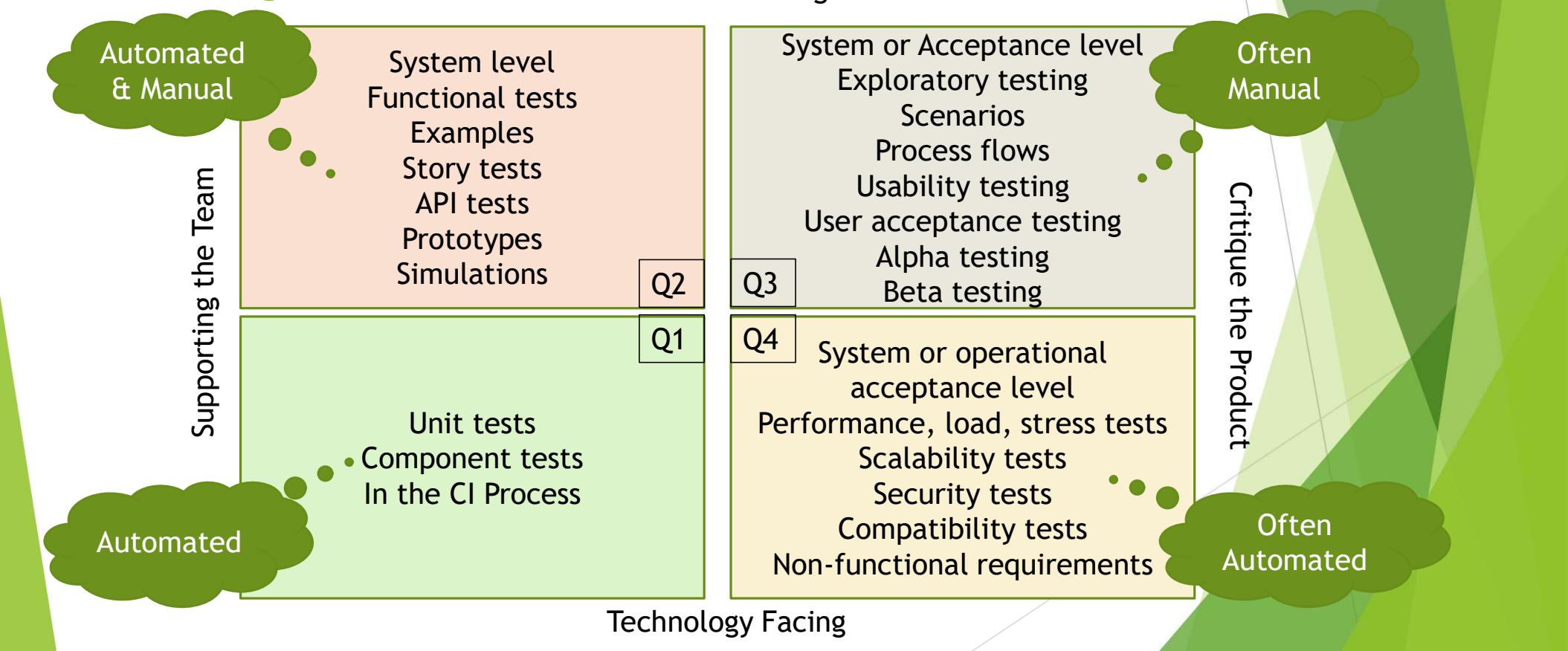
- ▶ Group the test levels with the appropriate:
 - ▶ Test Types
 - ▶ Test Levels
 - ▶ Activities
 - ▶ Test techniques
 - ▶ Work Products
- ▶ Support test management with visualization to:
 - ▶ Ensure that the appropriate test types and test levels are included in the SDLC
 - ▶ Understand that some test types are more relevant to certain test levels than others
- ▶ Provide a way to differentiate and describe the types of tests to stakeholders

Developers

Testers

Business
Stakeholders

Testing Quadrants

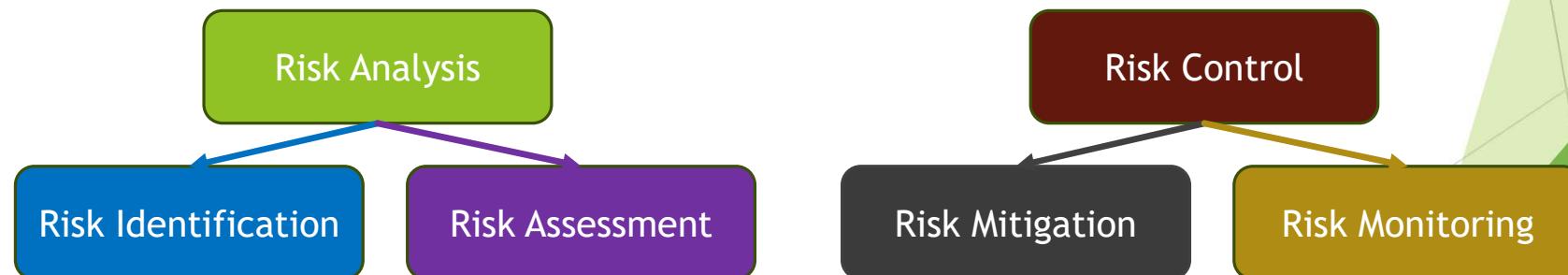




Risk Management - Introduction

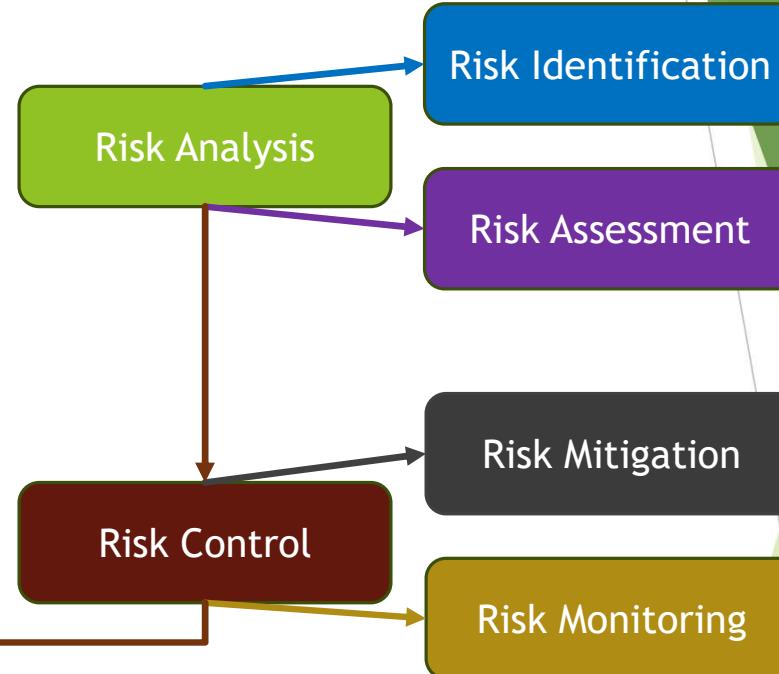
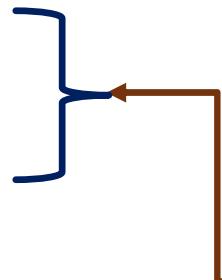
Risk Management

- ▶ Allows an organization to:
 - ▶ Increase the likelihood of achieving objectives
 - ▶ Improve the quality of the products
 - ▶ Increase stakeholders trust and confidence
- ▶ Main risk management activities are:



Risk-based Testing

- ▶ Test approach
- ▶ Risk Analysis and Risk Control are used to:
 - ▶ Select test activities
 - ▶ Prioritize test activities
 - ▶ Manage test activities





Risk Definition and Risk Attributes

What is risk?

'A potential event, hazard, threat or situation whose occurrence causes an adverse effect'

Risk is characterized by two factors:

- ▶ Risk likelihood
 - ▶ 'The probability of the risk occurrence' (greater than zero and less than 1)
- ▶ Risk impact
 - ▶ 'The consequences of this occurrence' - the harm

Risk Level

- ▶ A measure for the risk
 - ▶ The higher the risk level, the more important the treatment
- ▶ A calculated result from the combination of:
 - ▶ Risk Likelihood (greater than zero and smaller than one)
 - ▶ Risk Impact



Risk Likelihood

'The probability of the risk occurrence'

- ▶ *Factors that influence the likelihood include*

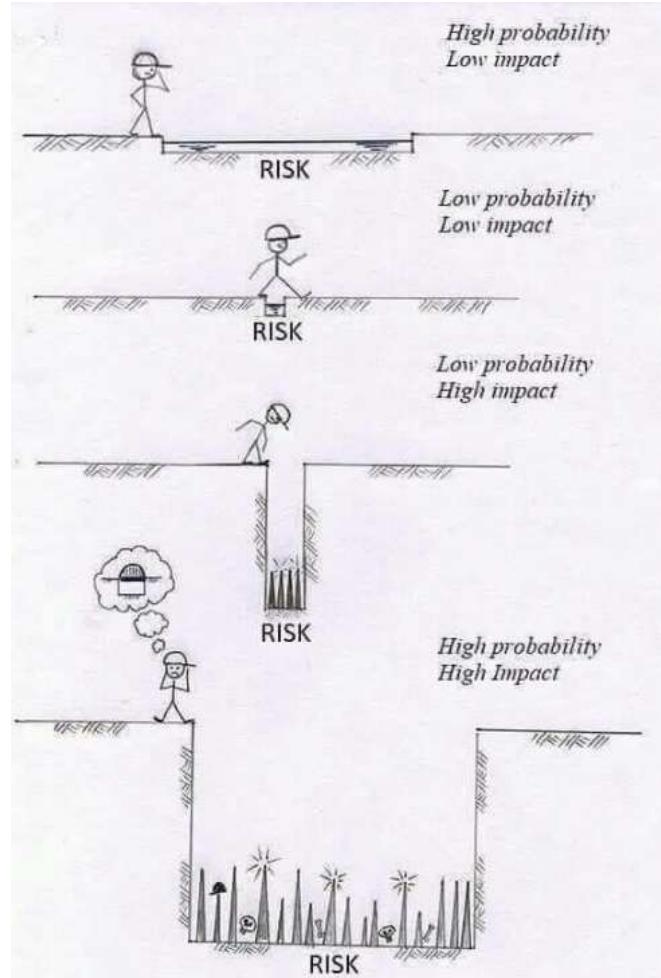
Complexity	Weak leadership
Personnel and training issues	Pressure
Conflict within the team	Lack of earlier quality assurance activities
Contractual problems with suppliers	High change rates
Geographically distributed team	High earlier defect rates
Legacy vs new approaches	Interfacing and integration issues
Tools and technology	

Risk Impact

‘The consequences of this occurrence’

► *Factors that influence the impact include*

Frequency of use of the affected feature	Civil or criminal legal sanctions
Criticality of the feature for a business goal	Loss of license
Damage to reputation	Lack of reasonable workarounds
Loss of business	Visibility of failure leading to negative publicity
Potential losses or liability	Safety





Project Risks and Product Risks

Project Risk

- ▶ Are related to the Management and Control of the project
 - ▶ If project risks occur, this may have a negative effect on
 - ▶ Project Schedule
 - ▶ Budget
 - ▶ Scope
- Affects the project's ability to achieve its objectives.



Project Risk - Examples

- ▶ **Organizational issues**
 - ▶ Delays in work product deliveries, inaccurate estimates, cost-cutting, etc
- ▶ **People issues**
 - ▶ Insufficient skill, training needs, staff shortage, personnel issues, conflicts, communication issues, etc
- ▶ **Technical issues**
 - ▶ Problems in defining the right requirements, requirements not met due to constraints, scope creep (increasing requirements), poor tool support, etc
- ▶ **Supplier issues**
 - ▶ Delivery failure of a third party, bankruptcy of a third party, contractual issues, etc

Product Risk

- ▶ The possibility that the system or software might fail to satisfy the legitimate needs of the users and/or stakeholders
- ▶ Related to the product quality characteristics - A risk to the **quality** of the product

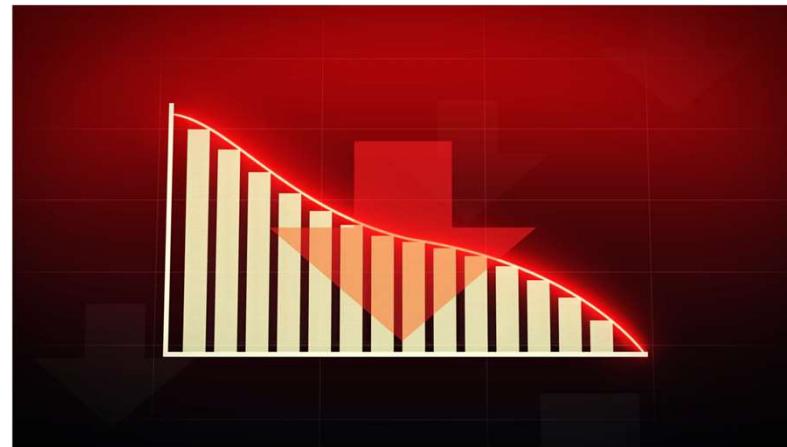
Product Risk



FL - 5.2.2 K2

Product Risk - Examples

- ▶ Missing or wrong functionality
- ▶ Incorrect calculations
- ▶ Runtime errors
- ▶ Poor architecture
- ▶ Inefficient algorithms
- ▶ Inadequate response time
- ▶ Poor user experience
- ▶ Security vulnerabilities



Product Risk - What if?

