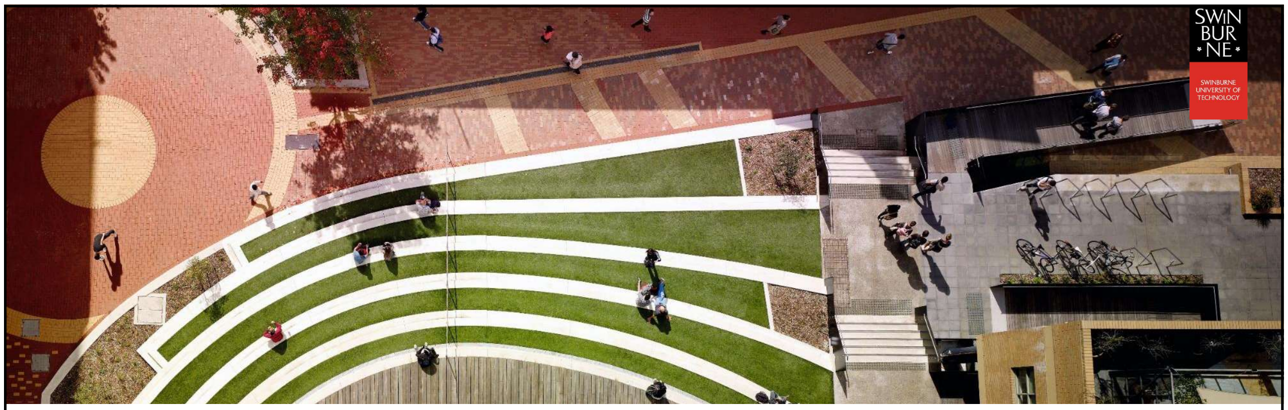


COS80022 – Software Quality and Testing

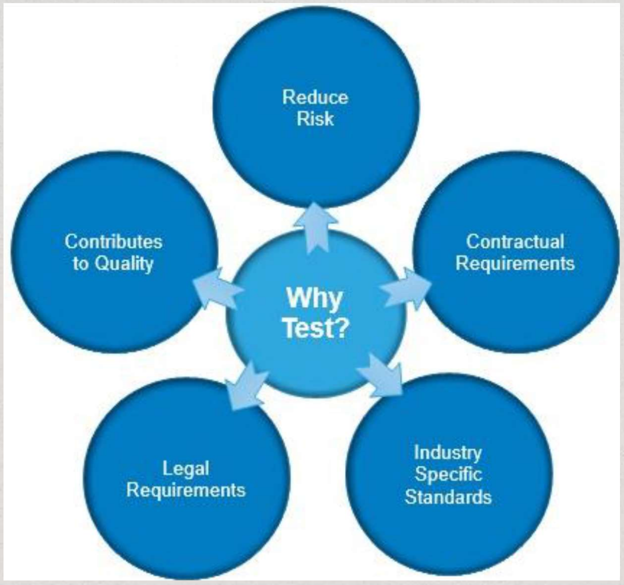
Week 5 – Testing Process, Levels and Types

Outline

- Testing process
- Testing in software development lifecycle
- Testing levels
- Testing types

Testing Process

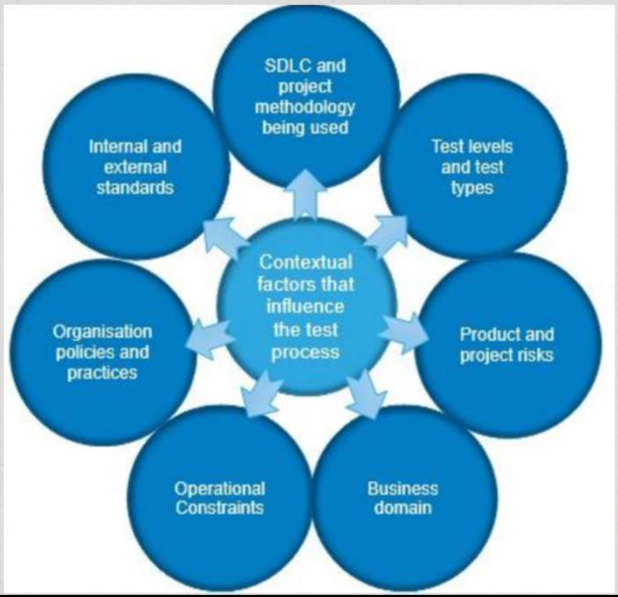
Software Testing



Testing Process

- There is no one universal software testing process, but there are common sets of activities without which testing will be less likely to achieve its established objectives. These sets of test activities are a testing process.
- Which test activities are involved in this testing process, how these activities are implemented, and when these activities occur may be discussed in an organisation's test strategy.

Testing Process in Context

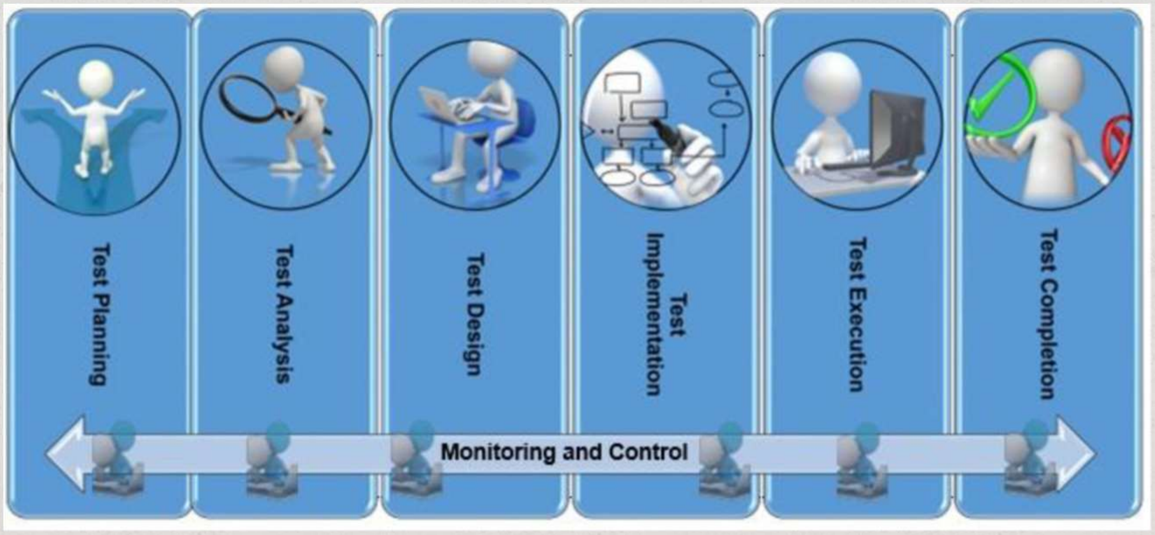

SDLC and project methodology being used
Internal and external standards
Test levels and test types
Contextual factors that influence the test process
Organisation policies and practices
Product and project risks
Operational Constraints
Business domain

General Aspects of Testing Process

- It is very useful if the test basis has measurable coverage criteria defined.
- Coverage criteria can act effectively as key performance indicators (KPI's)
  - Test activities and tasks
  - Test work products
  - Traceability between the test basis and test work products

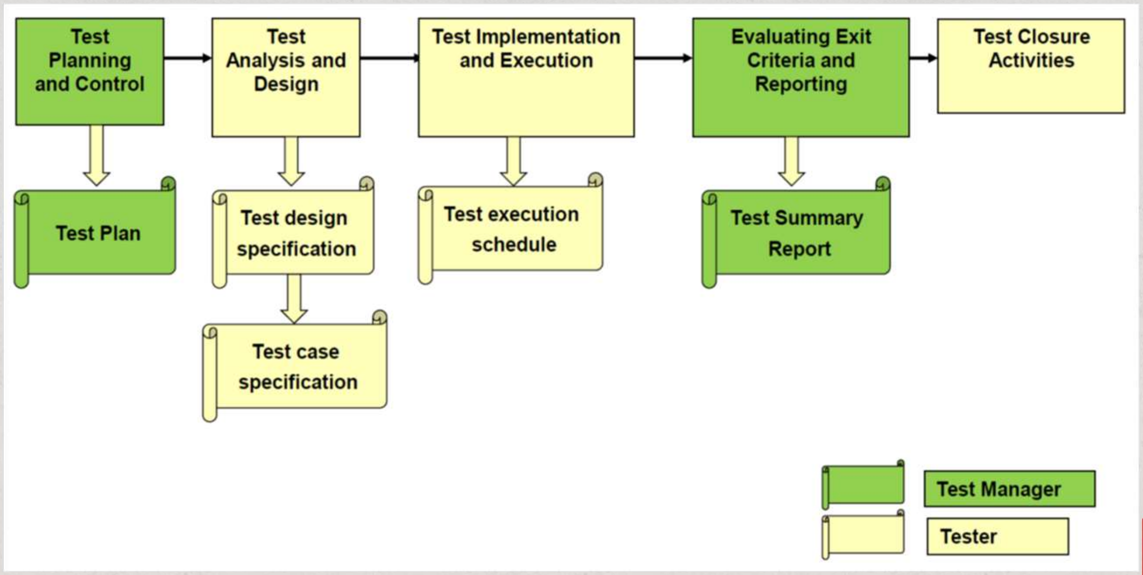
7

Test Activities and Tasks



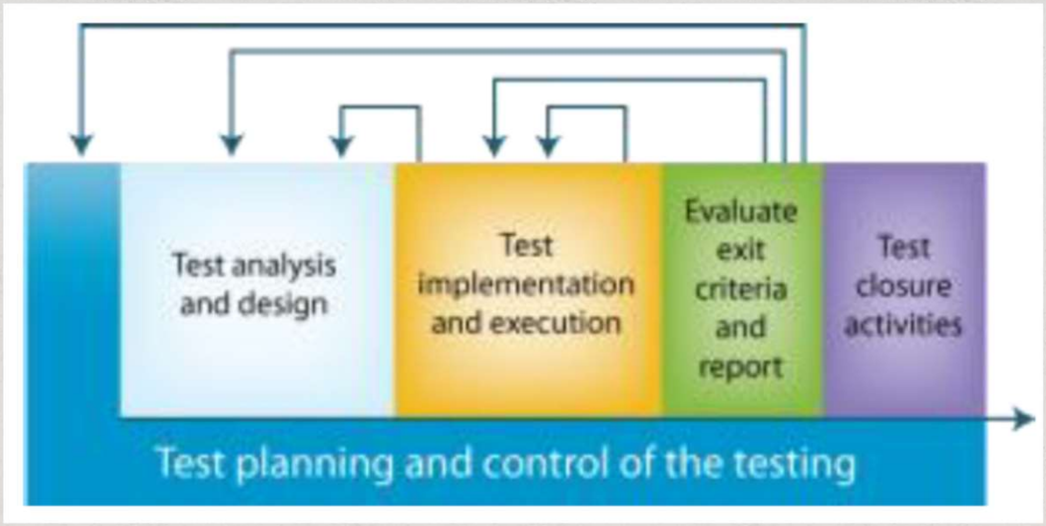
8

Test Process



KNOW ING

Test Process


Test analysis and design
Test implementation and execution
Evaluate exit criteria and report
Test closure activities
Test planning and control of the testing

KNOW ING

Test Planning

- Involves activities that define the objectives of testing and the approach for meeting test objectives within constraints imposed by the context
  - Specifying suitable test techniques and tasks
  - Formulating a test schedule for meeting a deadline

– To be covered in Week 6’s lecture

KNOW ING

Test Monitoring and Control

- Monitoring: the on-going comparison of actual progress against the test plan using any testing monitoring metrics defined in the test plan
- Control: taking actions necessary to meet the objectives of the test plan (which may be updated over time)
- Supported by the evaluation of **exit criteria**, which are referred to as the **definition of done** in some lifecycles.