May result in negative consequences, for example:

- ▶ User dissatisfaction
- ▶ Loss of revenue, trust, reputation
- ▶ Damage to third parties
- ▶ High maintenance costs, overload of the helpdesk
- ▶ Criminal penalties
- ▶ Physical damage, injuries or death





Product Risk Analysis

Risk-based testing and Product Quality

- ▶ Risk is used to focus effort required during testing
 - ▶ Where and when to start testing
 - ▶ Which areas need more attention
- ▶ Testing is used to:
 - ▶ Reduce the **likelihood** of a negative event occurring
 - ▶ Reduce the **impact** of a negative event occurring
 - ▶ Mitigate risk
 - ▶ Provide feedback about identified risks
 - ▶ Providing feedback on unresolved risks



Risk-based testing and Product Quality (2)

- ▶ Risk-based testing:
 - ▶ Can reduce the levels of product risk in an early stage in the SDLC
 - ▶ Includes **product risk analysis**
 - ▶ Identifies product risks
 - ▶ Assesses the likelihood and impact of each risk
- ▶ Product risk information is used to guide:
 - ▶ Test Planning
 - ▶ The specification, preparation and execution of test cases
 - ▶ Test monitoring and control



Risk-based testing and Product Quality (3)

- ▶ Product Risk Analysis can influence the scope and the thoroughness of testing
- ▶ In **risk-based testing**, the results of **product risk analysis** are used to:
 - ▶ Determine the scope of testing to be carried out
 - ▶ Determine the particular test levels and propose test types to be performed
 - ▶ Determine the test techniques to be used and the coverage to be achieved
 - ▶ Estimate the test effort required for each task
 - ▶ Prioritize testing in an attempt to find critical defects as early as possible
 - ▶ Determine whether other activities besides testing should be carried out to reduce risk
- ▶ Collective knowledge and insight of the project stakeholders is used for product risk analysis

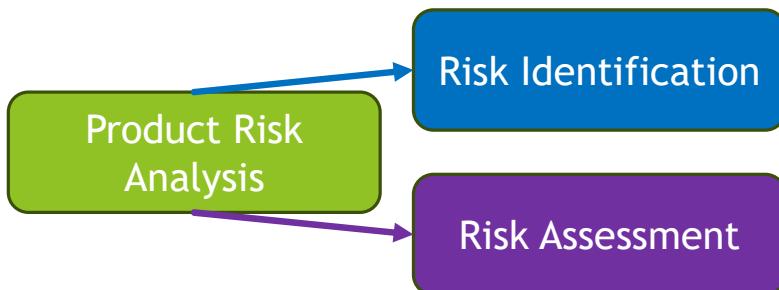
Risk-based testing and Product Quality (4)

- ▶ Minimizing the likelihood of a product failure, risk management provides a disciplined approach to:
 - ▶ Analyze risks
 - ▶ Determine which risks are important to deal with
 - ▶ Implement actions to mitigate those risks
 - ▶ Make contingency plans to deal with the risks should they become actual events
- ▶ Testing may
 - ▶ Identify new risks
 - ▶ Help to determine what risks should be mitigated
 - ▶ Lower uncertainty about risks



Product Risk Analysis

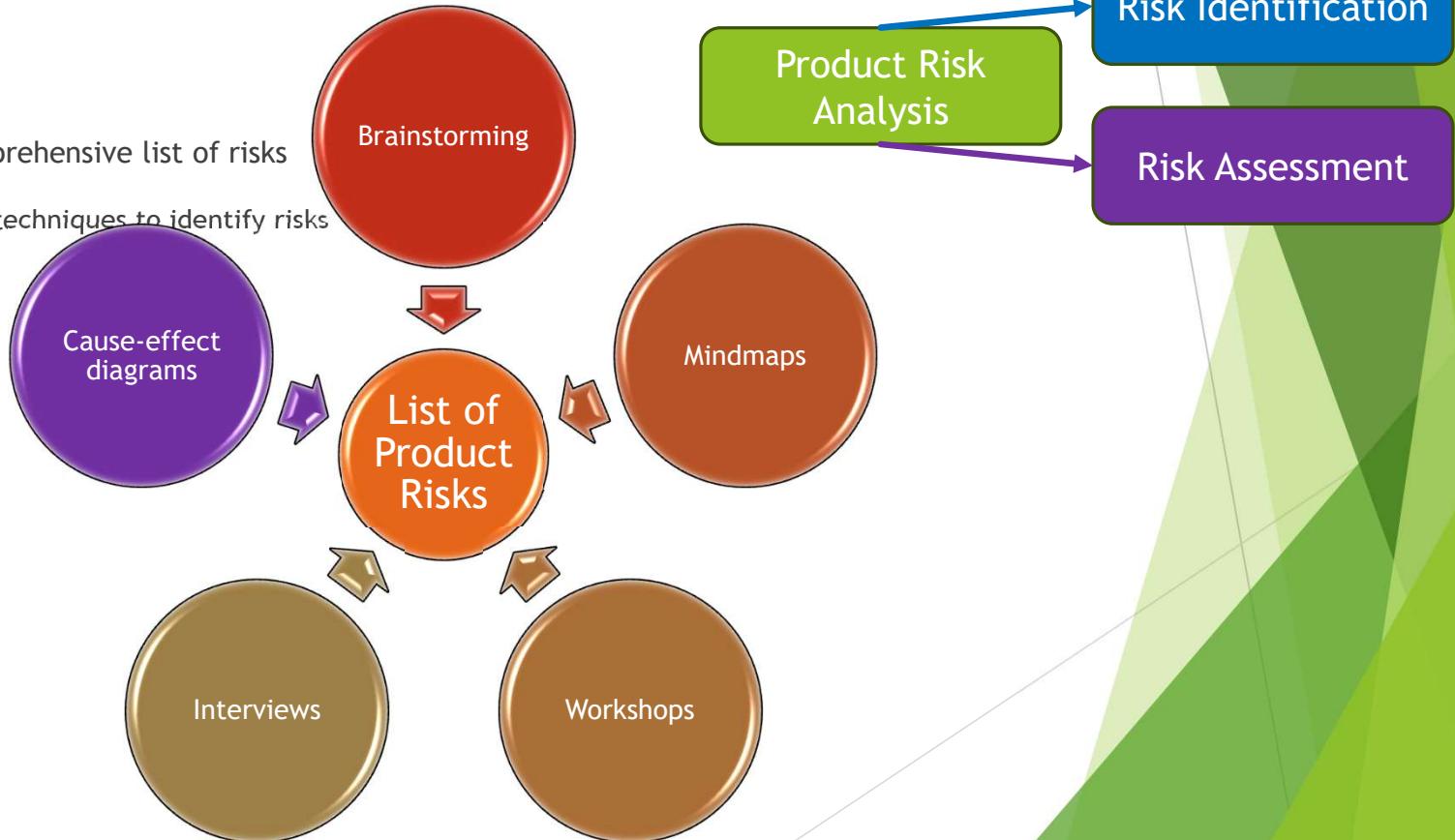
- ▶ Goal - Provide awareness of product risks
 - ▶ Focus testing efforts to minimize the residual level of product risk.
- ▶ Should start early in the SDLC
- ▶ Consists of two main activities



Product Risk Analysis - Risk Identification

► Risk Identification

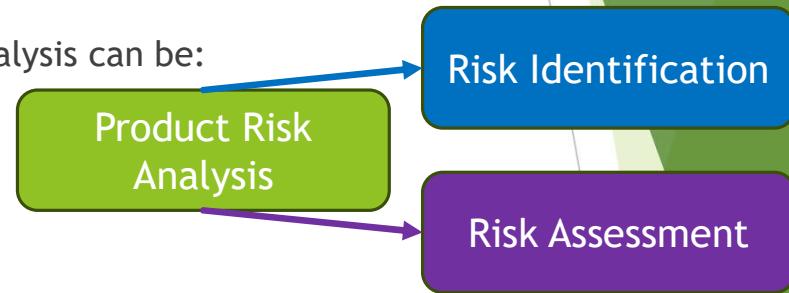
- Generate a comprehensive list of risks
- Stakeholders use techniques to identify risks



Product Risk Analysis - Risk Identification (2)

- ▶ Identification of risk items for the basis of Product Risk Analysis can be:

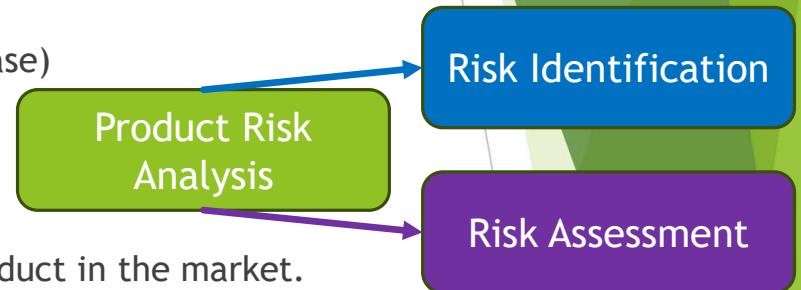
- ▶ Features
- ▶ Functionalities
- ▶ User Stories
- ▶ Requirements
- ▶ Use Cases
- ▶ Test Cases



We will use Features in our example.

Product Risk Analysis - Risk Identification (3)

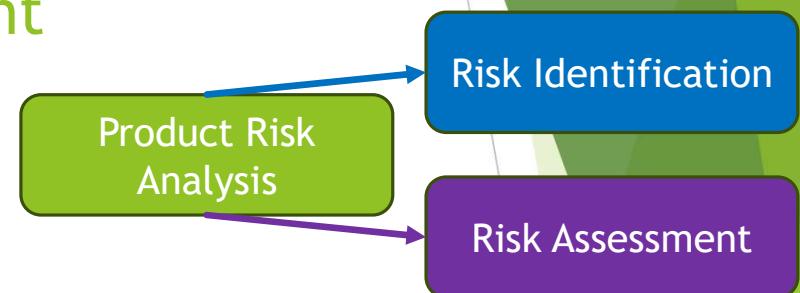
- ▶ Determining the **importance** of each risk item (feature in this case)
- ▶ This can be done in a lot of ways, including:
 - ▶ Project documentation.
 - ▶ Stakeholder's knowledge about the product and existing product in the market.
 - ▶ Inputs on most used functionality.
 - ▶ Inputs from consulting a domain expert.
 - ▶ Data from the previous version of the product or similar product in the market.



Product Risk Analysis - Risk Assessment

Risk Assessment involves:

- ▶ Categorizing identified risks (can help in assigning mitigation actions)
 - ▶ Determining the risk likelihood
 - ▶ Determining the risk impact
 - ▶ Prioritizing the risks
 - ▶ Proposing ways to handle the risks
- } Risk Level



Two approaches (or both mixed) can be used in Risk Assessment:

- ▶ **Quantitative approach** - Risk level is calculated by multiplying the risk likelihood and risk impact
- ▶ **Qualitative approach** - Risk level can be determined using a risk matrix

Product Risk Analysis - Determining the Likelihood

- ▶ Determining the **likelihood** of risk occurrence
- ▶ Likelihood of risk occurrence can be due to:
 - ▶ Poor understanding of the feature by the product development team.
 - ▶ Improper architecture and design.
 - ▶ Insufficient time to design.
 - ▶ Incompetency of the team.
 - ▶ Inadequate resources - people, tools, and technology.
- ▶ Use a 3x3 grid to execute the risk analysis
- ▶ Likelihood is on the Y axis and answers the question how likely a (in this case) feature is to fail in live operation if no further testing is undertaken in this area (likely, quite likely, unlikely)
- ▶ Executed by the **Technical team**



Product Risk Analysis - Determining the Likelihood (2)

What is the **likelihood** of failure in live operation if no further testing is undertaken in this area?

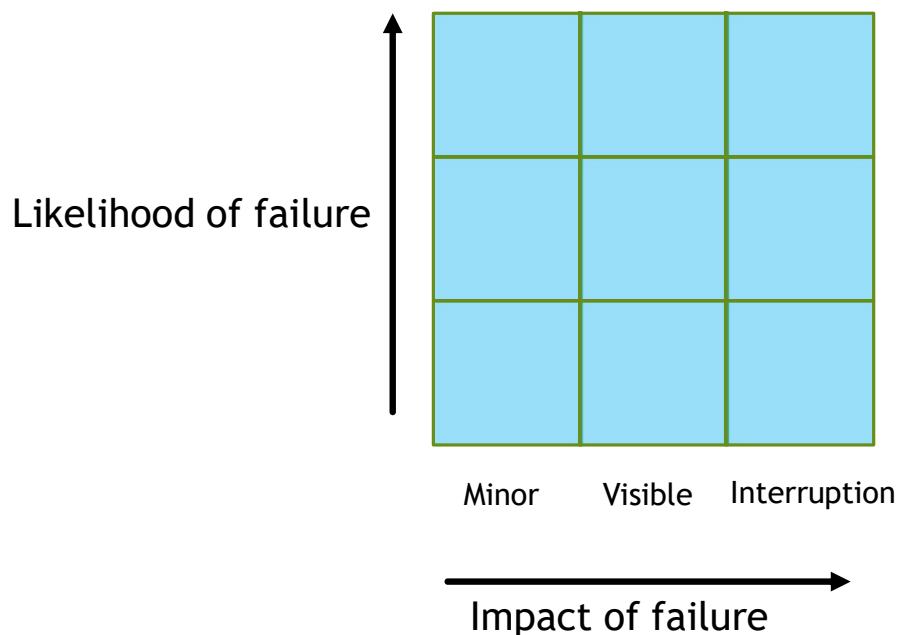
Likely			
Quite Likely			
Unlikely			

Product Risk Analysis - Determining the Impact

- ▶ Determining the impact of the risk if it would occur
- ▶ Impact of the risk could be:
 - ▶ Cost impact, resulting in a loss.
 - ▶ Business impact resulting in losing business or losing market share, Law proceedings, Loss of license.
 - ▶ Quality impact, resulting in substandard or incompetent product release.
 - ▶ Bad user experience, resulting in dissatisfaction and loss of a customer.
- ▶ Impact is on the X axis and answers the question what is the impact of a failure in live operation as experienced by the customer (interruption, minor, visible or High, medium, low)
- ▶ Executed by the **Business Specialists**

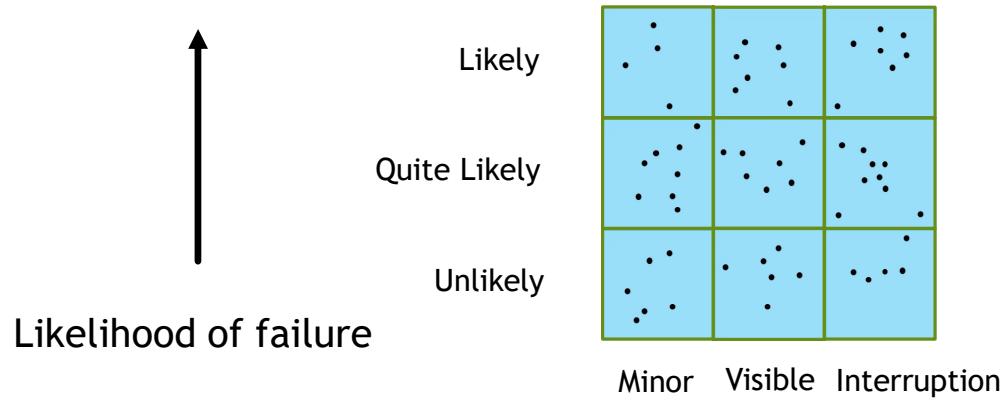
Product Risk Analysis - Risk Level

What is the impact of a failure in live operation as experienced by the customer?



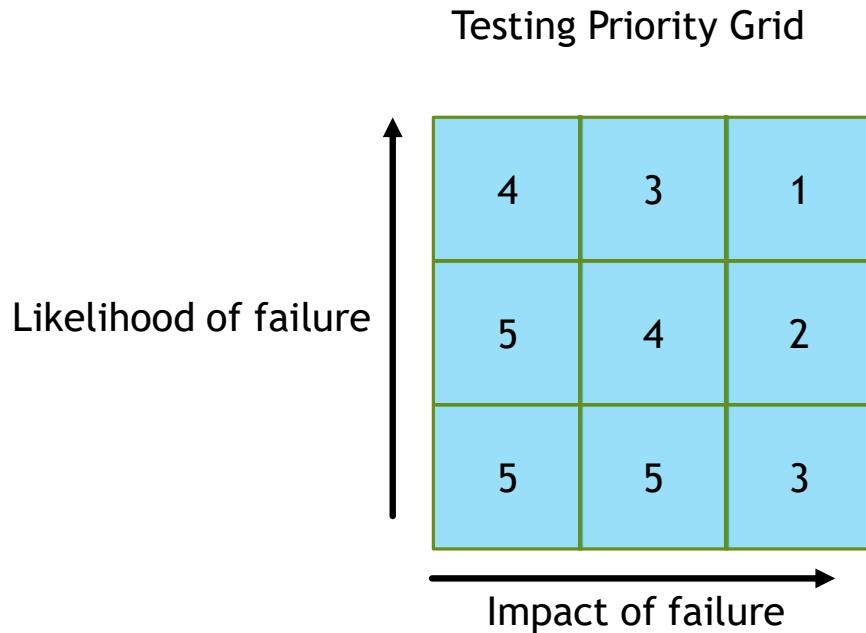
Product Risk Analysis - Risk Level (2)

Create the combined likelihood - impact grid



Impact of failure →

Product Risk Analysis - Focusing testing effort



Product Risk Analysis - Effect on Testing - Example (9)

Level of Detail of Testing

Identification	Thorough	More Thorough
De-scoped	Identification	Thorough
De-scoped	De-scoped	Thorough



Product Risk Control

Product Risk Control

- ▶ Includes all measures taken as response to the Product Risk Analysis
- ▶ Consists of:
 - ▶ Risk mitigation:
 - ▶ Implement the actions proposed in risk assessment to reduce the risk level
 - ▶ Risk monitoring aims to:
 - ▶ Ensure that the mitigation actions are effective
 - ▶ Obtain further information to improve risk assessment
 - ▶ Identify emerging risks

Responding to risk

Analyzed risks can be responded to with:

- ▶ Risk mitigation by testing
- ▶ Risk acceptance
- ▶ Risk transfer
- ▶ Contingency plan

Mitigating Product Risks by Testing

- ▶ Select the testers with the right level of experience and skill for a given risk type
- ▶ Apply an appropriate level of independence of testing
- ▶ Conduct reviews and static analysis
- ▶ Apply the correct test techniques and coverage levels
- ▶ Apply the appropriate test types addressing the affected quality characteristics
- ▶ Perform dynamic testing, including regression testing



Independence of Testing

Low

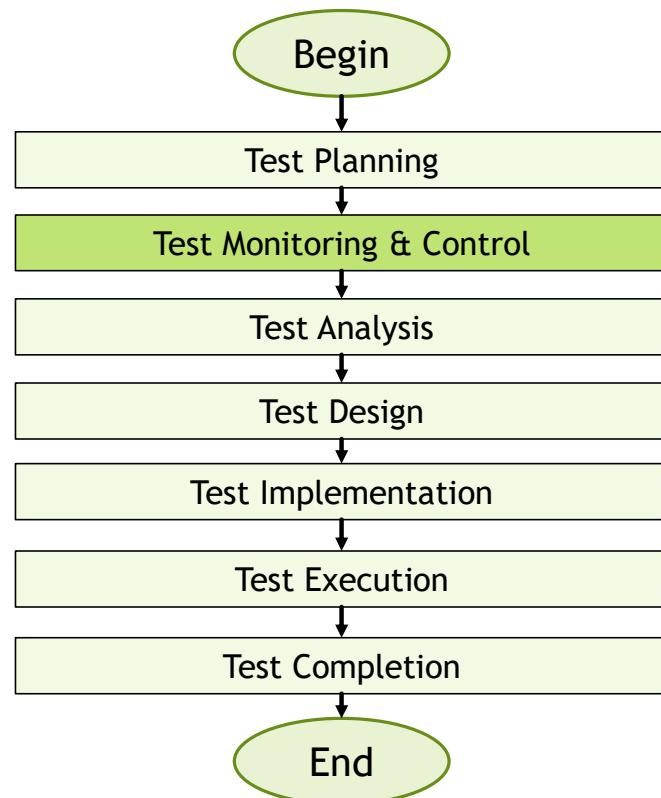
- ▶ Authors test their own work product
- ▶ Work products are tested by the author's peers in the same team
- ▶ Independent test team or group within the organization but outside the team
- ▶ Independent testers external to the organization (on-site or off-site)

High



Test Monitoring, Test
Control and Test
Completion

Test Monitoring & Control



Test Monitoring

- ▶ **ONGOING** activity comparing progress with the test plan using metrics
- ▶ **Concerned with gathering** information about testing, which is then used to:
 - ▶ Provide feedback and visibility about test activities
 - ▶ Assess Test Progress
 - ▶ Measure whether the test exit criteria or associated tasks are satisfied
 - ▶ Targets for:
 - ▶ Coverage of product risks
 - ▶ Requirements
 - ▶ Acceptance criteria



Test Control

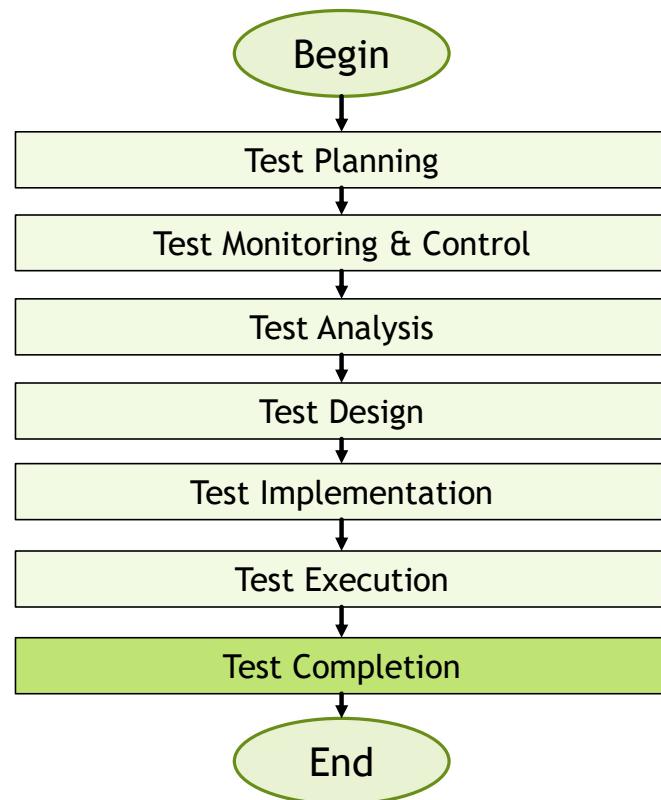
- ▶ Takes and uses the information from Test Monitoring to make testing more effective and efficient (using control directives)
 - ▶ Give guidance
 - ▶ Implement corrective actions
- ▶ Control directives can include
 - ▶ Reprioritizing tests when an identified risk becomes an issue
 - ▶ Re-evaluating whether a test item meets entry or exit criteria due to rework
 - ▶ Adjusting the test schedule to address a delay in the delivery of the test environment
 - ▶ Adding more resources when and where needed

Test Monitoring & Control

- ▶ Supported by the evaluation of exit criteria (the definition of done for testing), which may include:
 - ▶ Checking of test results and logs against coverage criteria (requirement coverage for example)
 - ▶ Assessing component or system quality based on the test results and logs
 - ▶ Determining if more tests are needed
- ▶ Progress, deviations and information needed to determine to stop testing is reported in **Test Progress Reports**
 - ▶ Test Progress Reports
 - ▶ Test Summary Reports



Test Completion



Test Completion

- ▶ Collect data from completed test activities to consolidate experience, testware and other relevant information
- ▶ Can include the following major activities:
 - ▶ Checking whether all defect reports are closed, entering change requests or Product Backlog items for any unsolved defects
 - ▶ Creating a test summary report for the stakeholders
 - ▶ Finalizing and archiving the test environment, test data, test infrastructure and other testware
 - ▶ Handing over the testware to the maintenance teams, other project teams or stakeholders
 - ▶ Analyzing lessons learned from the completed test activities to determine changes needed
 - ▶ Using the information gathered to improve test process maturity

Test Completion (2)

Activities occur at project milestones:

- ▶ Release
- ▶ Completion of the test project
- ▶ Cancellation of the test project
- ▶ An Agile project iteration that is finished
- ▶ A test level that is completed
- ▶ A maintenance release that is completed





Metrics used for Testing

Defining and Using Test Metrics

'What doesn't get measured, doesn't get done'



- ▶ Testing metrics categories:
 - ▶ **Project metrics** - progress towards project exit criteria
 - ▶ **Product metrics** - attribute of the product
 - ▶ **Process metrics** - capability of the process
 - ▶ **People metrics** - capability of individuals or groups

Metrics used for Testing

- ▶ Test Metrics can be collected at the end of any test activity to assess:
 - ▶ Progress against the planned schedule and budget
 - ▶ Current quality of the test object
 - ▶ Adequacy of the test approach
 - ▶ Effectiveness of the test activities with respect to the objectives
- ▶ Mainly gathered in Test Monitoring to support Test Control and Test Completion.