KNOW ING

Example of Evaluating Exit Criteria

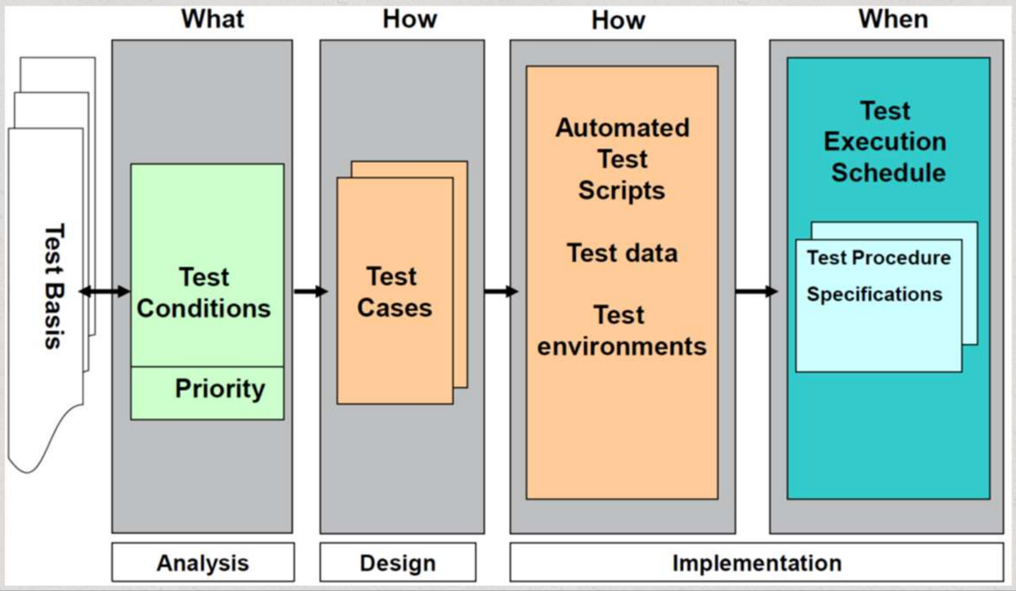
- Checking test results and logs against specified coverage criteria
- Assessing the level of component or system quality based on test results and logs
- Determining if more tests are needed

Test progress against the plan is communicated to stakeholders in test progress reports, including deviations from the plan and information to support any decision to stop testing

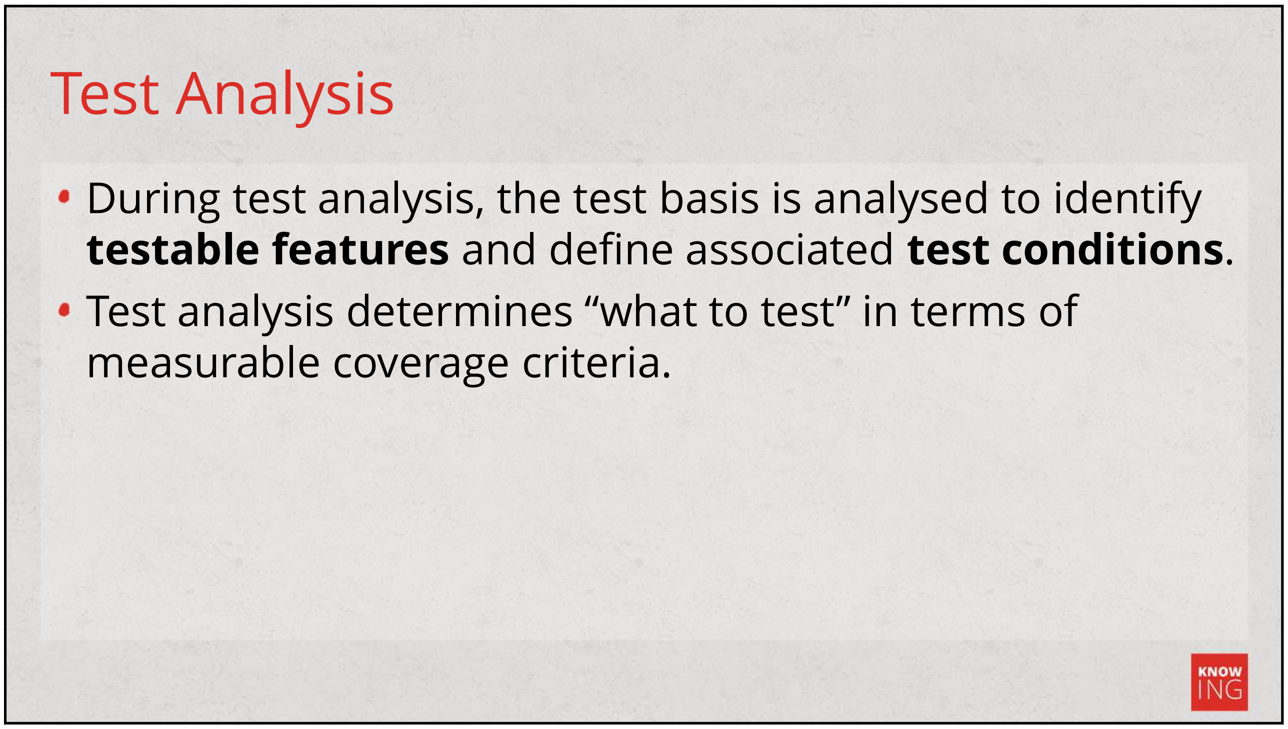
– To be covered in Week 6's Lecture

13

Test Analysis, Design and Implementation



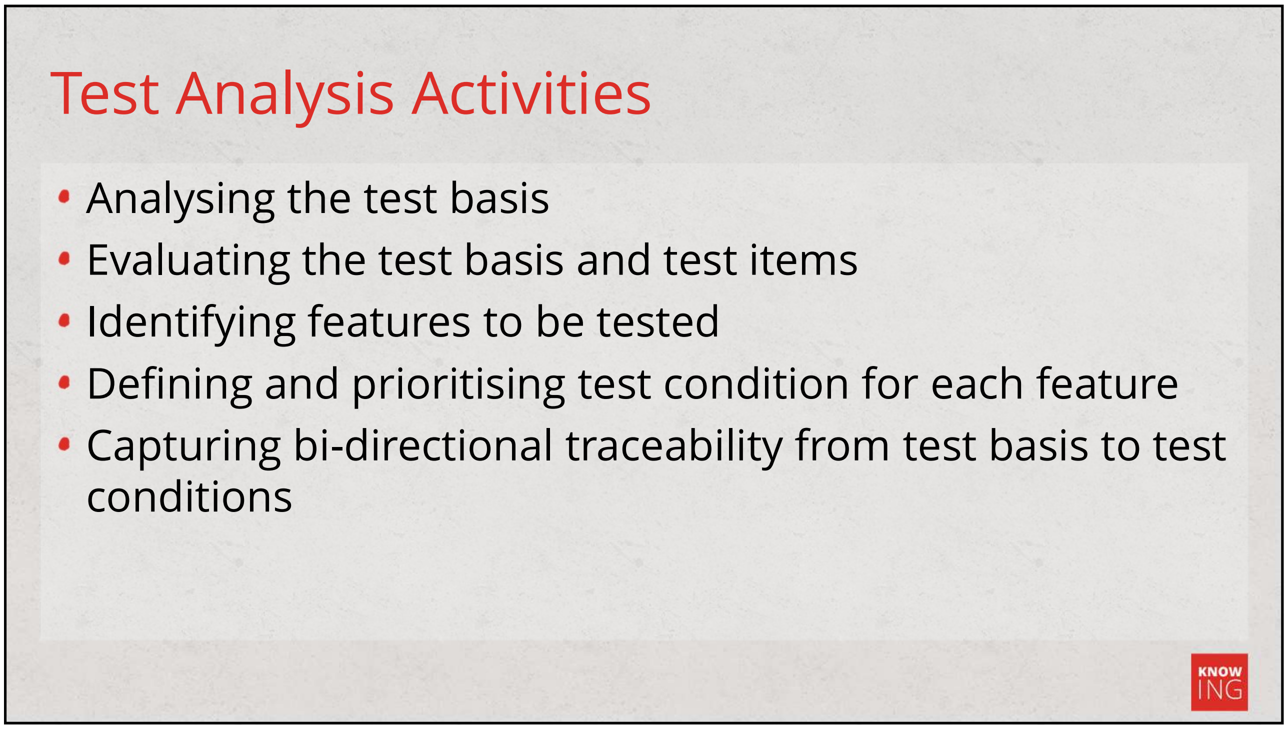
14



Test Analysis

- During test analysis, the test basis is analysed to identify **testable features** and define associated **test conditions**.
- Test analysis determines “what to test” in terms of measurable coverage criteria.

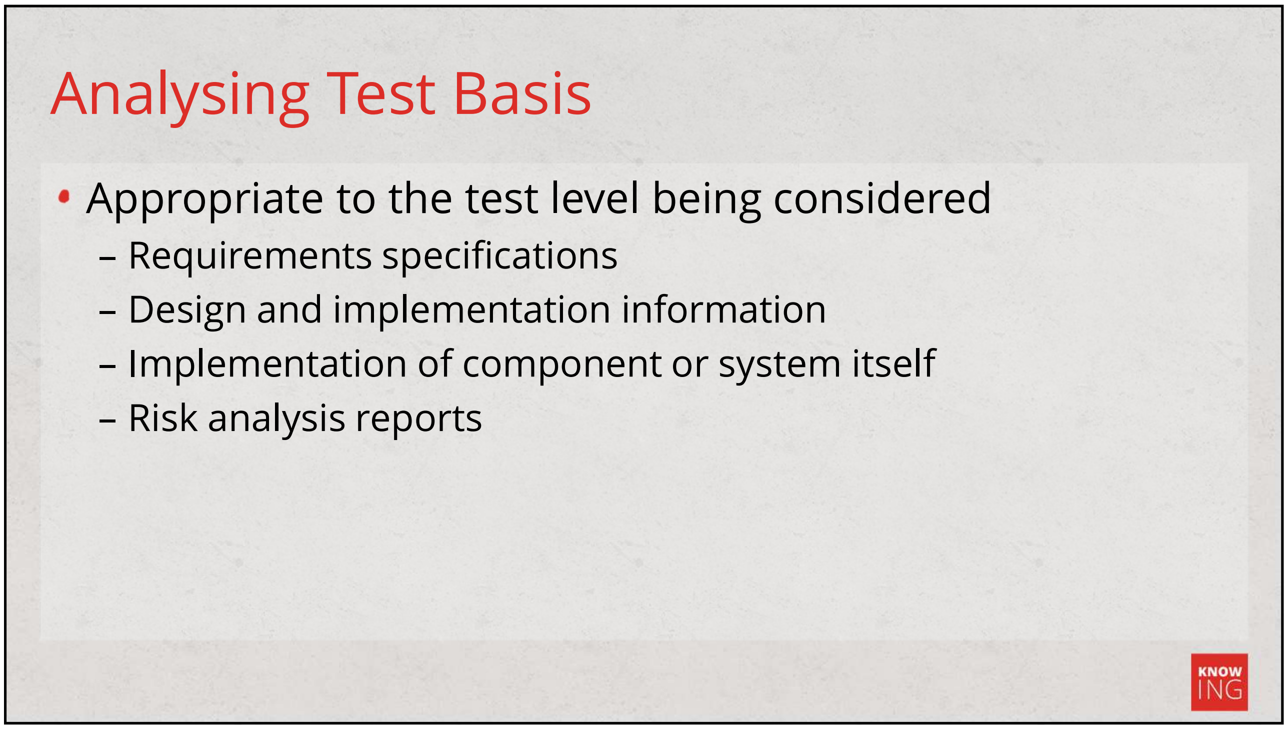
15



Test Analysis Activities

- Analysing the test basis
- Evaluating the test basis and test items
- Identifying features to be tested
- Defining and prioritising test condition for each feature
- Capturing bi-directional traceability from test basis to test conditions

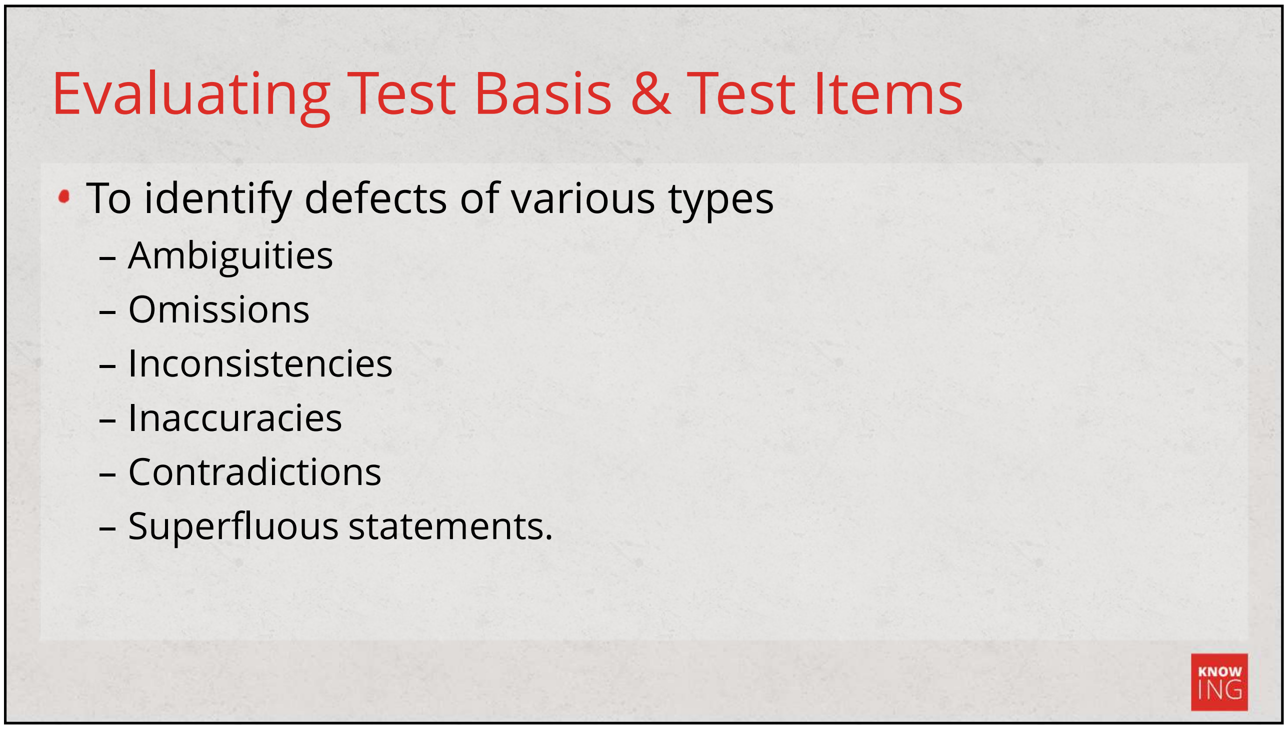
16



Analysing Test Basis

- Appropriate to the test level being considered
  - Requirements specifications
  - Design and implementation information
  - Implementation of component or system itself
  - Risk analysis reports