Common Test Metrics

Project process metrics

- Task completion, resource usage, test effort

Test progress metrics

- Test case implementation progress, test environment preparation progress, nr of test cases run/not run, passed/failed, test execution time

Product quality metrics

- Availability, response time, mean time to failure

Defect metrics

- Nr and priorities of defects found/fixed, defect density, defect detection percentage

Risk metrics

- Residual risk level

Coverage metrics

- Requirements coverage, code coverage

Cost metrics

- Cost of testing, organizational cost of quality

Test Metrics - Considerations

- ▶ Metrics must frequently be reported to various stakeholders
- ▶ As metrics are used to determine the overall project success, the following must be considered:
 - ▶ Definition of metrics
 - ▶ Tracking of metrics
 - ▶ Reporting of metrics
 - ▶ Validity of metrics



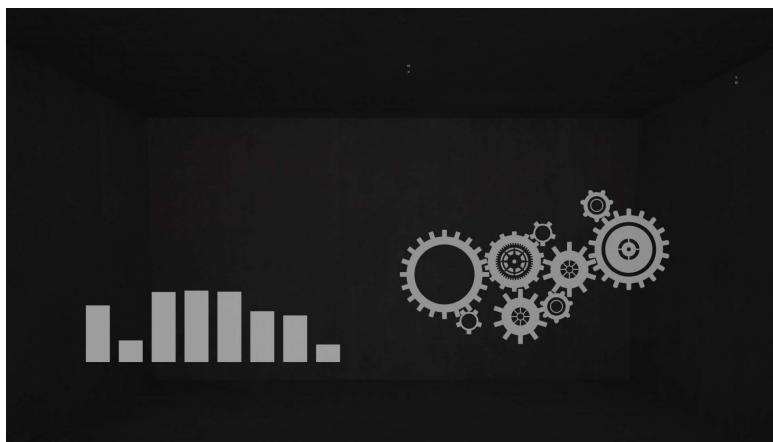
Definition of Metrics

- ▶ A limited set of useful metrics should be defined
- ▶ Metrics should be based on specific objectives for the process, product or product
- ▶ Metrics should be defined for balance (as a whole)
- ▶ Interpretation of the defined metrics must be agreed upon by all stakeholders
- ▶ Often too many metrics are defined instead of the most useful ones



Tracking of Metrics

- ▶ Reporting and merging metrics should be as much automated as possible
- ▶ Variations of measurements over time may differ from the agreed upon interpretation
- ▶ Possible differences between reality and expectations should be analyzed as well as the reasons for these differences



Reporting of Metrics

- ▶ Objective is to provide an immediate understanding of the information, for management purposes
- ▶ Presentations may show:
 - ▶ A snapshot of a metric at a certain time;
 - ▶ The evolution of a metric over time so that trends can be evaluated.



Validity of Metrics

- ▶ The information that is being reported must be verified:
 - ▶ Measurement taken for a metric may not reflect the true status
 - ▶ Measurement taken for a metric may convey an overly positive or negative trend
- ▶ Before any data is presented, it should be reviewed for:
 - ▶ Accuracy
 - ▶ The message that it is likely to convey





Purpose, Content and Audience for Test Reports

Test Reports - Purpose

- ▶ Summarize and communicate test information during and after testing:
 - ▶ Test Progress Report (during a test activity)
 - ▶ Test Completion Report (end of a test activity)
- ▶ Are about helping project stakeholders understand the progress and results of a test period
- ▶ Can help in supporting conclusions, recommendations and decisions



Test Progress Reports

- ▶ Issued to stakeholders regularly during test monitoring and control
- ▶ Often reported frequent and informal in the team
- ▶ Support the ongoing control of the testing
- ▶ Summarizes information on the testing performed in a specific reporting period
- ▶ Should provide enough information so that informed decisions can be made if needed due to deviation from the plan or changed circumstances:
 - ▶ Modifications to the test schedule
 - ▶ Modifications to the resources
 - ▶ Modifications to the test plan



Test Progress Reports - Typical Contents

- ▶ Test period
- ▶ Test progress (ahead or behind schedule, deviations, etc)
- ▶ Impediments for testing and workarounds
- ▶ Test metrics
- ▶ New and changed risks within the testing period
- ▶ Testing planned for the next period

Test Progress Report - ISO/IEC/IEEE 29119 Standard

- ▶ Reporting period
- ▶ Progress against the test plan
- ▶ Factors blocking progress
- ▶ Test Measures
- ▶ New and changed risks
- ▶ Planned testing

Test Completion Reports

- ▶ Prepared and issued to stakeholders during test completion:
 - ▶ When reaching exit criteria are met (hopefully) or at completion of:
 - ▶ A project
 - ▶ A test level
 - ▶ A test type
- ▶ Provides a summary of the performed testing based on the latest test progress report and other information
- ▶ Often reported only once and more formal than test progress reports using a set template



Test Completion Reports (2)

► Contents will vary greatly per:

- ▶ Context of the project
- ▶ Organizational requirements
- ▶ Software development lifecycle
- ▶ Audience

Test Reports - Audience

- ▶ Audience greatly influences:
 - ▶ Content in the reports
 - ▶ The degree of formality
 - ▶ The frequency of reporting



Test Completion Reports - Typical Contents

- ▶ Summary of testing performed
- ▶ Testing and product quality evaluation based on the original plan (test objectives and exit criteria)
- ▶ Deviations from the test plan (differences from the planned schedule, duration, effort)
- ▶ Factors that have blocked or are blocking progress and their workarounds
- ▶ Test Metrics based on the test progress reports
- ▶ Unmitigated risks, defects not fixed
- ▶ Lessons learned that are relevant to testing

Test Completion Report - ISO/IEC/IEEE 29119 Standard

- ▶ Scope
- ▶ Testing performed
- ▶ Deviations from planned testing
- ▶ Test completion evaluation
- ▶ Factors that blocked progress
- ▶ Test measures
- ▶ Residual risks
- ▶ Test deliverables
- ▶ Reusable test assets
- ▶ Lessons learned



Communicating the Status of Testing

Communicating the Status of Testing

There is no best way to communicate testing status to stakeholders.

- ▶ It depends on:
 - ▶ Test management concerns
 - ▶ Organizational test strategies
 - ▶ Regulatory standards
 - ▶ The team (in self-organizing teams)

Forms of communication of the test status

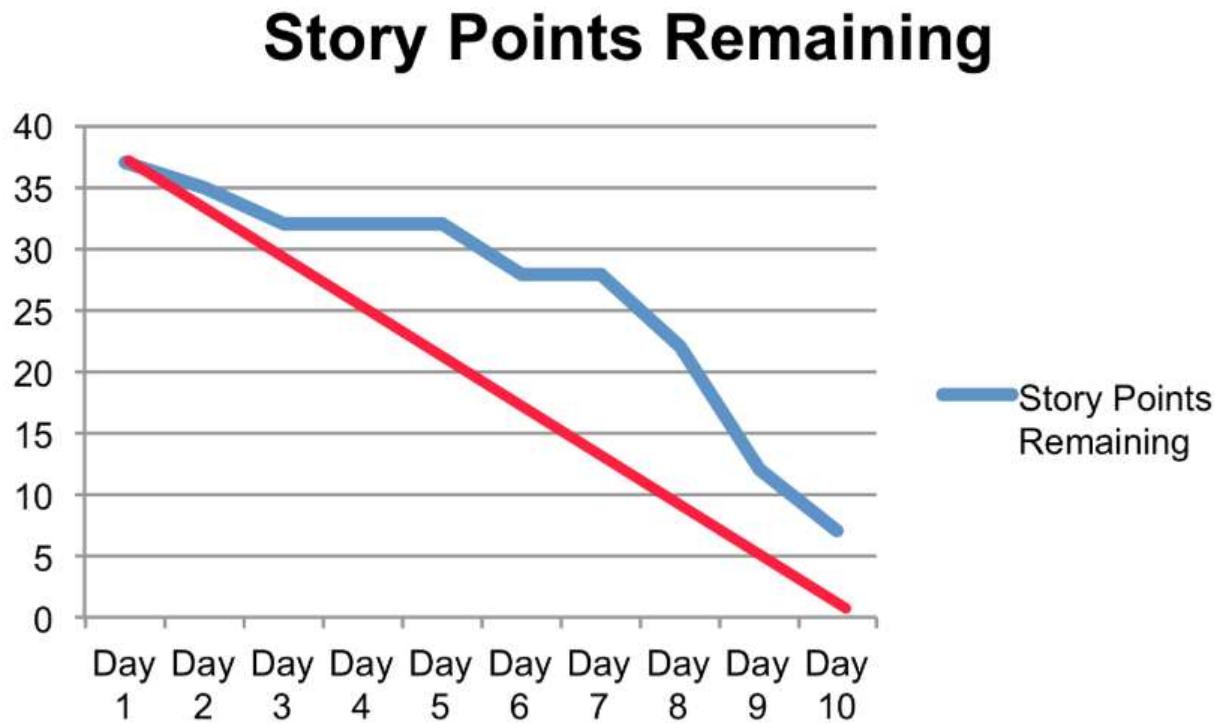
- ▶ Verbal communication
- ▶ Dashboards
- ▶ Electronic communication channels
- ▶ Online documentation
- ▶ Formal test reports:
 - ▶ Test Progress Reports
 - ▶ Test Completion reports

Information reported and the degree of formality greatly depends on audience

Monitoring and communication in Agile

- ▶ Agile teams progress by having working software at the end of each iteration.
 - ▶ Need to monitor the progress of all work items in the iteration and release.
- ▶ Testers in Agile teams utilize various methods to record test progress and status:
 - ▶ Test automation results
 - ▶ Progression of test tasks and stories on the Agile task board
 - ▶ Burndown charts showing the team's progress.
 - ▶ These can then be communicated using media such as wiki dashboards and dashboard-style emails, as well as verbally
- ▶ Agile teams may use **tools** that automatically generate status reports based on test results and task progress, which in turn update wiki-style dashboards and emails. This method of communication also gathers metrics from the testing process, which can be used in process improvement.
- ▶ Communicating test status in an automated manner also frees testers' time to focus on designing and executing more test cases.

Burndown Chart



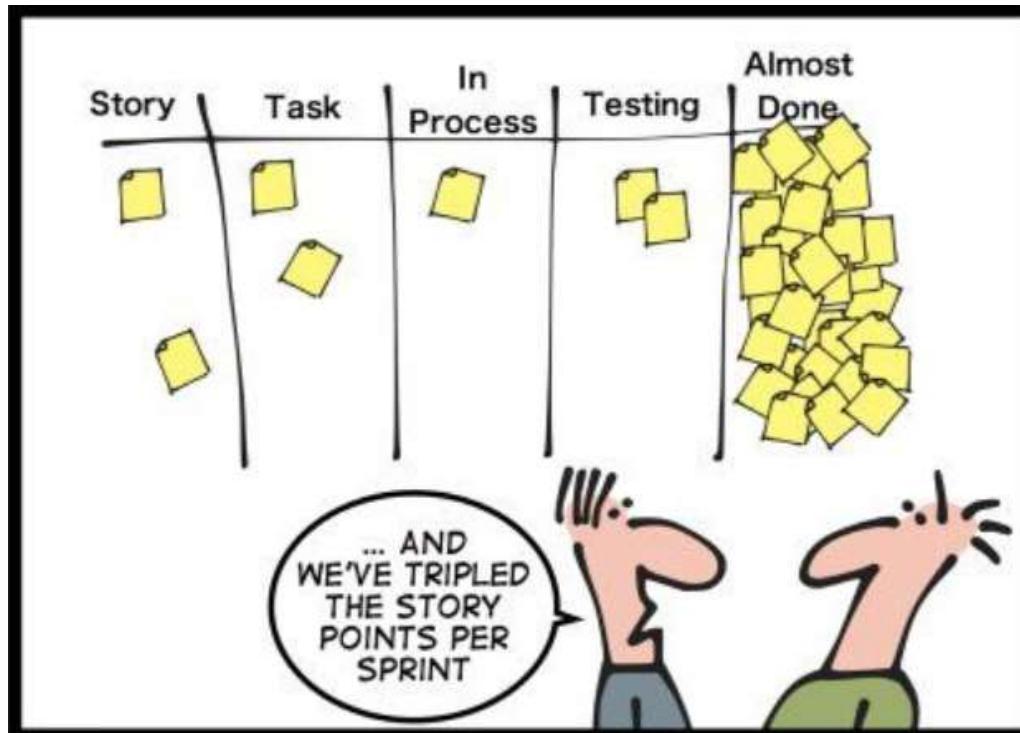
Task Boards - Example - KANBAN

To do	Plan	Code	Create	Test	Done
<p>Security patch Unassigned Start Date:</p> <p>Japanese font shows up as bold and regular Unassigned Start Date:</p> <p>Export to PowerPoint Unassigned Start Date:</p>	<p>Close tab alert: "Do you want to save changes?" Katie - Dec 15</p>	<p>Search bar won't allow user to type out full search query Tyler - Dec 13</p> <p>Export to Google Docs Holly - Dec 12</p> <p>Support Sahan Tyler - Dec 15</p>	<p>Add new categories to drop-down menu Kane - Dec 31</p>		

Task Boards - Example - SCRUM

User stories	To do	In progress	To test	Complete
As a user, I am able to search for documents so I can find them more easily.	Release file	Complete automatic testing	Update/rewrite code Complete testing	Find existing code Identify a solution
As a site visitor, I can compare different types of accounts to see which account type suits me best.	Publish pricing page	Write code	Complete automatic testing	Design the pricing web page Discuss CTA to include
As a user, I can submit questions through the website so I know how to better use the product.	Create a platform to store feedback Release code Complete testing	Write code Design a feedback form		

Task Boards

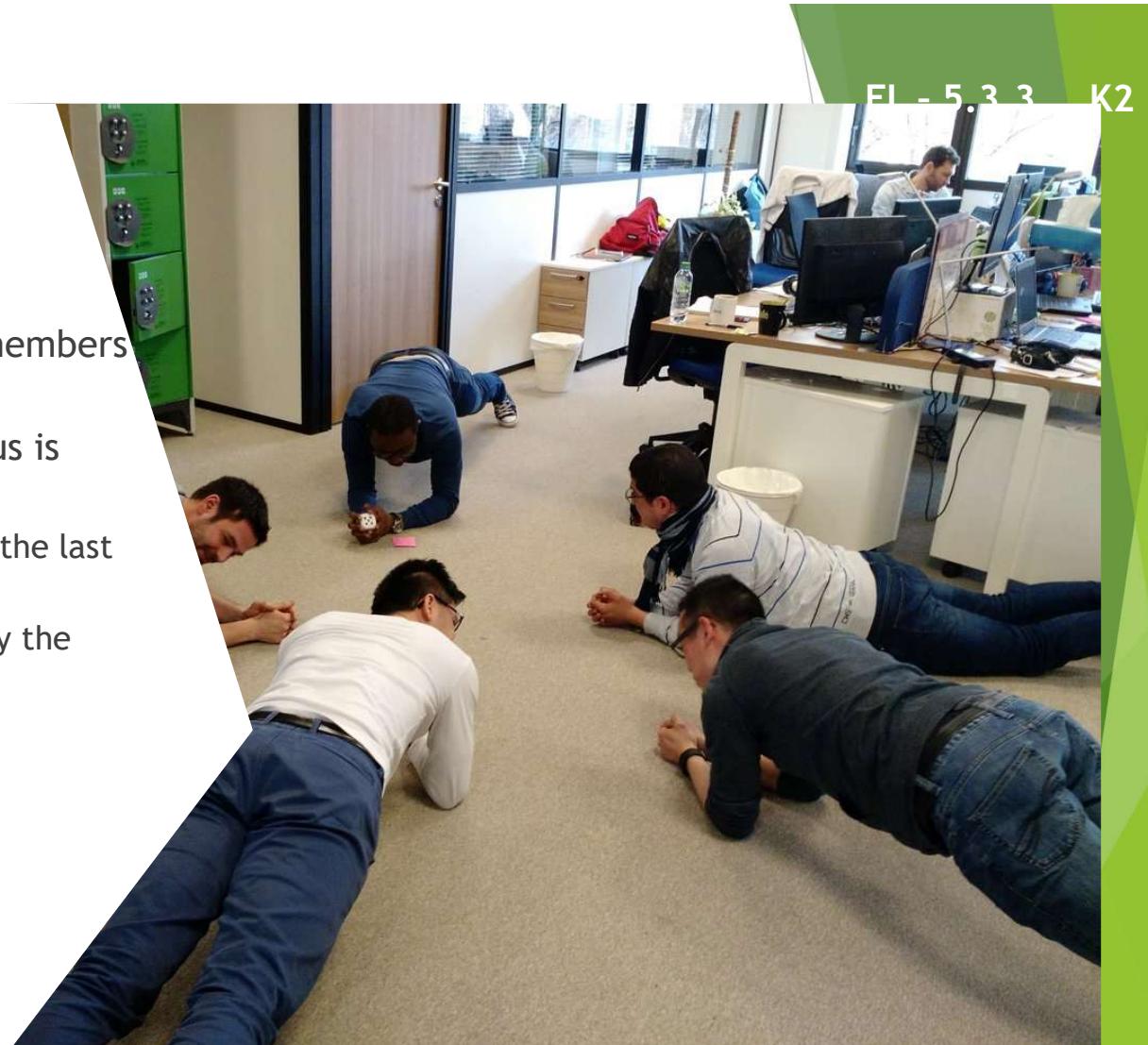


Task boards - testing tasks

- ▶ Relate to the acceptance criteria of the user stories
- ▶ Testing task moves in the ‘Done’ column when the below pass:
 - ▶ Test automation script
 - ▶ Manual test
 - ▶ Exploratory tests

Daily stand-up

- ▶ Daily meeting that includes all members of the Agile team.
- ▶ In the meeting, the current status is communicated:
 - ▶ What have you completed since the last meeting?
 - ▶ What do you plan to complete by the next meeting?
 - ▶ What is getting in your way?
(impediments)





Configuration Management

Configuration Management

- ▶ Establish and maintain the integrity of the component, system, testware and their relationships to one another through the project and product life cycle.
- ▶ For testing, configuration management involves ensuring:
 - ▶ All test items and items of testware are:
 - ▶ Identified
 - ▶ Version controlled
 - ▶ Tracked for changes



Configuration Management (2)

- ▶ Complex configuration items (i.e. test environment), CM records:
 - ▶ All items it consists of
 - ▶ Their relationships and
 - ▶ Their versions
- ▶ After the configuration item is approved for testing:
 - ▶ It becomes the new baseline
 - ▶ It can only be changed through a formal change control process
 - ▶ CM keeps a record of ALL changed configuration items
 - ▶ If needed to reproduce previous test results, a previous baseline can be reverted to



Configuration Management (3)

- ▶ All configuration items including each part of the test object should be:

- ▶ Uniquely identified
- ▶ Version controlled
- ▶ Tracked for changes
- ▶ Related to other configuration items for traceability



- ▶ All documentation and software items should be referenced unambiguously in test documentation:

- ▶ Including the ID
- ▶ The version
- ▶ Change record

Testing and configuration management

Heavy use of automated tools to develop, test and manage software development

Tools are used by developers for:

- ▶ Static Analysis
- ▶ Unit Testing
- ▶ Code Coverage



Code & Unit tests are constantly checked into the Configuration Management system, using automated build and test frameworks for CI / CD including automated DevOps pipelines

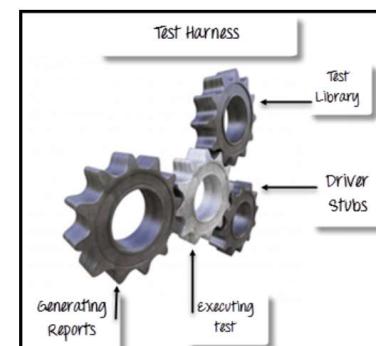
Functional testing and configuration management

Automated tests can also include functional tests on the integration and system levels.

These tests can be integrated with the other automated tests as part of the CI framework.

Functional automated tests can be created using:

- ▶ Functional testing harnesses
- ▶ Open-source UI test tools
- ▶ Commercial Tools



As automated functional tests take longer than unit tests, often these are separated:

- ▶ E.g. unit tests run every day or during every check-in; functional tests every few days

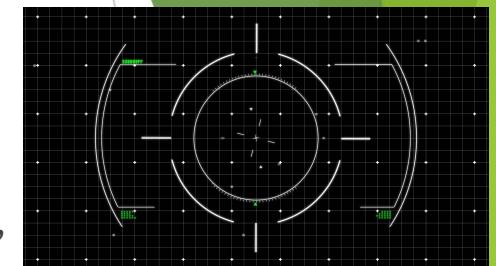


Defect Management

Defect management

- ▶ Any defect or anomaly should be identified and tracked from discovery to resolution
- ▶ To manage all defects to resolution, a defect management process should be in place, including workflow and rules for classification
- ▶ Workflow typically includes activities for logging, analyzing and classifying them
- ▶ The way in which defects are logged depends on:
 - ▶ Context of the component or system
 - ▶ Test level
 - ▶ Software Development Lifecycle model
- ▶ Process must be agreed upon with all stakeholders involved in defect management

FL - 5.5.1 K3



Defect management (2)

- ▶ ‘Defects’ are not always issues in the item under test (change request, false positive, etc)
- ▶ Defect reports have the following objectives:
 - ▶ Provide those responsible for handling and resolving reported defects with enough information to resolve the issue
 - ▶ Provide a means of tracking the quality of the work product
 - ▶ Provide ideas for development and test process improvements

Defect management (3)

- ▶ Defects can be reported in all stages of the development lifecycle
- ▶ Defects may be reported on all work products
- ▶ **Defect Detection Percentage (DDP)** compares field defects (found in production) with test defects (found before production) to measure the test process effectiveness.

$$\frac{\text{Defects (testers)}}{\text{Defects (testers) + Defects (field)}} \times 100 \%$$

- ▶ In the defect reports, some data may automatically be included or managed
- ▶ Defects found in static testing will normally be documented differently (review meeting notes) but handled in a similar way
- ▶ Standards can be found in ISO/IEC/IEEE 29119-3) ‘Incident Reports’