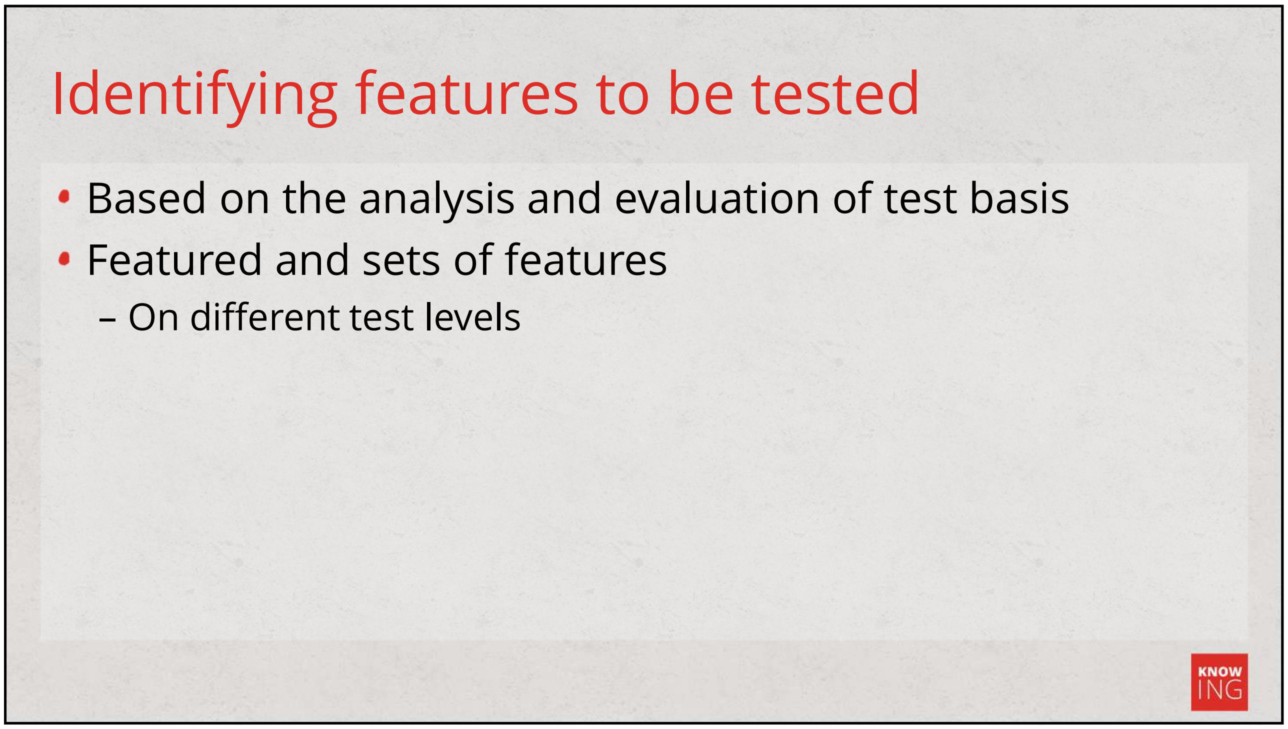
17



Evaluating Test Basis & Test Items

- To identify defects of various types
  - Ambiguities
  - Omissions
  - Inconsistencies
  - Inaccuracies
  - Contradictions
  - Superfluous statements.

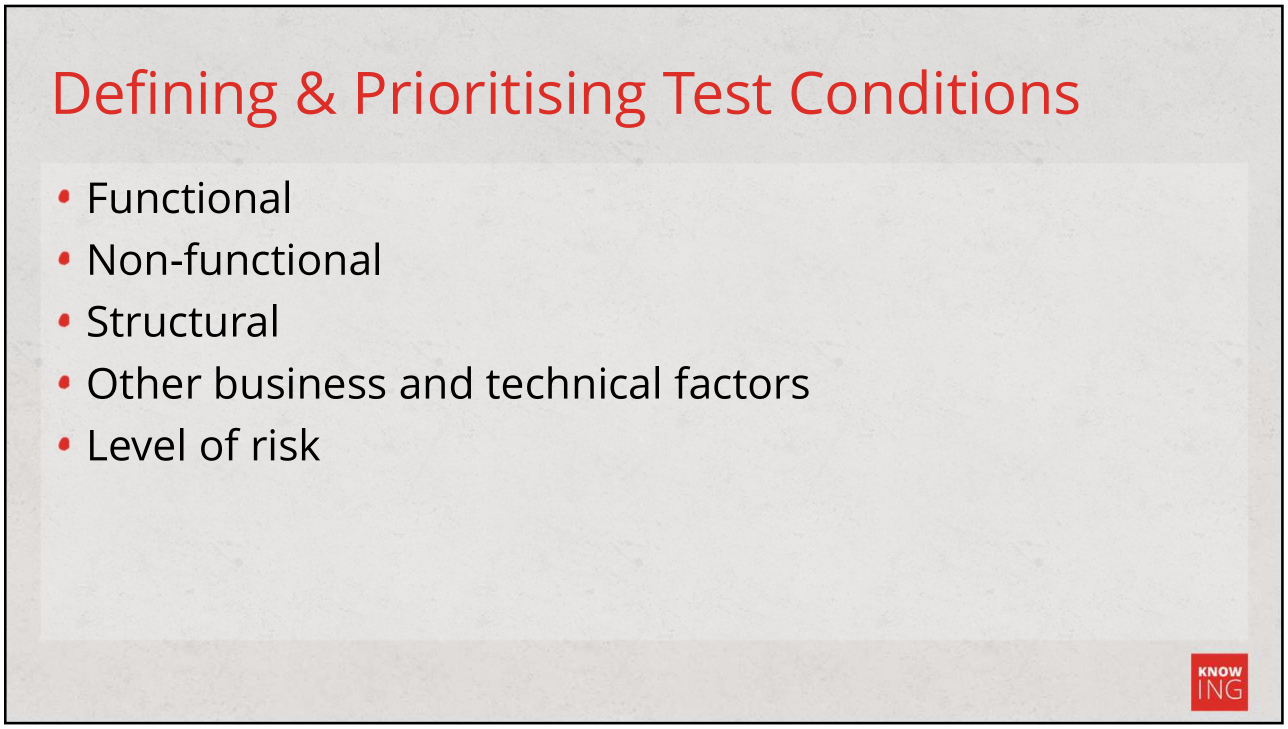
18



Identifying features to be tested

- Based on the analysis and evaluation of test basis
- Featured and sets of features
  - On different test levels

19



Defining & Prioritising Test Conditions

- Functional
- Non-functional
- Structural
- Other business and technical factors
- Level of risk

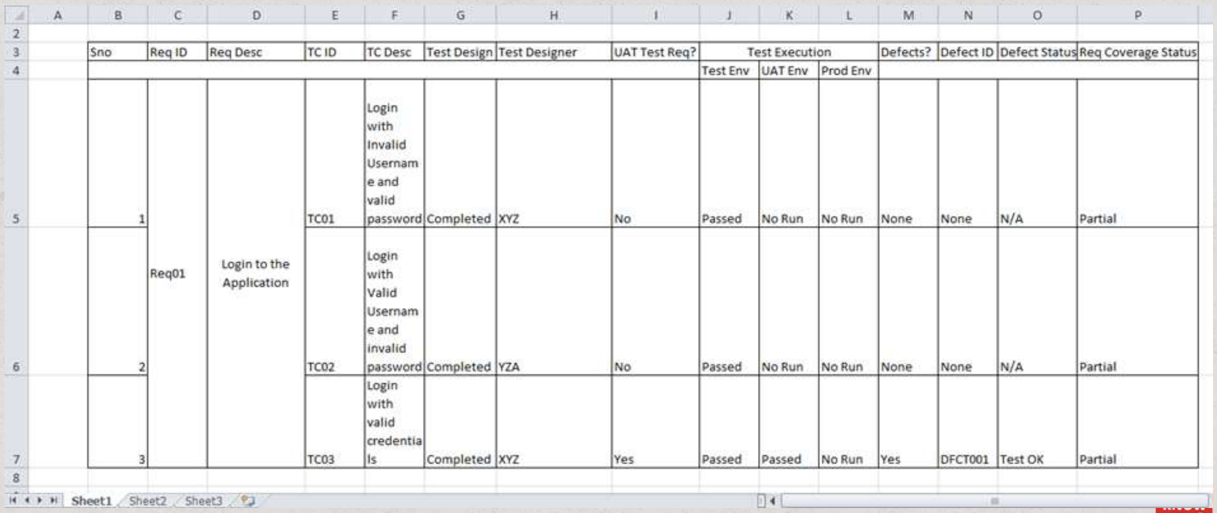
20

Test Condition

- “An item or event of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute, or structural element.”
  - ‘To demonstrate that a user with a valid User ID and Password can log into the application’

21

Bi-Directional Traceability



Sno	Req ID	Req Desc	TC ID	TC Desc	Test Design	Test Designer	UAT Test Req?	Test Execution			Defects?	Defect ID	Defect Status	Req Coverage Status
								Test Env	UAT Env	Prod Env				
1	Req01	Login to the Application	TC01	Login with Invalid Username and valid password	Completed	XYZ	No	Passed	No Run	No Run	None	None	N/A	Partial
2			TC02	Login with Valid Username and invalid password	Completed	YZA	No	Passed	No Run	No Run	None	None	N/A	Partial
3			TC03	Login with valid credentials	Completed	XYZ	Yes	Passed	Passed	No Run	Yes	DFCT001	Test OK	Partial

Sheet1 Sheet2 Sheet3

22

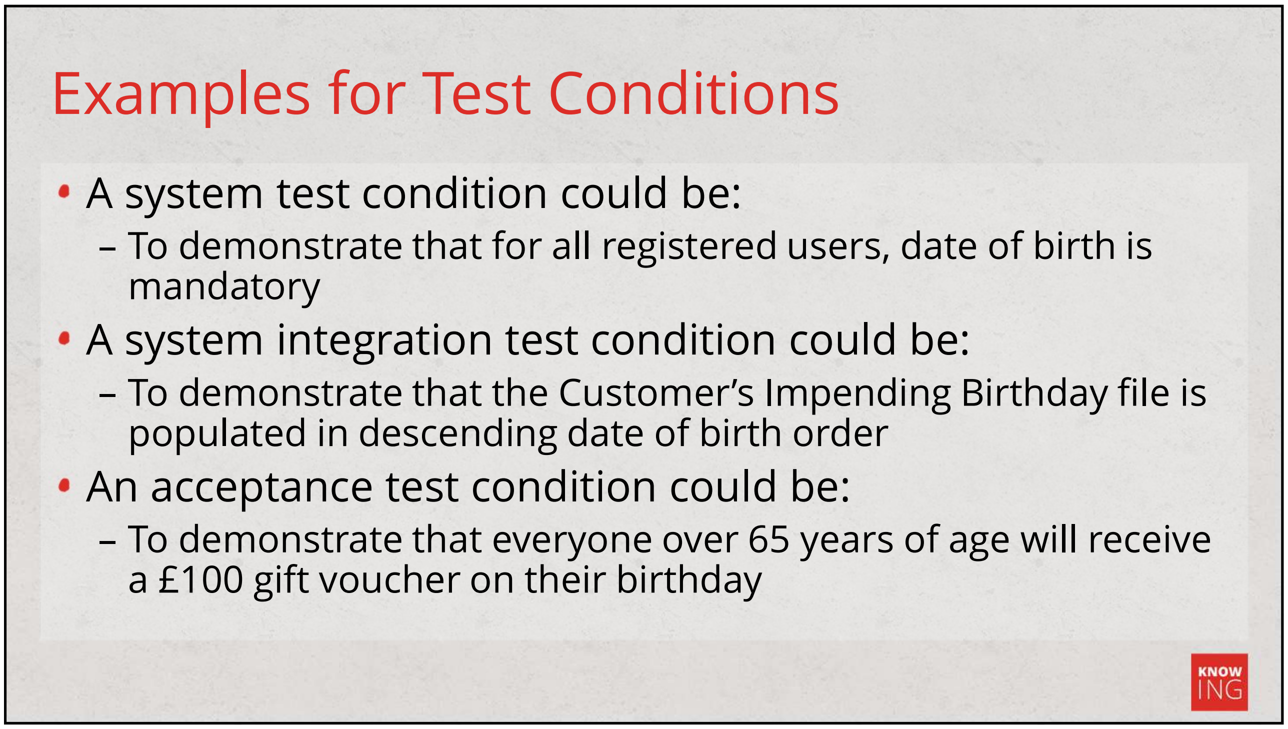
Test Analysis Techniques

- Black-box, white-box, and experience-based
- Produce test conditions which are to be used as test objectives in test charters. Test charters are typical work products in some types of experience-based testing
- Behaviour driven development (BDD) and acceptance test driven development (ATDD), which involve generating test conditions and test cases from user stories and acceptance criteria may also be used

Examples for Test Conditions

- Example: the format of a date of birth is a key attribute of a customer loyalty process.
- At component level*, the test condition could be:
  - To demonstrate that the date of birth is stored as CCYYMMDD
- At component integration, the test condition could be:
  - To demonstrate that the date of birth passed between the two components is in the format CCYYMMDD

* Test levels will be covered later in this lecture

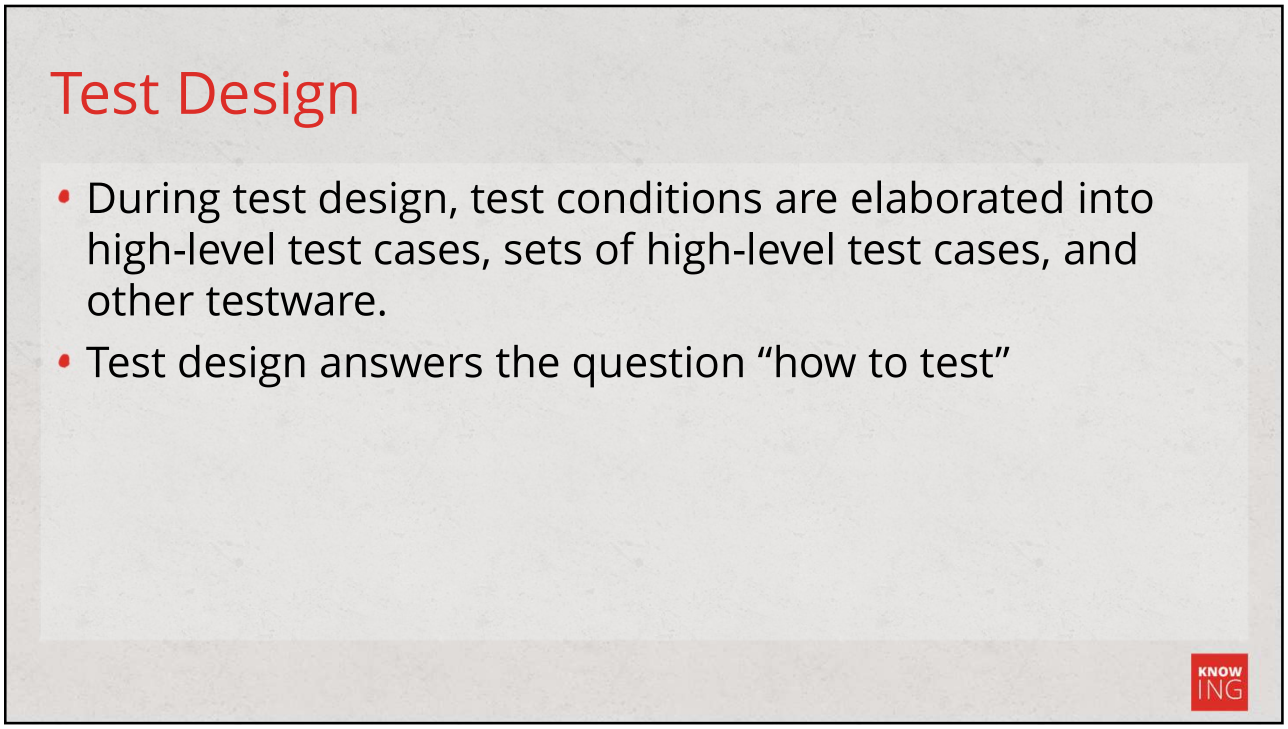


Examples for Test Conditions

- A system test condition could be:
  - To demonstrate that for all registered users, date of birth is mandatory
- A system integration test condition could be:
  - To demonstrate that the Customer's Impending Birthday file is populated in descending date of birth order
- An acceptance test condition could be:
  - To demonstrate that everyone over 65 years of age will receive a £100 gift voucher on their birthday

KNOW ING

25



Test Design

- During test design, test conditions are elaborated into high-level test cases, sets of high-level test cases, and other testware.
- Test design answers the question "how to test"

KNOW ING

26

Test Design Activities

Designing and prioritising test cases

Identifying necessary test data

Designing the test environment

Identifying any required infrastructure and tools

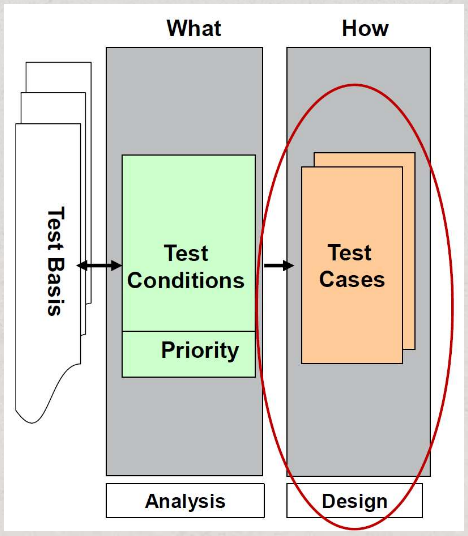
Capturing bi-directional tarceability from test basis to test cases

KNOW ING

27

Test Design

- For each test condition:
  - Determine whether it falls into an area suitable for high-level (logical) or low-level (concrete) test cases
  - Determine which test design techniques are most appropriate to employ for test case design
  - Create Test Case(s) to exercise the condition



KNOW ING

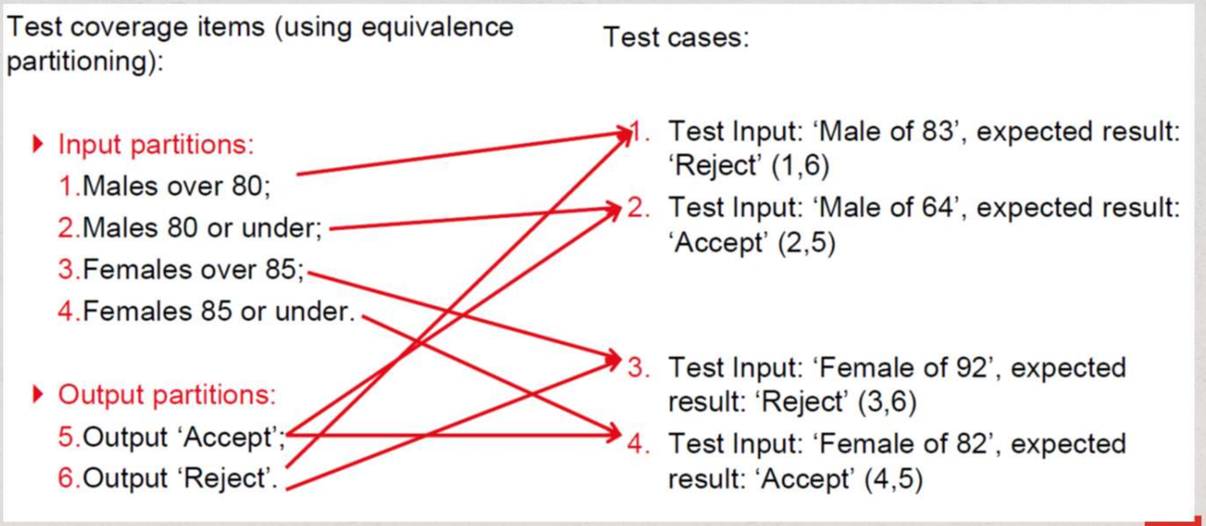
28

Example for Test Design

- Test conditions:
  1. Reject male insurance applicants over the age of 80 years on day of application;
  2. Reject female insurance applicants over the age of 85 years on day of application;
- Test coverage items (using equivalence partitioning*):
  - Input partitions:
    1. Males over 80;
    2. Males 80 or under;
    3. Females over 85;
    4. Females 85 or under.
  - Output partitions:
    5. Output 'Accept';
    6. Output 'Reject'.

* Equivalence partitioning will be studies in Week 9's lecture

Example for Test Design

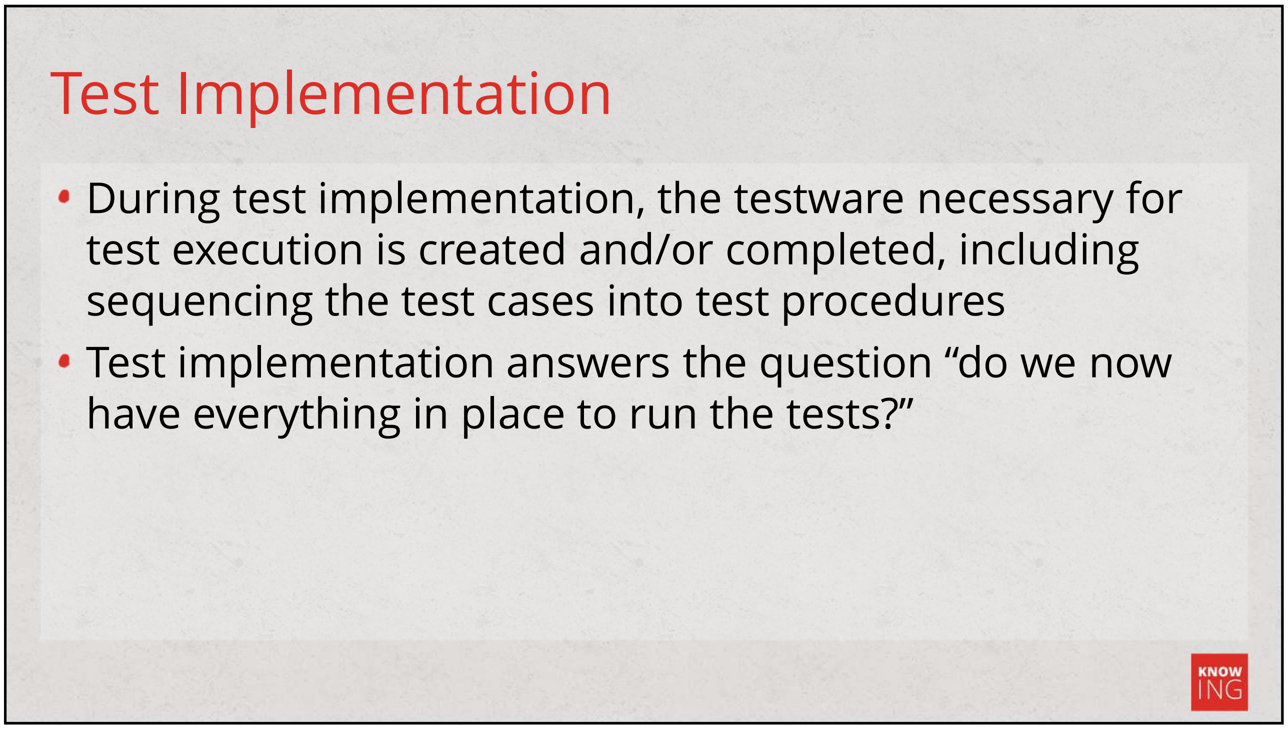


Test coverage items (using equivalence partitioning):

- Input partitions:
  1. Males over 80;
  2. Males 80 or under;
  3. Females over 85;
  4. Females 85 or under.
- Output partitions:
  5. Output 'Accept';
  6. Output 'Reject'.