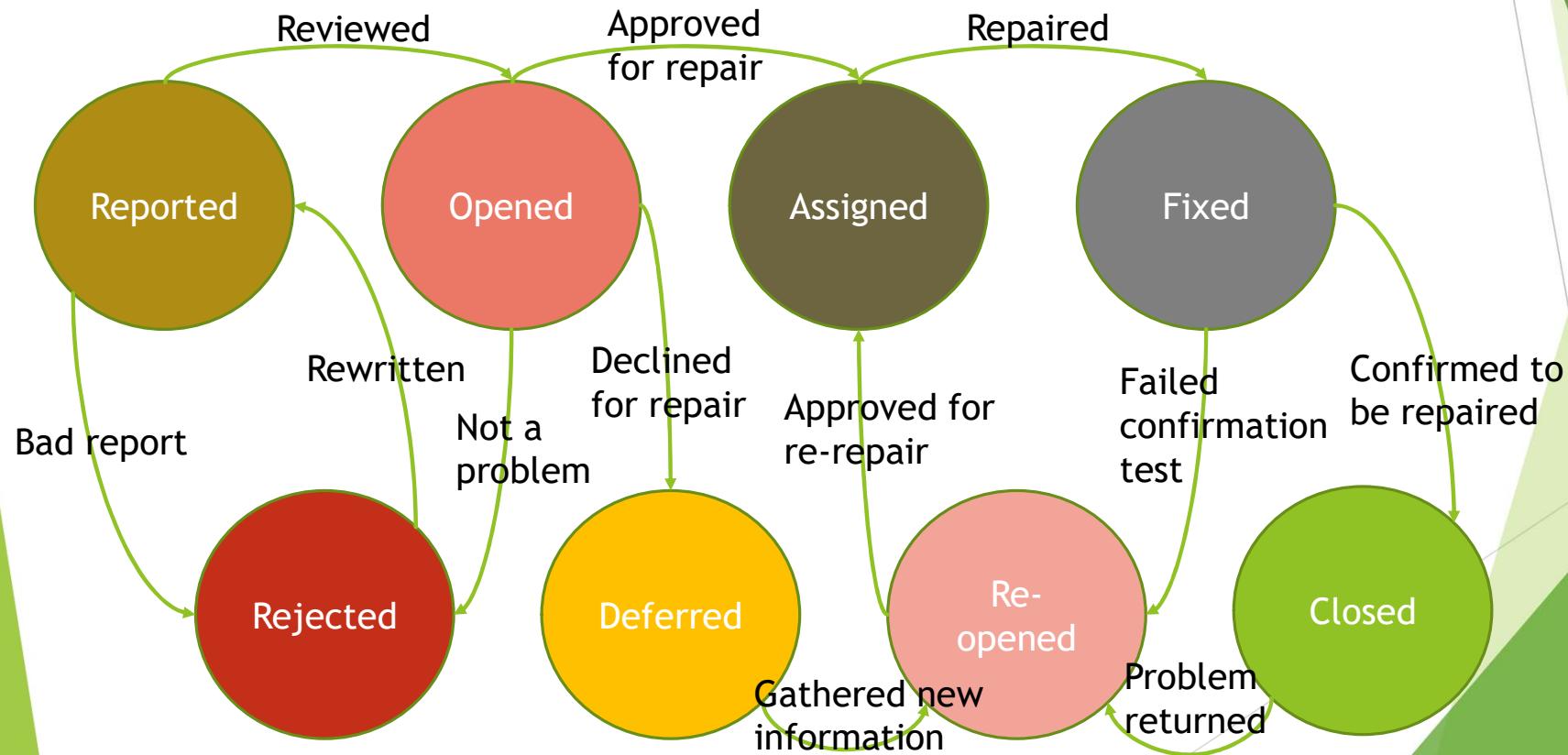
Defect Report - Typical Contents

- ▶ Unique Identifier
- ▶ Title and short summary of the of the anomaly
- ▶ Date when the anomaly was observed, issuing organization, author, author role
- ▶ Identification of the test object and test environment
- ▶ Context of the defect:
 - ▶ Development lifecycle phase
 - ▶ Test case being run
 - ▶ Test activity being performed
 - ▶ Any other relevant information
- ▶ Description of the failure incl logs, db dumps, screenshots (reproduceable steps)
- ▶ Expected and actual results
- ▶ Severity of the defect on the stakeholder's interests or requirements
- ▶ Priority to fix
- ▶ Status of the defect (open, deferred, duplicated, fixed, etc)
- ▶ References

Incident Report - ISO/IEC/IEEE 29119 Standard

- ▶ Overview
- ▶ Timing information
- ▶ Originator
- ▶ Context
- ▶ Description of the incident
- ▶ Originator's assessment of the severity
- ▶ Originator's assessment of the impact
- ▶ Risk
- ▶ Status of the incident

Defect Report Life Cycle





Managing the Test Activities

Summary - Keywords Explained

The purpose of a Test Plan

- ▶ Documents the means and the schedule for achieving the test objectives (how and when)
- ▶ Helps to ensure that the performed test activities will meet the established criteria
- ▶ Serves as a means of communication with team members and other stakeholders
- ▶ Demonstrates that testing will be in line with the test policy and test strategy (or explains where it will not be in line)

Typical Contents of a Test Plan

- ▶ Context of testing:
 - ▶ Scope
 - ▶ Test objectives
 - ▶ Constraints
 - ▶ Test basis
- ▶ Assumptions and constraints
- ▶ Stakeholders
 - ▶ Roles
 - ▶ Responsibilities
 - ▶ Relevance to testing
 - ▶ Hiring and training needs
- ▶ Communication
 - ▶ Forms and frequency
 - ▶ Documentation templates
- ▶ Risk register
 - ▶ Project risks
 - ▶ Product risks
- ▶ Test approach
 - ▶ Test levels and test types
 - ▶ Test techniques
 - ▶ Test deliverables
 - ▶ Entry and Exit criteria
 - ▶ Independence of testing
- ▶ Metrics to be collected
- ▶ Test data requirements
- ▶ Test environment requirements
- ▶ Deviations from strategy or policy
- ▶ Budget and Schedule

Entry vs Exit Criteria

Entry Criteria: 'define the preconditions for starting a certain activity.'

Aka '**Definition of Ready**'

- ▶ *Entry Criteria not being defined or not met will make the activity:*

More Difficult

Time
consuming

More Costly

Riskier

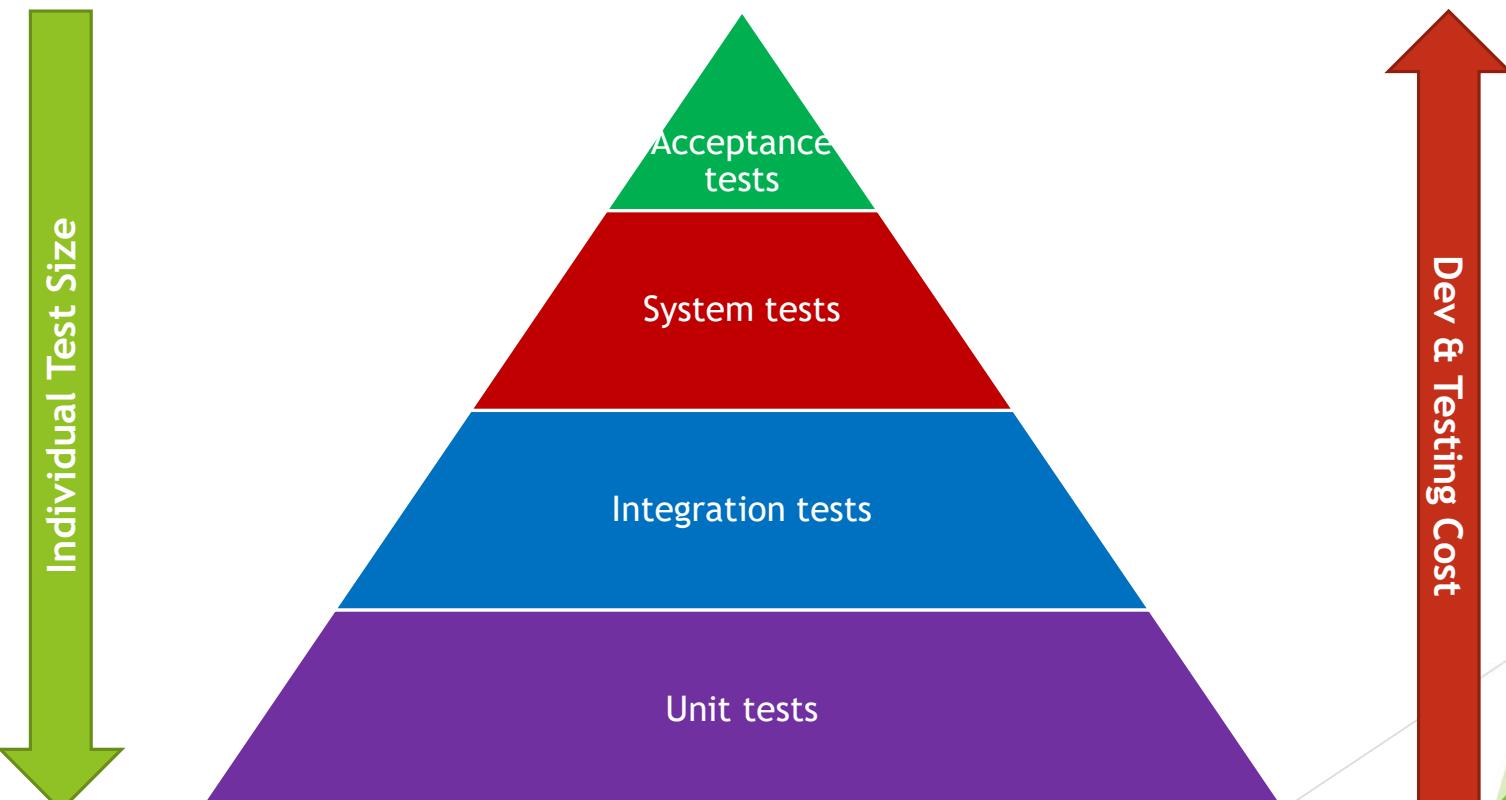
Exit Criteria: 'define what must be achieved to declare an activity completed'

Aka '**Definition of Done**'

Both Entry and Exit criteria should be defined for:

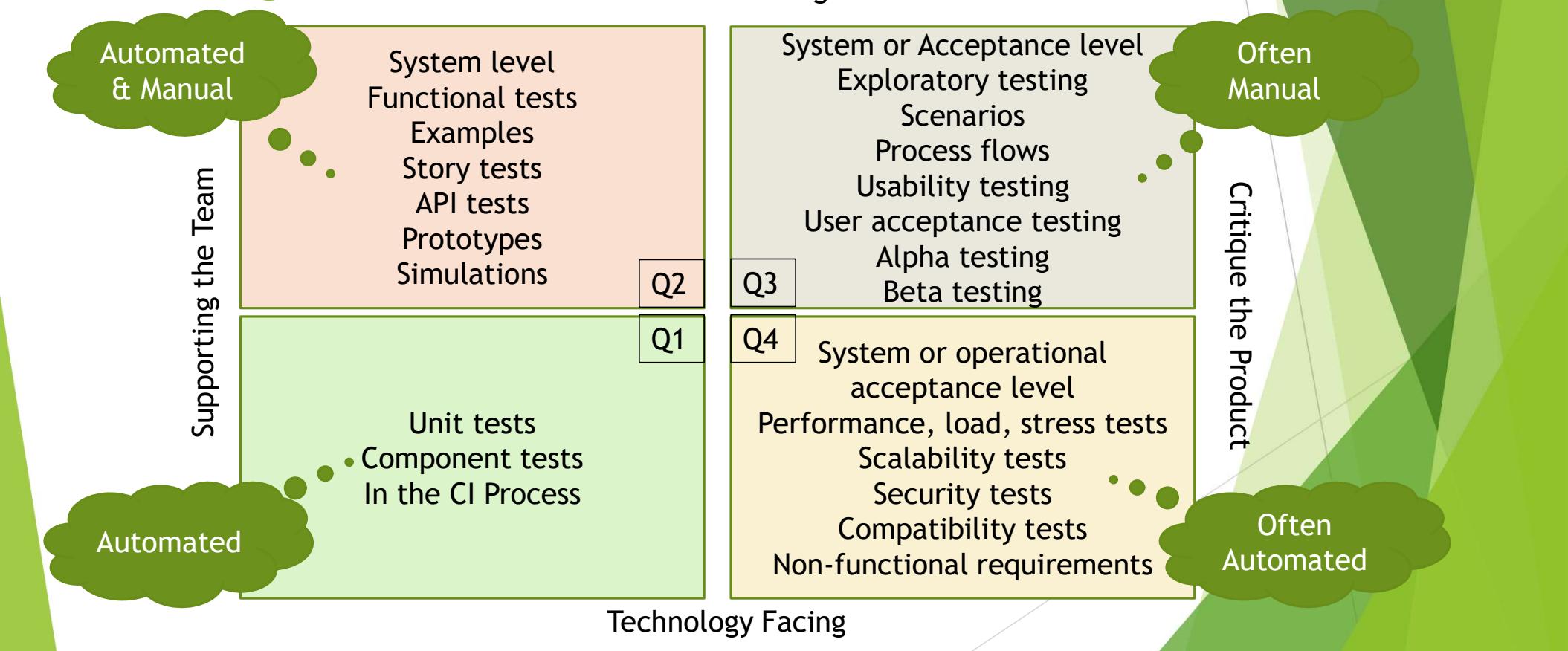
- ▶ Each test level
- ▶ Each test type
- ▶ Differ based on the test objectives

The Test Pyramid



FL - 5.1.6 K1

Testing Quadrants



What is risk?