Test cases:

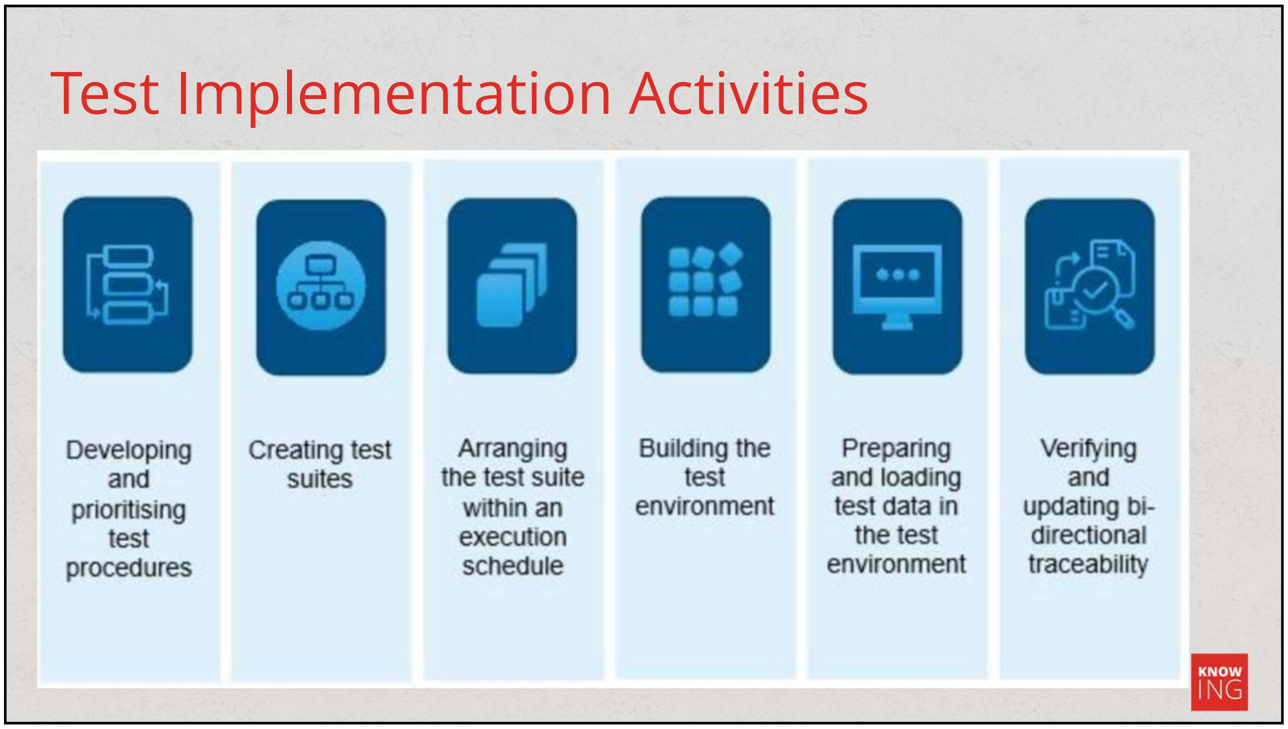
1. Test Input: 'Male of 83', expected result: 'Reject' (1,6)
2. Test Input: 'Male of 64', expected result: 'Accept' (2,5)
3. Test Input: 'Female of 92', expected result: 'Reject' (3,6)
4. Test Input: 'Female of 82', expected result: 'Accept' (4,5)



Test Implementation

- During test implementation, the testware necessary for test execution is created and/or completed, including sequencing the test cases into test procedures
- Test implementation answers the question “do we now have everything in place to run the tests?”

31



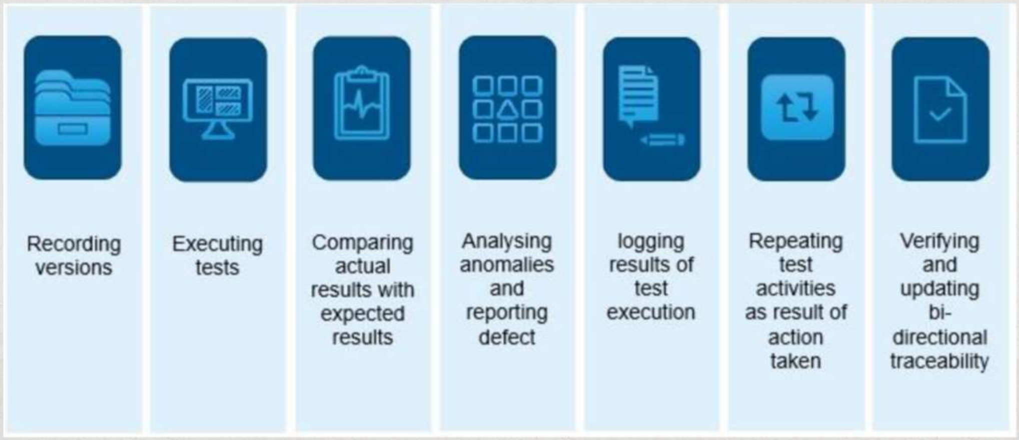
Test Implementation Activities

Developing and prioritising test procedures	Creating test suites	Arranging the test suite within an execution schedule	Building the test environment	Preparing and loading test data in the test environment	Verifying and updating bi-directional traceability

32

Test Execution

- During test execution, test suites are run in accordance with the test execution schedule


Recording versions
Executing tests
Comparing actual results with expected results
Analysing anomalies and reporting defect
logging results of test execution
Repeating test activities as result of action taken
Verifying and updating bi-directional traceability

KNOW ING

33

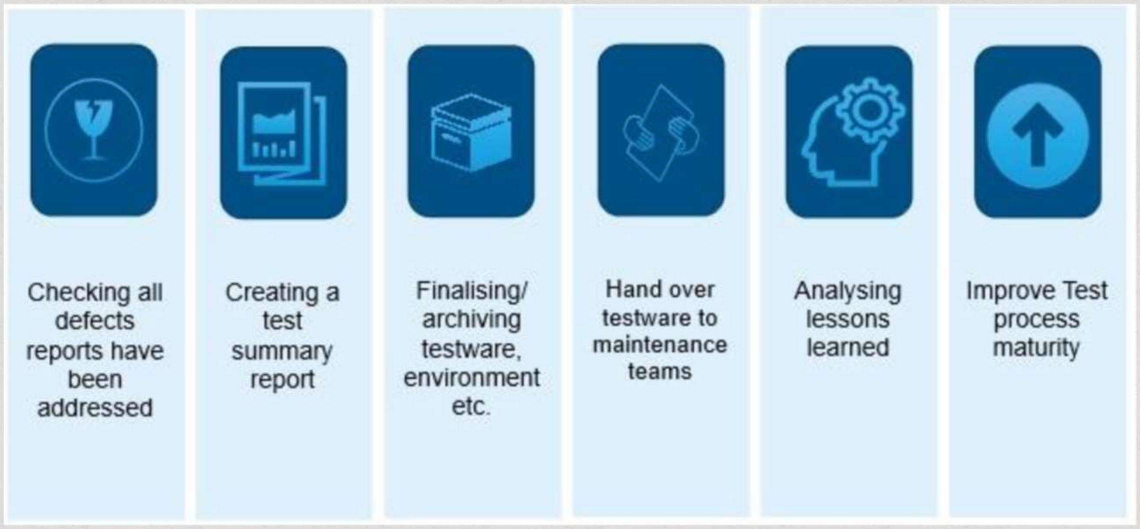
Test Completion

- Collect data from completed test activities to consolidate experience, testware, and any other relevant information
- Occur at project milestones

KNOW ING

34

Test Completion Activities

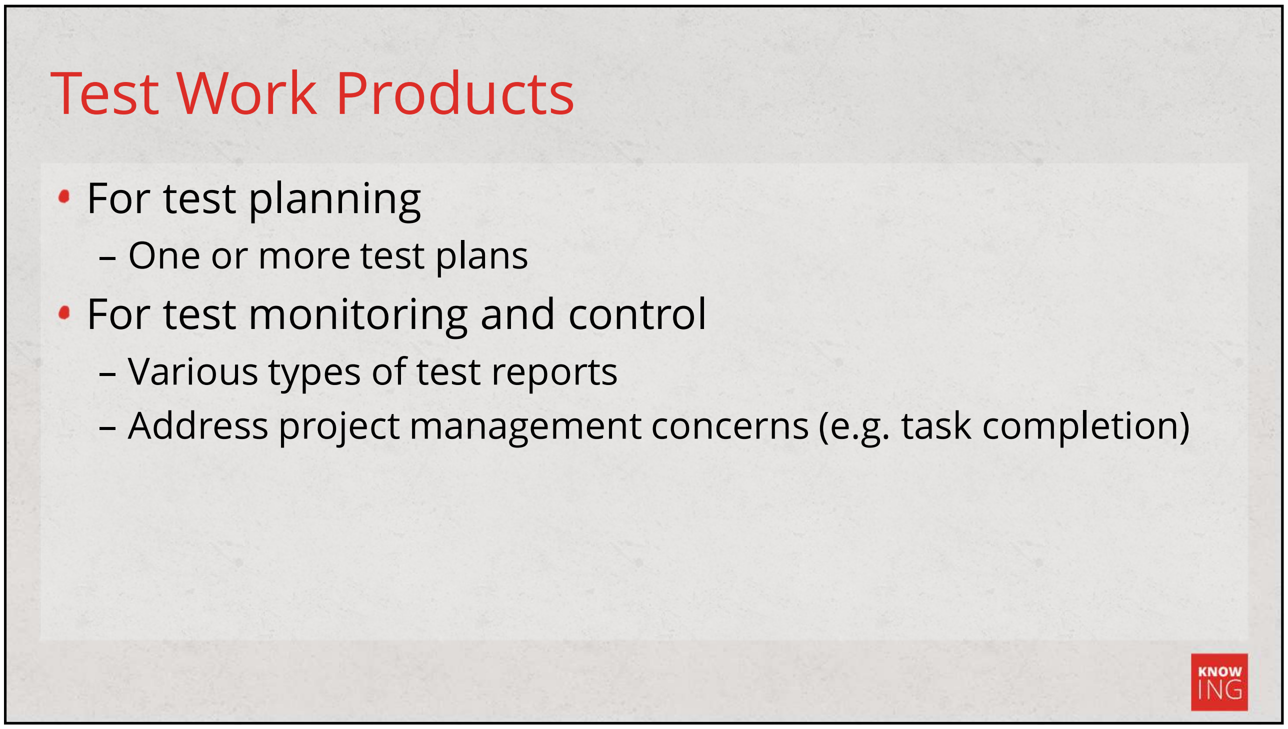

Checking all defects reports have been addressed
Creating a test summary report
Finalising/ archiving testware, environment etc.
Hand over testware to maintenance teams
Analysing lessons learned
Improve Test process maturity

35

Test Work Products

- Created as part of the test process
- There can be significant variation in the types of work products created depending on an organisations test process implementation

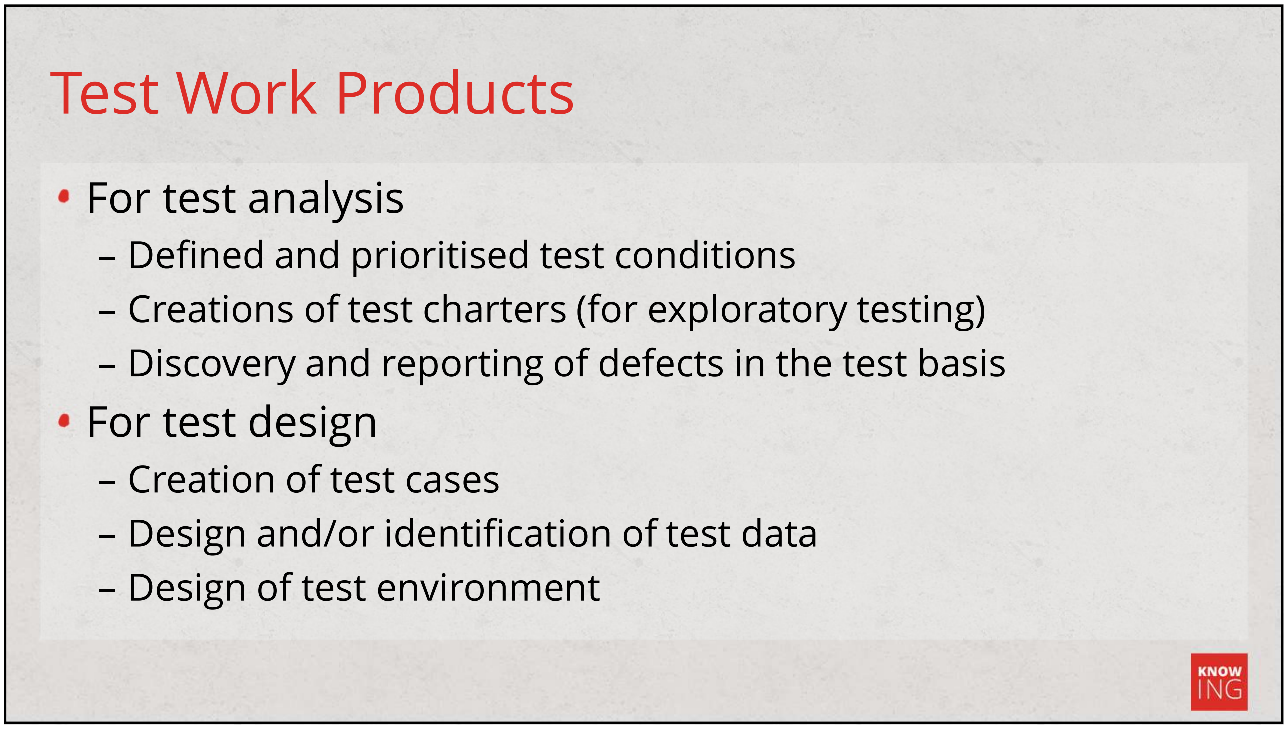
36



Test Work Products

- For test planning
  - One or more test plans
- For test monitoring and control
  - Various types of test reports
  - Address project management concerns (e.g. task completion)

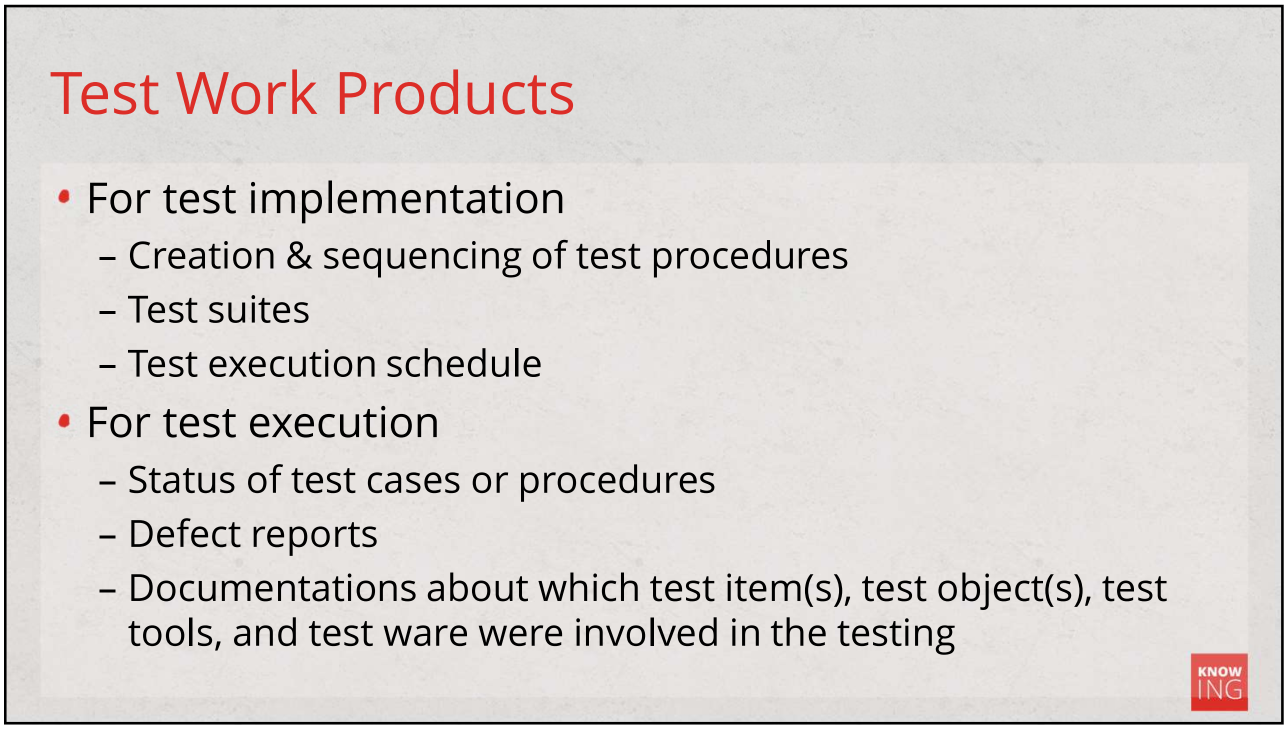
37



Test Work Products

- For test analysis
  - Defined and prioritised test conditions
  - Creations of test charters (for exploratory testing)
  - Discovery and reporting of defects in the test basis
- For test design
  - Creation of test cases
  - Design and/or identification of test data
  - Design of test environment

38

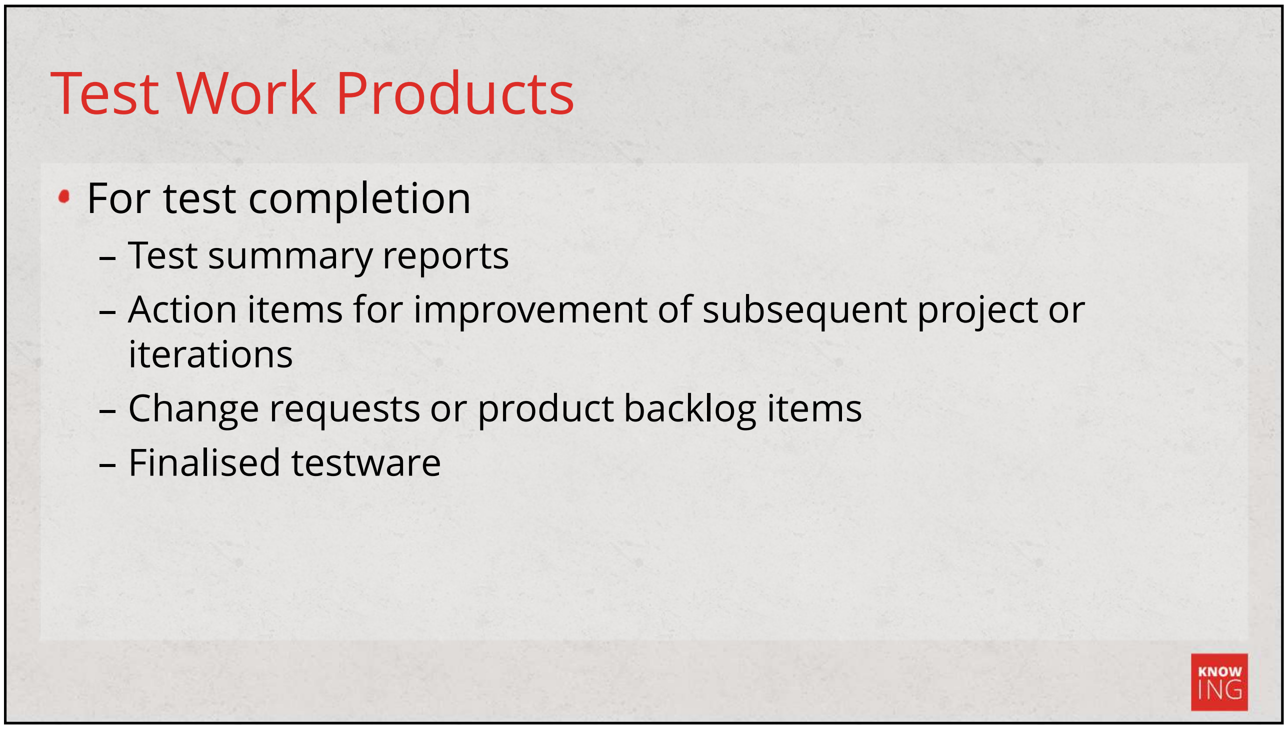


Test Work Products

- For test implementation
  - Creation & sequencing of test procedures
  - Test suites
  - Test execution schedule
- For test execution
  - Status of test cases or procedures
  - Defect reports
  - Documentations about which test item(s), test object(s), test tools, and test ware were involved in the testing

KNOW ING

39



Test Work Products

- For test completion
  - Test summary reports
  - Action items for improvement of subsequent project or iterations
  - Change requests or product backlog items
  - Finalised testware

KNOW ING

40

Traceability – Test Basis & Test Work Products

- To implement effective test monitoring and control, it is important to establish and maintain traceability throughout the test process.
- Good traceability supports: Analysing impact of changes; making testing auditable, meeting IT governance criteria; improving understandability of test progress/summary reports; relating technical aspects of testing to stakeholders; providing information to assess product quality, process capability, and project progress against business goals

KNOW ING

41

SWIN BUR NE

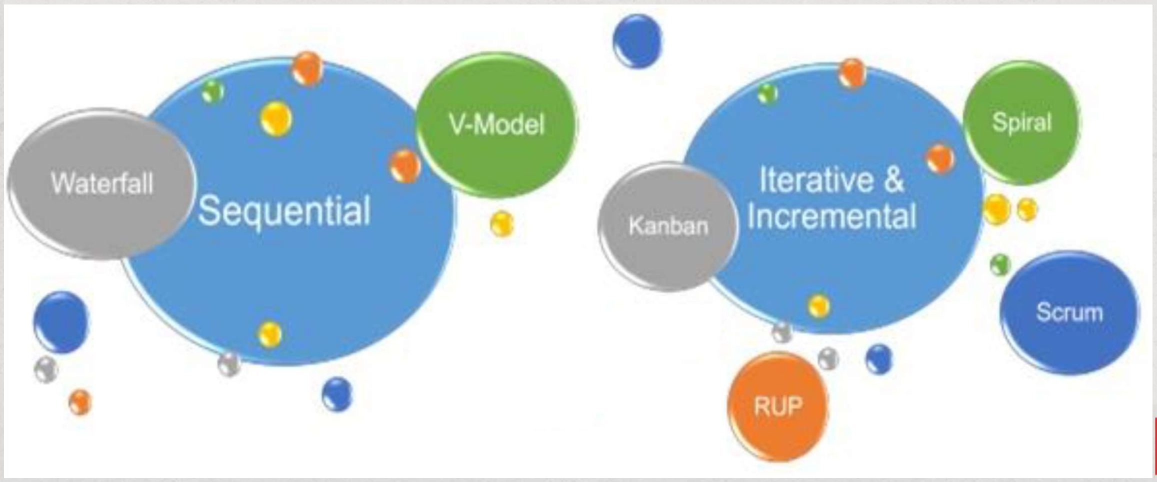
SWINBURNE UNIVERSITY OF TECHNOLOGY

Test Levels

42

Testing in SDLC

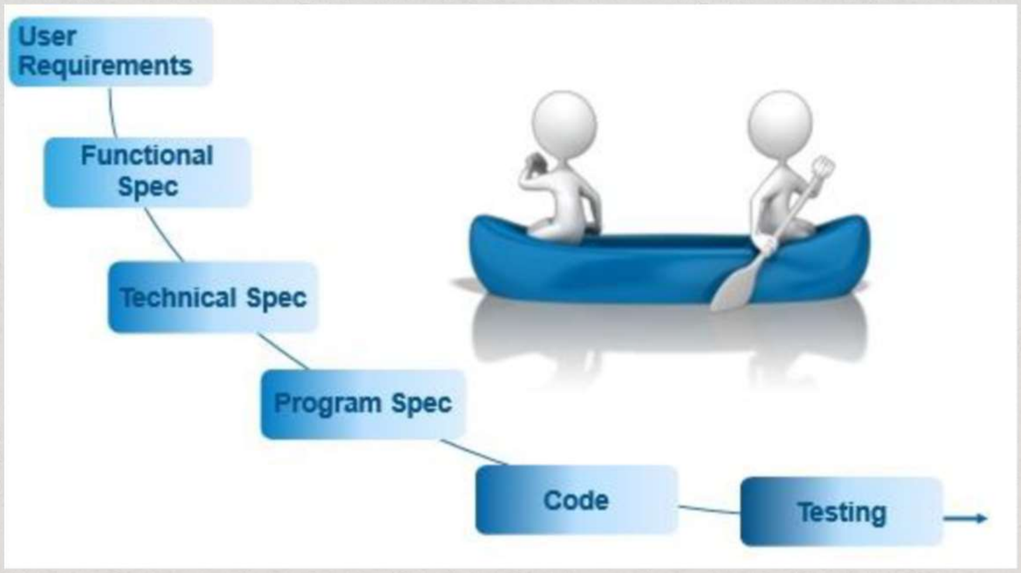
- Revisit SDLC



KNOW ING

43

Sequential Development – Waterfall Model

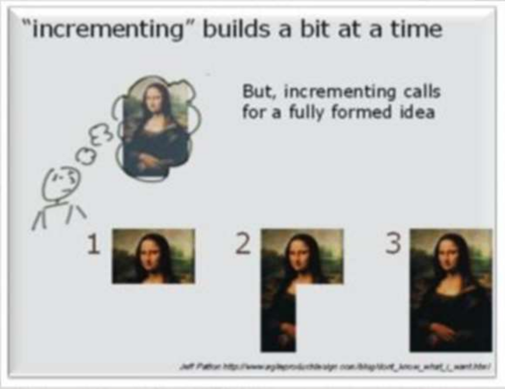


KNOW ING

44

Incremental Development Models

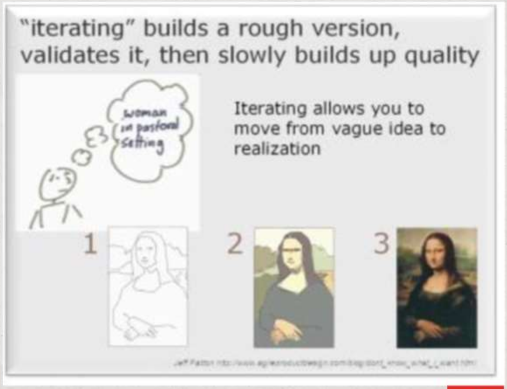
- Establishing requirements, designing, building and testing a system in pieces
- Features grow incrementally
- Size of feature increments vary