'A potential event, hazard, threat or situation whose occurrence causes an adverse effect'

Risk is characterized by two factors:

- ▶ Risk likelihood
 - ▶ 'The probability of the risk occurrence' (greater than zero and less than 1)
- ▶ Risk impact
 - ▶ 'The consequences of this occurrence' - the harm

Risk Level

- ▶ A measure for the risk
 - ▶ The higher the risk level, the more important the treatment
- ▶ A calculated result from the combination of:
 - ▶ Risk Likelihood (greater than zero and smaller than one)
 - ▶ Risk Impact



Project Risk

- ▶ Are related to the Management and Control of the project
 - ▶ If project risks occur, this may have a negative effect on
 - ▶ Project Schedule
 - ▶ Budget
 - ▶ Scope
- Affects the project's ability to achieve its objectives.



Product Risk

- ▶ The possibility that the system or software might fail to satisfy the legitimate needs of the users and/or stakeholders
- ▶ Related to the product quality characteristics - A risk to the **quality** of the product

Product Risk



FL - 5.2.2 K2

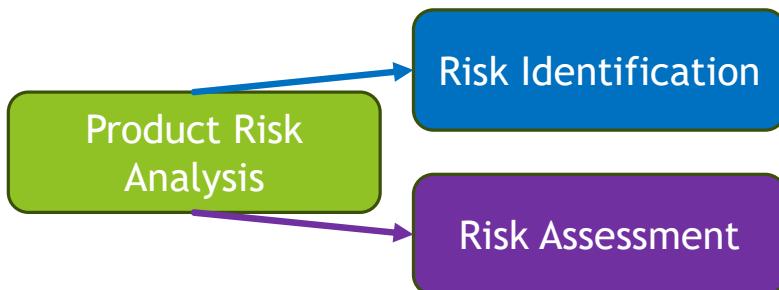
Risk-based Testing

- ▶ Test approach
- ▶ Risk Analysis and Risk Control are used to:
 - ▶ Select test activities
 - ▶ Prioritize test activities
 - ▶ Manage test activities



Product Risk Analysis

- ▶ Goal - Provide awareness of product risks
 - ▶ Focus testing efforts to minimize the residual level of product risk.
- ▶ Should start early in the SDLC
- ▶ Consists of two main activities



Product Risk Analysis - Risk Identification

► Risk Identification

- Generate a comprehensive list of risks
- Stakeholders use techniques to identify risks



Product Risk Analysis - Risk Assessment

Risk Assessment involves:

- ▶ Categorizing identified risks (can help in assigning mitigation actions)
 - ▶ Determining the risk likelihood
 - ▶ Determining the risk impact
 - ▶ Prioritizing the risks
 - ▶ Proposing ways to handle the risks
- } Risk Level



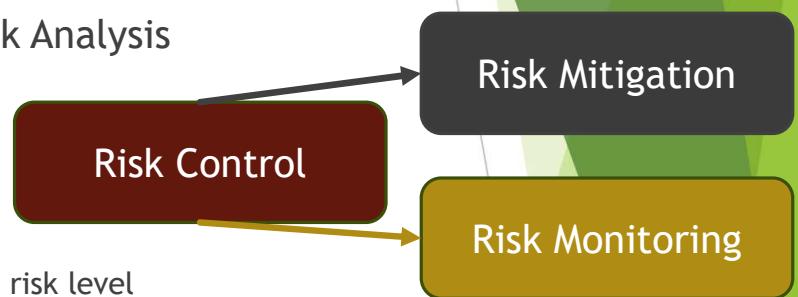
Two approaches (or both mixed) can be used in Risk Assessment:

- ▶ **Quantitative approach** - Risk level is calculated by multiplying the risk likelihood and risk impact
- ▶ **Qualitative approach** - Risk level can be determined using a risk matrix

Product Risk Control

- ▶ Includes all measures taken as response to the Product Risk Analysis
- ▶ Consists of:

- ▶ Risk mitigation:
 - ▶ Implement the actions proposed in risk assessment to reduce the risk level
- ▶ Risk monitoring aims to:
 - ▶ Ensure that the mitigation actions are effective
 - ▶ Obtain further information to improve risk assessment
 - ▶ Identify emerging risks



Test Monitoring

- ▶ **ONGOING** activity comparing progress with the test plan using metrics
- ▶ **Concerned with gathering** information about testing, which is then used to:
 - ▶ Provide feedback and visibility about test activities
 - ▶ Assess Test Progress
 - ▶ Measure whether the test exit criteria or associated tasks are satisfied
 - ▶ Targets for:
 - ▶ Coverage of product risks
 - ▶ Requirements
 - ▶ Acceptance criteria



Test Control

- ▶ Takes and uses the information from Test Monitoring to make testing more effective and efficient (using control directives)
 - ▶ Give guidance
 - ▶ Implement corrective actions
- ▶ Control directives can include
 - ▶ Reprioritizing tests when an identified risk becomes an issue
 - ▶ Re-evaluating whether a test item meets entry or exit criteria due to rework
 - ▶ Adjusting the test schedule to address a delay in the delivery of the test environment
 - ▶ Adding more resources when and where needed

Test Monitoring & Control

- ▶ Supported by the evaluation of exit criteria (the definition of done for testing), which may include:
 - ▶ Checking of test results and logs against coverage criteria (requirement coverage for example)
 - ▶ Assessing component or system quality based on the test results and logs
 - ▶ Determining if more tests are needed
- ▶ Progress, deviations and information needed to determine to stop testing is reported in **Test Progress Reports**
 - ▶ Test Progress Reports
 - ▶ Test Summary Reports



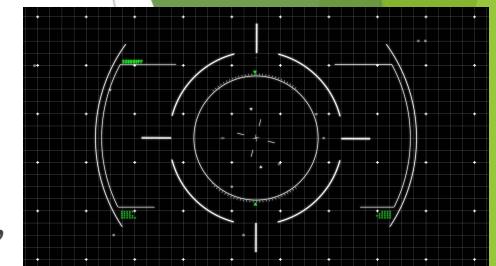
Test Completion

- ▶ Collect data from completed test activities to consolidate experience, testware and other relevant information
- ▶ Can include the following major activities:
 - ▶ Checking whether all defect reports are closed, entering change requests or Product Backlog items for any unsolved defects
 - ▶ Creating a test completion report for the stakeholders
 - ▶ Finalizing and archiving the test environment, test data, test infrastructure and other testware
 - ▶ Handing over the testware to the maintenance teams, other project teams or stakeholders
 - ▶ Analyzing lessons learned from the completed test activities to determine changes needed
 - ▶ Using the information gathered to improve test process maturity

Defect management

- ▶ Any defect or anomaly should be identified and tracked from discovery to resolution
- ▶ To manage all defects to resolution, a defect management process should be in place, including workflow and rules for classification
- ▶ Workflow typically includes activities for logging, analyzing and classifying them
- ▶ The way in which defects are logged depends on:
 - ▶ Context of the component or system
 - ▶ Test level
 - ▶ Software Development Lifecycle model
- ▶ Process must be agreed upon with all stakeholders involved in defect management

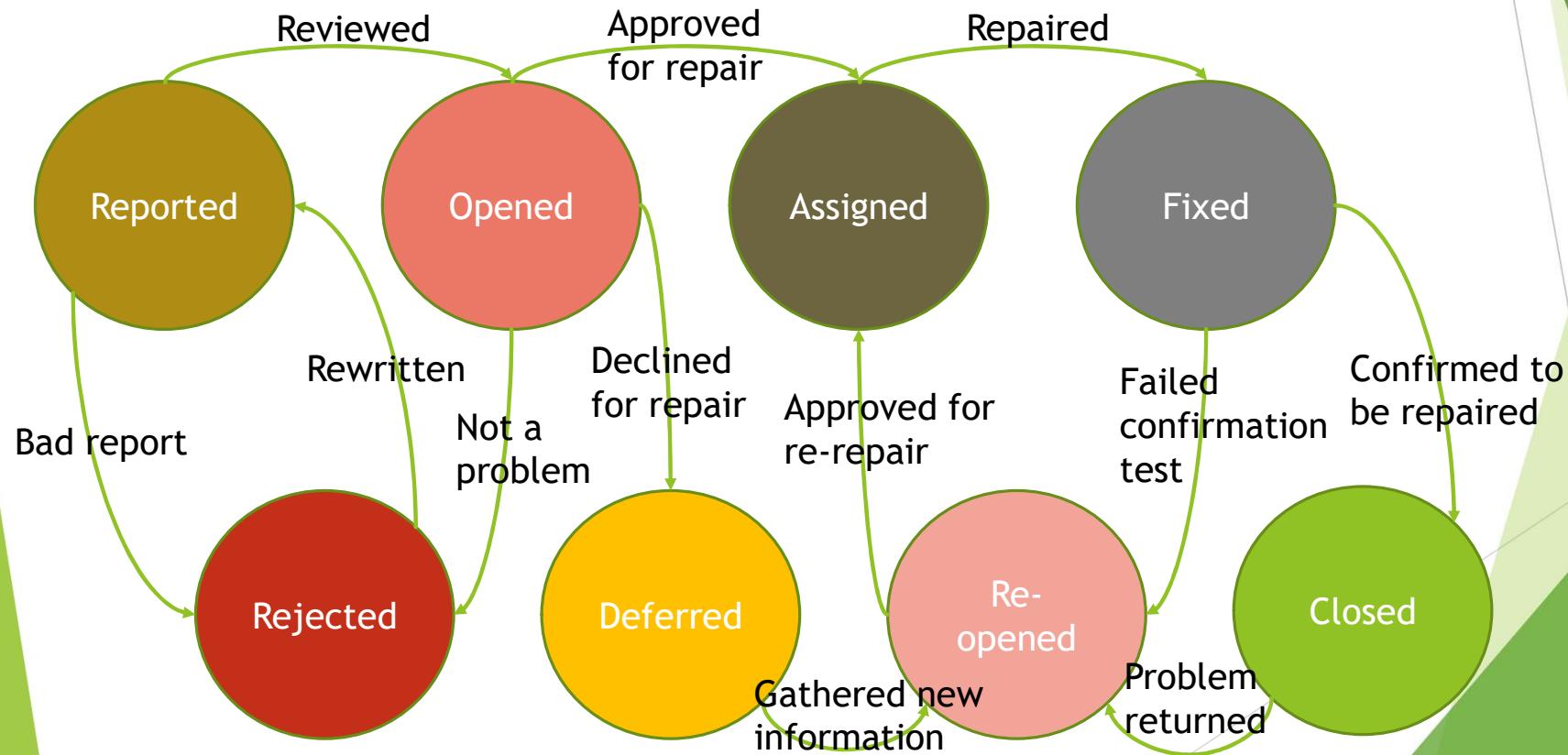
FL - 5.5.1 K3



Defect Report - Typical Contents

- ▶ Unique Identifier
- ▶ Title and short summary of the of the anomaly
- ▶ Date when the anomaly was observed, issuing organization, author, author role
- ▶ Identification of the test object and test environment
- ▶ Context of the defect:
 - ▶ Development lifecycle phase
 - ▶ Test case being run
 - ▶ Test activity being performed
 - ▶ Any other relevant information
- ▶ Description of the failure incl logs, db dumps, screenshots (reproduceable steps)
- ▶ Expected and actual results
- ▶ Severity of the defect on the stakeholder's interests or requirements
- ▶ Priority to fix
- ▶ Status of the defect (open, deferred, duplicated, fixed, etc)
- ▶ References

Defect Report Life Cycle





Test Tools

Introduction

Agenda

6.1 Tool Support for Testing

- ▶ FL-6.1.1 (K2) Explain how different types of test tools support testing

6.2 Benefits and Risks of Test Automation

- ▶ FL-6.2.1 (K1) Recall the benefits and risks of test automation





Tool Support for Testing

Tool support for testing

- ▶ Why use tools?
 - ▶ People are good in certain things and less good in other things
 - ▶ Computers are good in certain things and less good in other things
 - ▶ Give each his own and get a more reliable and efficient process
- ▶ Tools can be classified on purpose, pricing, licensing, technology, etc.
- ▶ In this course we classify on test activities they support
- ▶ Some tools are specialized in one activity; others support more than one
- ▶ If tools from a single provider work together, they might be provided as an integrated suite



Probe effect = A tool that measures some aspect of software might have unexpected side-effects on that software and the measured aspect

Purpose of using tools in testing

- ▶ What can tools be used for?
 - ▶ Directly in testing
 - ▶ Managing the test process
 - ▶ Investigation and evaluation
 - ▶ Aid in testing
- ▶ What is the purpose of tool support?
 - ▶ Improve efficiency through automation
 - ▶ Improve efficiency of test activities by supporting manual test activities
 - ▶ Improve the quality of test activities - more consistent and higher level of defect reproducibility
 - ▶ Automate activities that can not be done manually
 - ▶ Increase reliability of testing