"incrementing" builds a bit at a time
But, incrementing calls for a fully formed idea
1
2
3
Jeff Patton http://www.agileproductdesign.com/blog/dont_know_what_i_want.html

KNOW ING

Iterative Development Models

- Occur when groups of features are specified, designed, built, and tested together in a series of cycles, often of a fixed duration
- May involve changes to features developed in earlier iterations, along with changes in project scope

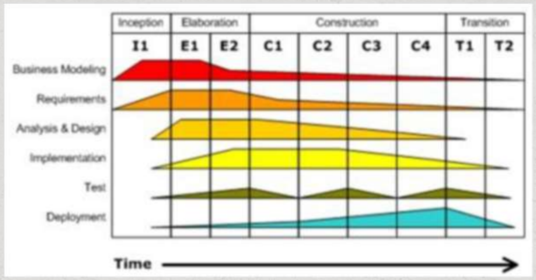

"iterating" builds a rough version, validates it, then slowly builds up quality
woman in pastoral setting
Iterating allows you to move from vague idea to realization
1
2
3

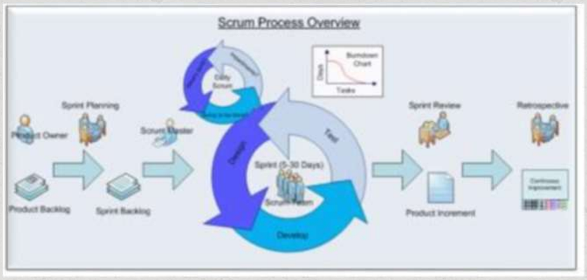
KNOW ING

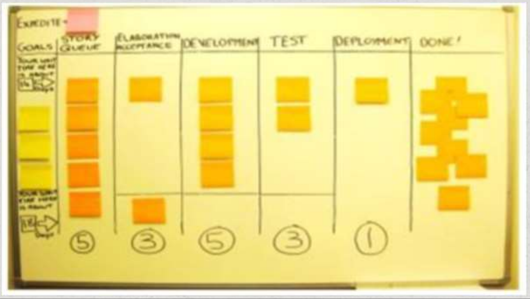
Incremental & Iterative Development

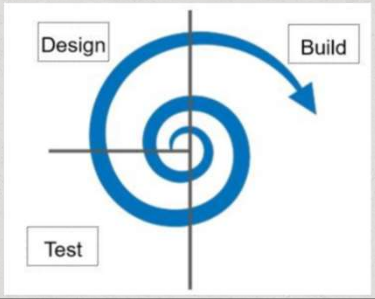
- Overlapping and iterating test levels
- Continuous delivery or continuous deployment
  - – Significant automation of multiple test levels
- Self-organising teams
- Released to end-users on a feature-by-feature or iteration-by-iteration basis, or more traditional major-release fashion
- Regression testing: increasingly important
- Deliver working software in weeks or days

Incremental and Iterative Development









Combining Test Levels & Test Activities

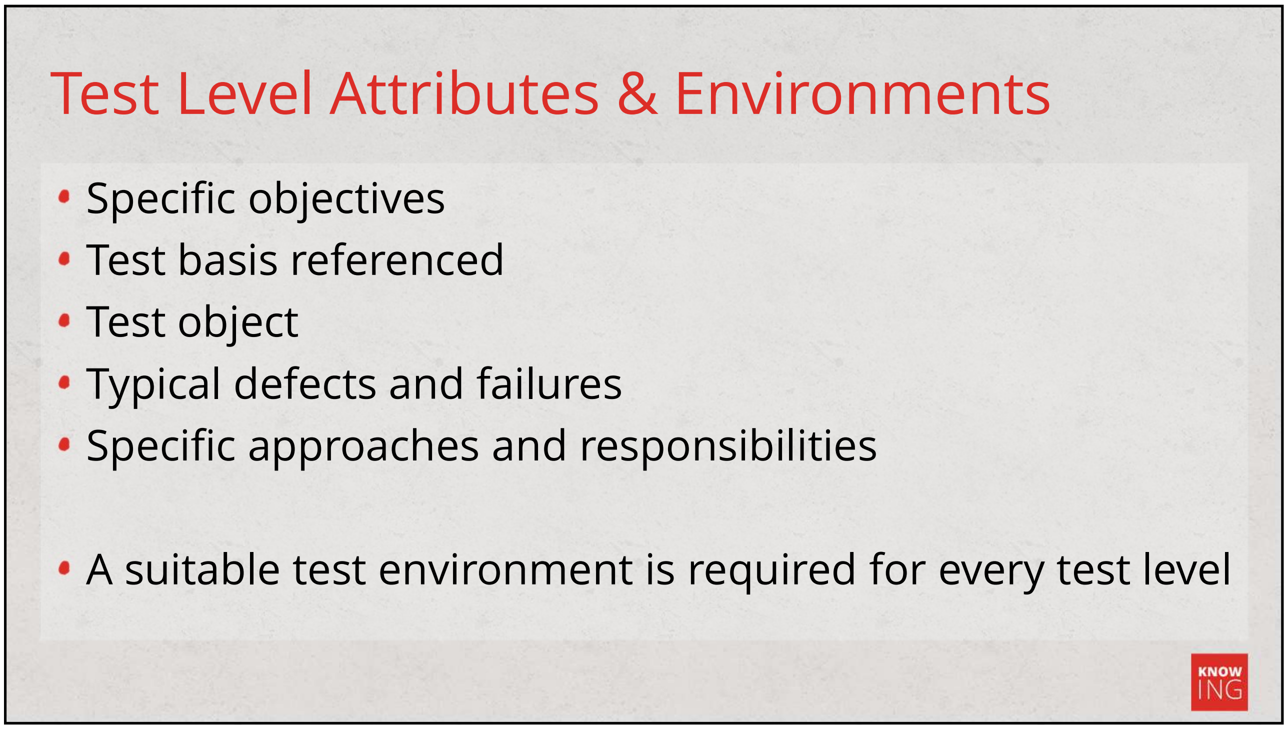
- Depending on the context of the project, it may be necessary to combine or reorganise test levels and/or test activities
- For example, for integration of a commercial-off-the-shelf (COTS) software product into a large system
  - Purchaser may perform interoperability testing at:
    - system integration test level (e.g., integration to the infrastructure and other systems)
    - acceptance test level (e.g., functional and non-functional, along with user acceptance testing and operational acceptance testing)

49

Test Levels

- Groups of test activities that are organised and managed together and are related to other activities within SDLC
  - Component testing
  - Integration testing
  - System testing
  - Acceptance testing

50

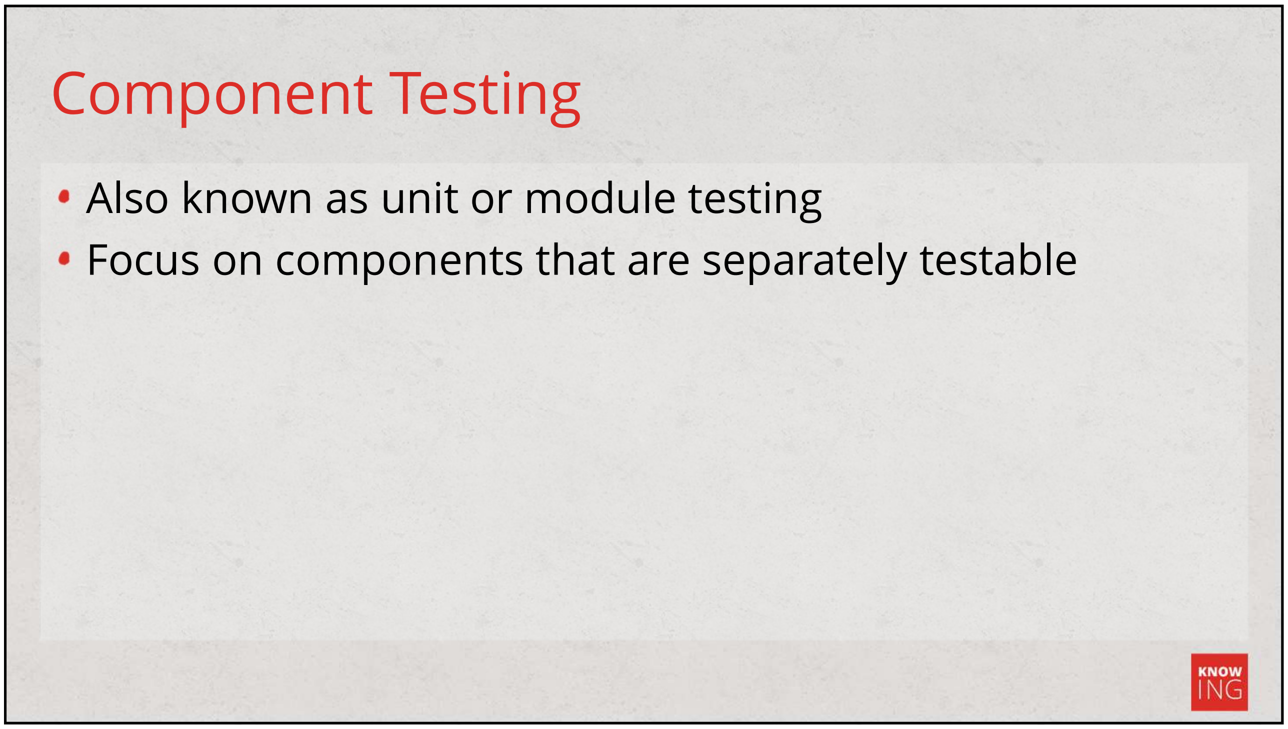


Test Level Attributes & Environments

- Specific objectives
- Test basis referenced
- Test object
- Typical defects and failures
- Specific approaches and responsibilities

- A suitable test environment is required for every test level

51



Component Testing

- Also known as unit or module testing
- Focus on components that are separately testable