Test Tool Classification

- ▶ Management tools
 - ▶ Increase the test process efficiency by facilitating management of the SDLC, requirements, tests, defects, configuration
- ▶ Static testing tools
 - ▶ Support the tester in performing reviews and static analysis
- ▶ Test design and implementation tools
 - ▶ Facilitate generation of test cases, test data and test procedures
- ▶ Test execution and coverage tools
 - ▶ Facilitate automated test execution and coverage measurement



Playwright



Test Tool Classification

- ▶ Non-functional testing tools
 - ▶ Allow the tester to perform non-functional testing that is difficult or impossible to perform manually

 **BlazeMeter**

 **LOAD RUNNER**

- ▶ DevOps tools
 - ▶ Support the DevOps delivery pipeline, workflow tracking, automated build process(es), CI/CD

 **kubernetes**

 **Jenkins**

- ▶ Collaboration tools
 - ▶ Facilitate communication



- ▶ Tools supporting scalability and deployment standardization
 - ▶ Virtual machines, containerization tools



**CITRIX®
Hypervisor**

- ▶ Any other tool that assists in testing



Tool Support for Testing

Benefits & Risks of Test
Automation

Potential benefits of using tools

Acquisition of a tool for automated test execution does not guarantee success. Effort is required to achieve real and lasting benefits

Potential benefits include:

- ▶ Reduction of repetitive manual work
- ▶ Greater consistency and repeatability
- ▶ More objective assessment
- ▶ Ease of access to information about tests or testing
- ▶ Reduced test execution time
- ▶ More time for testers to create better tests



Potential risks of using tools

- ▶ Unrealistic expectations
- ▶ Underestimating the time, cost and effort involved for the introduction, use and maintenance
- ▶ Underestimating the effort to maintain the test assets generated by the tool
- ▶ Over-reliance on the tool
- ▶ Failing to use proper configuration management for the test assets
- ▶ Relationships and interoperability between critical tools may be overlooked or neglected
- ▶ Possibility that the vendor goes out of business or provide poor or no response
- ▶ Open source project might be suspended
- ▶ The automation tool is incompatible with the development platform.
- ▶ A tool is chosen that does not comply with regulatory or safety standards





Tool Support for Testing

Considerations for Test
Execution and Test
Management Tools

Points of consideration - Test Execution Tools

- ▶ Requires significant effort to achieve significant benefit
- ▶ Linear scripts are not stable nor maintainable - 'smart' image capturing technology is an improvement
- ▶ Data-driven approach separates the test input (data) from the test script; testers without scripting knowledge can create the test data for the predefined scripts
- ▶ Keyword-driven approach separates the test input and actions (keywords); testers without scripting knowledge can then create tests using the keywords and the test data
- ▶ Model-based testing tools enables a functional specification to be captured as a model (i.e. activity diagram) which the tool then interprets and creates test case specifications, which can be saved in a test management tool and executed by a test execution tool.

Points of consideration - Test Management Tools

- ▶ Often needs to interface with other tools in order to:
 - ▶ Produce useful information in a useful format
 - ▶ Maintain consistent traceability to requirements in a requirement management tool
 - ▶ Link with test object version information in the configuration management tool
- ▶ Particularly important when using an integrated tool that is used by different groups in the organization



Tool Support for Testing

Effective Use of Tools

Effective use of tools - Selection

- ▶ You should not change processes to fit the tool, but choose the tool to fulfil the need of the organization
- ▶ Factors that should be taken into account when selecting a tool:

Organization's maturity	Evaluation of the vendor
Identification of opportunities for an improved test process with tools	Identification of internal requirements for coaching and mentoring
Understanding of the technologies used by the test object	Evaluation of training needs
Build and continuous integration tools already used	Pros and cons of various licensing models
Evaluation of the tool against clear requirements and objectives	Estimation of cost-benefit ratio

- ▶ Proof-of-concept to prove the tool works with the test object and in the infrastructure, or not...

Introducing a tool into an organization - Pilot

- ▶ A pilot project has the following objectives
 - ▶ Gain in-depth knowledge about the tool / its strengths and weaknesses
 - ▶ Evaluate how the tool would fit with existing processes and practices
 - ▶ Determine which changes are needed
 - ▶ Decide on standard ways of using, managing, storing and maintaining the tool and test assets
 - ▶ Assess whether the benefits will be achieved at reasonable cost
 - ▶ Understand the metrics you wish the tool to collect and report and configure the tool to make sure these metrics can be captured and reported on

Introducing a tool into an organization - Success Factors

- ▶ Roll-out incrementally
- ▶ Adapt and improve processes, testware and artifacts to get proper balance
- ▶ Provide support, training, coaching and mentoring for new users
- ▶ Define usage guidelines
- ▶ Gather usage information from the actual use of the tool
- ▶ Implement a continuous improvement mechanism
- ▶ Monitor the use of the tool and benefits
- ▶ Provide continuing support to the users
- ▶ Gather lessons learned from all teams
- ▶ Should be technically and organizationally integrated into the software development lifecycle



Test Tools

Summary and Keywords
explained

Keywords

Test Automation

- ▶ The use of software to perform or support test activities.

Test Automation Strategy -

- ▶ A high-level plan to achieve long-term objectives of test automation under given boundary conditions.

Testing is a craft

With tools you can better do that craft

Testing is MUCH more than test automation

Focus on learning how to be a good tester first!

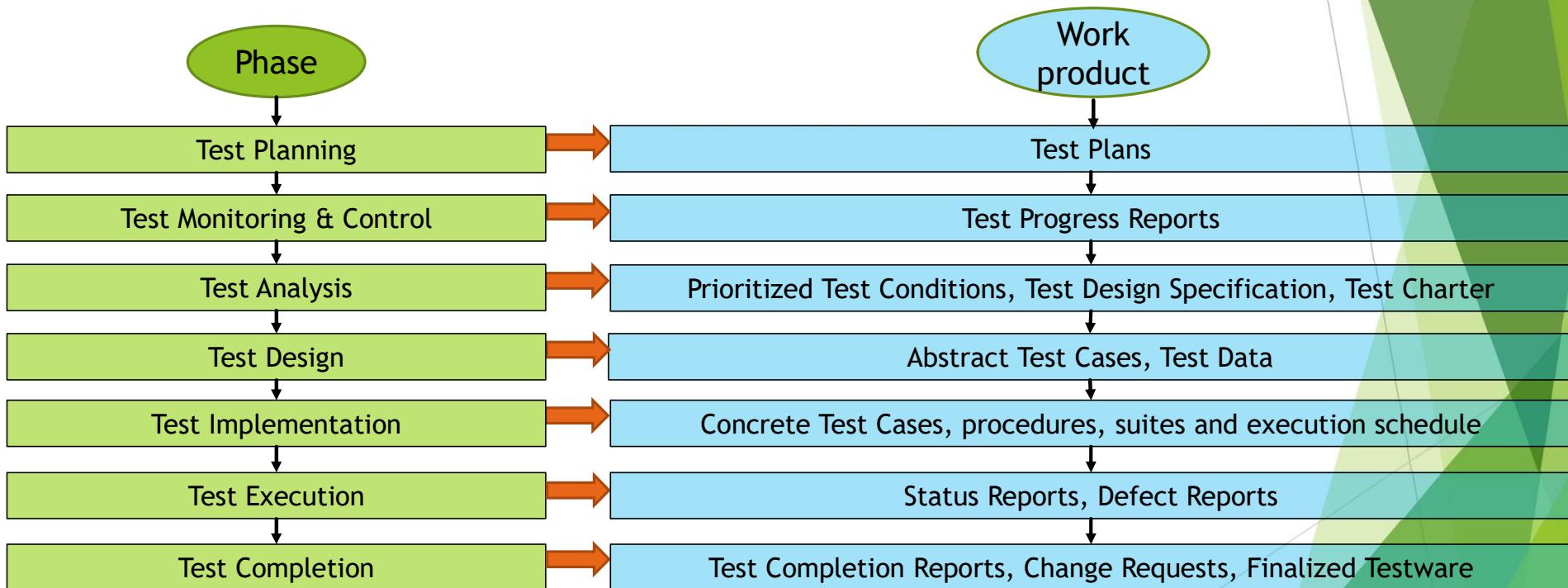




Summary on Industry Standard Test Work Products

ISO/IEC/IEEE 29119 Standard

Test Work products - Summary



Test Plan - ISO/IEC/IEEE 29119 Standard

- ▶ Context of testing
 - ▶ Type of Test Plan
 - ▶ Test Items
 - ▶ Test Scope
 - ▶ Assumptions and Constraints
 - ▶ Stakeholders
- ▶ Testing Lines of Communication
- ▶ Risk Register (Risks)
- ▶ Testing activities and estimates
- ▶ Staffing
 - ▶ Roles, tasks, responsibilities
 - ▶ Training needs
 - ▶ Hiring needs
- ▶ Schedule
- ▶ Test Strategy (of the project)
 - ▶ Test sub-Processes
 - ▶ Test Deliverables
 - ▶ Test Design Techniques
 - ▶ Test completion criteria
 - ▶ Metrics to be collected
 - ▶ Test Data Requirements
 - ▶ Test Environment Requirements
 - ▶ Retesting and regression testing
 - ▶ Suspension and resumption criteria
 - ▶ Deviations from the organizational test strategy

Test Design Specification - ISO/IEC/IEEE 29119 Standard

- ▶ Feature Sets
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Objective
 - ▶ Priority
 - ▶ Specific strategy
 - ▶ Traceability
- ▶ Test Conditions
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Description
 - ▶ Priority
 - ▶ Traceability

Test Case Specification - ISO/IEC/IEEE 29119 Standard

- ▶ Test Coverage Items
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Description
 - ▶ Priority
 - ▶ Traceability
- ▶ Test Cases
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Objective
 - ▶ Priority
 - ▶ Traceability
 - ▶ Preconditions
 - ▶ Inputs
 - ▶ Expected results
 - ▶ Actual results

Test Procedure Specification - ISO/IEC/IEEE 29119 Standard

- ▶ Test Sets
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Objective
 - ▶ Priority
 - ▶ Test Cases
- ▶ Test Procedures
 - ▶ Overview
 - ▶ Unique identifier
 - ▶ Objective
 - ▶ Priority
 - ▶ Start-up
 - ▶ Test cases
 - ▶ Relationship to other procedures
 - ▶ Stop & wrap-up



Test Data Requirements - ISO/IEC/IEEE 29119 Standard

- ▶ Overview
- ▶ Unique identifier
- ▶ Description
- ▶ Responsibility
- ▶ Period needed
- ▶ Resetting needed
- ▶ Archiving or disposal



Test Environment Requirements - ISO/IEC/IEEE 29119 Standard

- ▶ Overview
- ▶ Unique identifier
- ▶ Description
- ▶ Responsibility
- ▶ Period needed



Test Execution Log - ISO/IEC/IEEE 29119 Standard

- ▶ Unique identifier
- ▶ Time
- ▶ Description
- ▶ Impact

Incident Report - ISO/IEC/IEEE 29119 Standard

- ▶ Overview
- ▶ Timing information
- ▶ Originator
- ▶ Context
- ▶ Description of the incident
- ▶ Originator's assessment of the severity
- ▶ Originator's assessment of the impact
- ▶ Risk
- ▶ Status of the incident

Test Progress Report - ISO/IEC/IEEE 29119 Standard

- ▶ Reporting period
- ▶ Progress against the test plan
- ▶ Factors blocking progress
- ▶ Test Measures
- ▶ New and changed risks
- ▶ Planned testing

Test Completion Report - ISO/IEC/IEEE 29119 Standard

- ▶ Scope
- ▶ Testing performed
- ▶ Deviations from planned testing
- ▶ Test completion evaluation
- ▶ Factors that blocked progress
- ▶ Test measures
- ▶ Residual risks
- ▶ Test deliverables
- ▶ Reusable test assets
- ▶ Lessons learned

References

Standards

- ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering - Software testing - Part 1: Concepts and definitions
- ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering - Software testing - Part 2: Test Processes
- ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering - Software testing - Part 3: Test Documentation
- ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering - Software testing - Part 4: Test Techniques

ISTQB® Documents

- ISTQB® Glossary
- ISTQB® Foundation Level Overview V4.0
- ISTQB® Foundation Level Syllabus V4.0
- ISTQB® Exam Structure and Rules V4.0
- ISTQB® Sample Exams