52

Component Testing

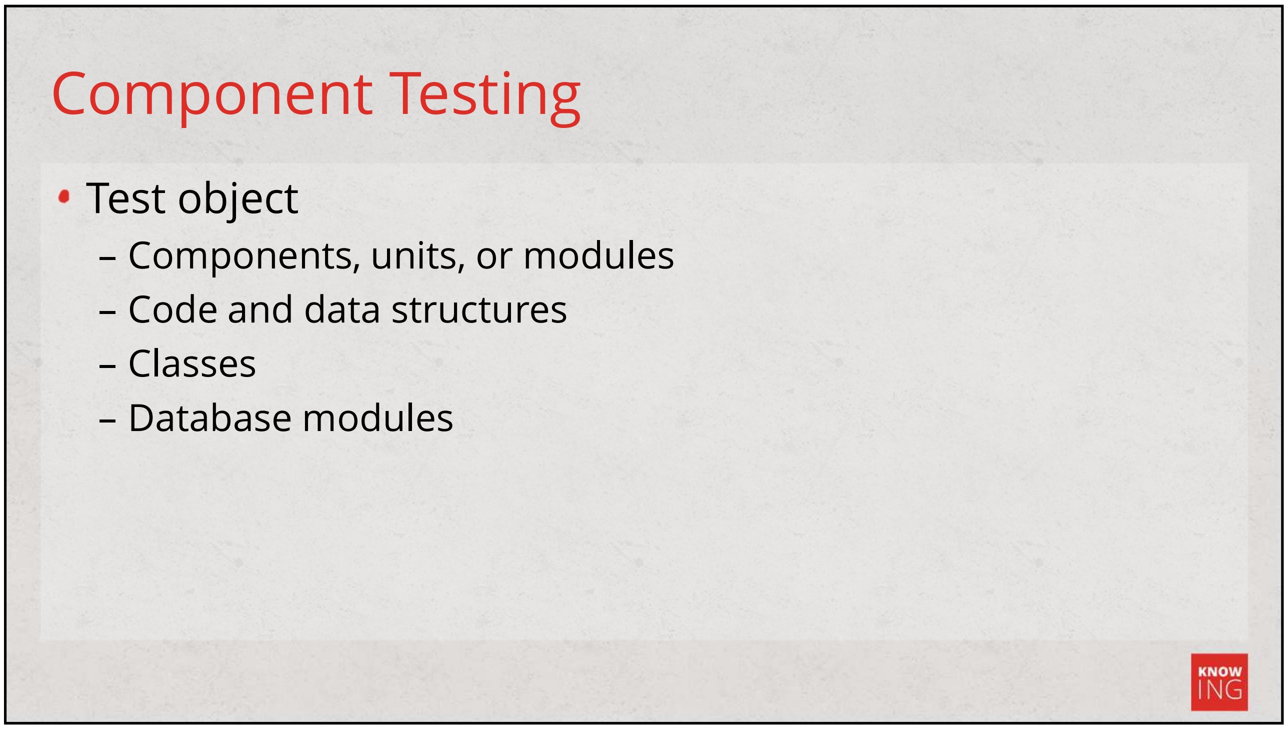
- Objectives
  - Reducing risks
  - Verifying functional and non-functional behaviours of component
  - Building confidence in the component's quality
  - Finding defects in component
  - Preventing defects from escaping to higher test levels
- Done in isolation: mock objects, service virtualisation, harness, stubs, and drivers
- Functionality, non-functional characteristics, structural properties

KNOW ING

Component Testing

- Test basis
  - Detailed design
  - Code
  - Data model
  - Component specifications

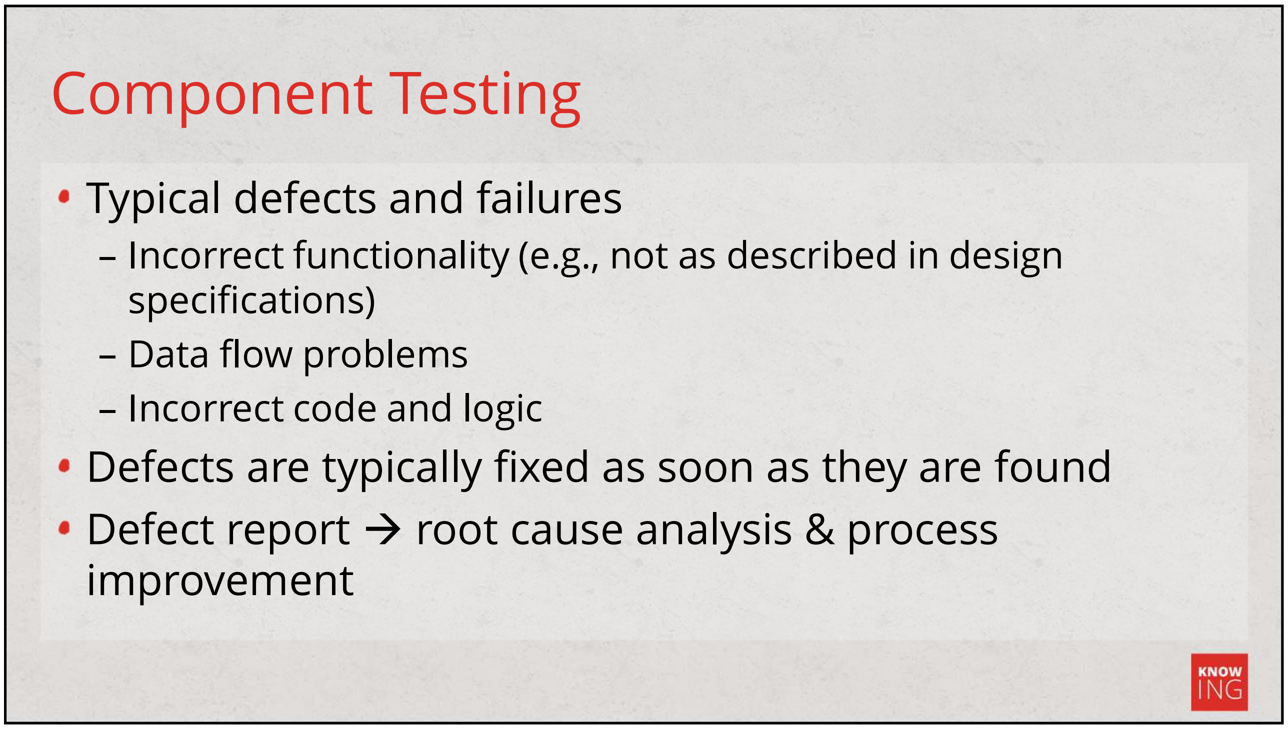
KNOW ING



Component Testing

- Test object
  - Components, units, or modules
  - Code and data structures
  - Classes
  - Database modules

55



Component Testing

- Typical defects and failures
  - Incorrect functionality (e.g., not as described in design specifications)
  - Data flow problems
  - Incorrect code and logic
- Defects are typically fixed as soon as they are found
- Defect report → root cause analysis & process improvement

56

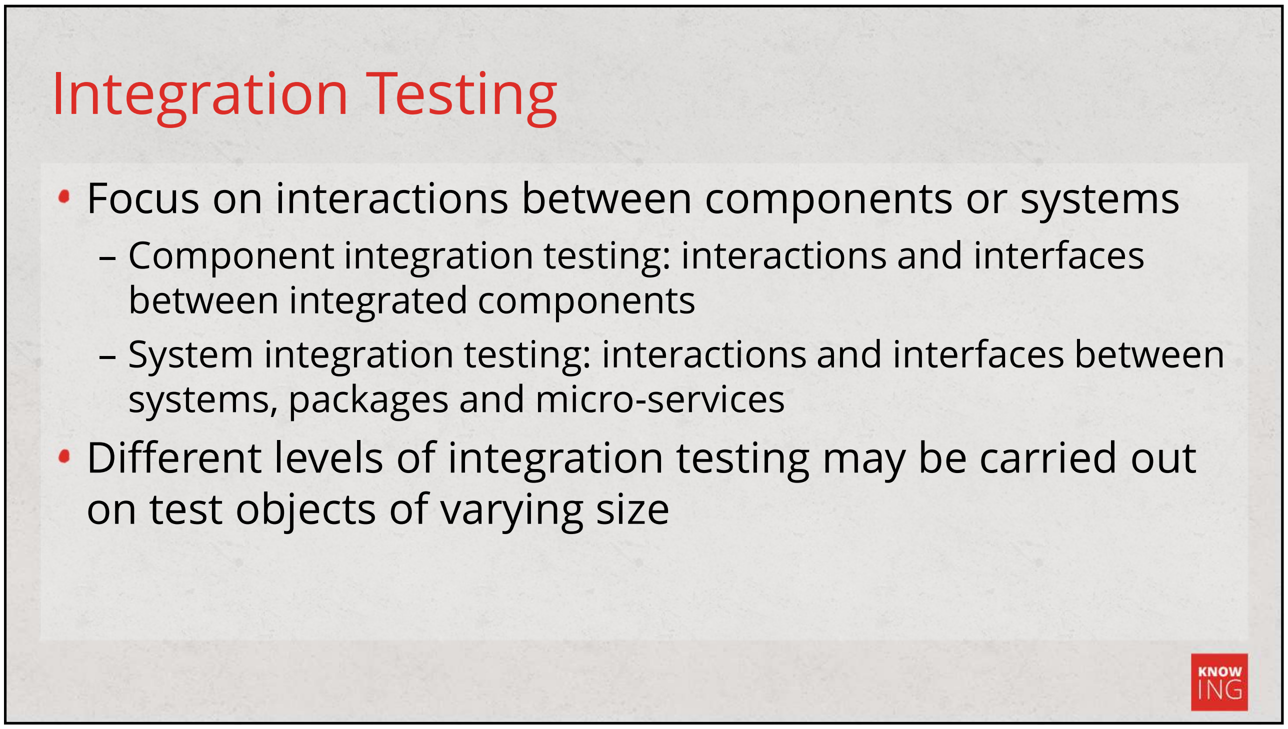
Component Testing

- Approaches & responsibility
  - Usually performed by developer who wrote the code, but it at least requires access to the code being tested
- Developers may alternate component development with finding and fixing defects
- Developers will often write and execute tests after having written the code for a component
- In agile development, writing automated component tests may precede writing application code

Component Testing – Techniques

- Black-box
  - Equivalence class partitioning (ECP)
  - Boundary value analysis (BVA)
- White-box
  - Code coverage
    - Statement, branch, path
- State-based testing
  - State machine diagram

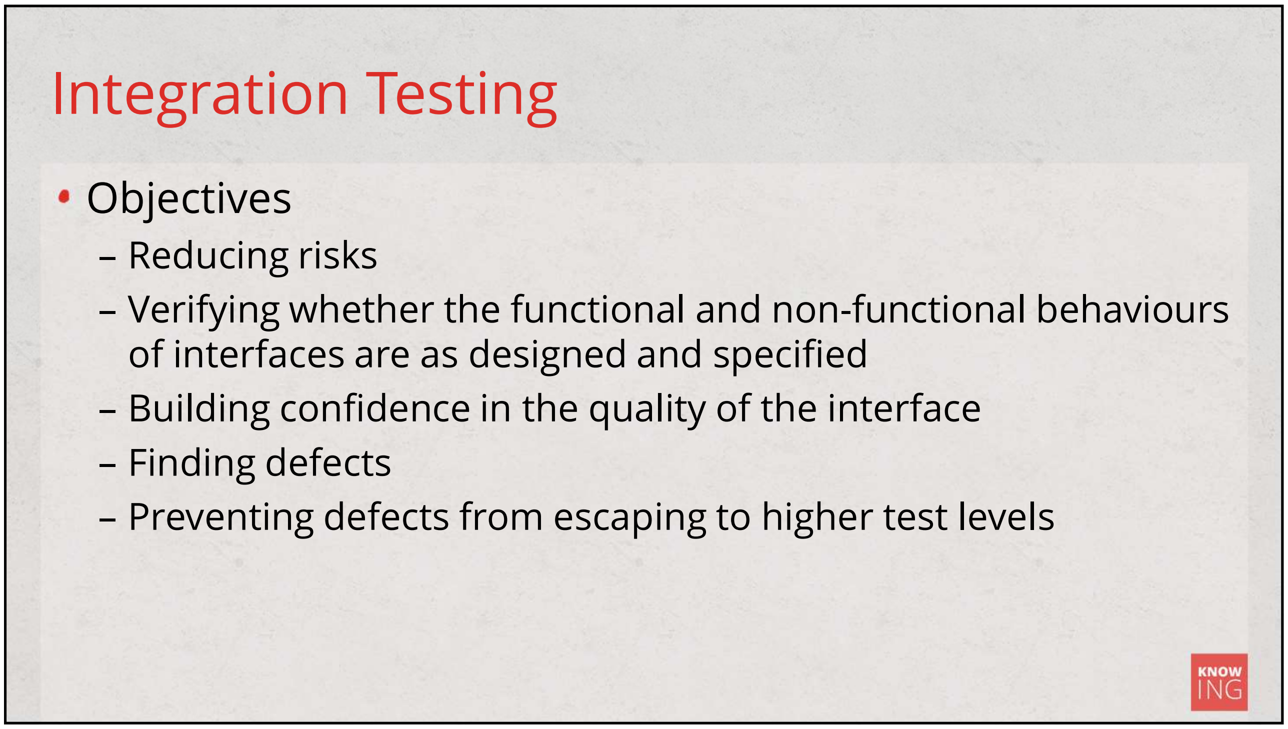
– To be elaborated in lectures in Weeks 7&8



Integration Testing

- Focus on interactions between components or systems
  - Component integration testing: interactions and interfaces between integrated components
  - System integration testing: interactions and interfaces between systems, packages and micro-services
- Different levels of integration testing may be carried out on test objects of varying size

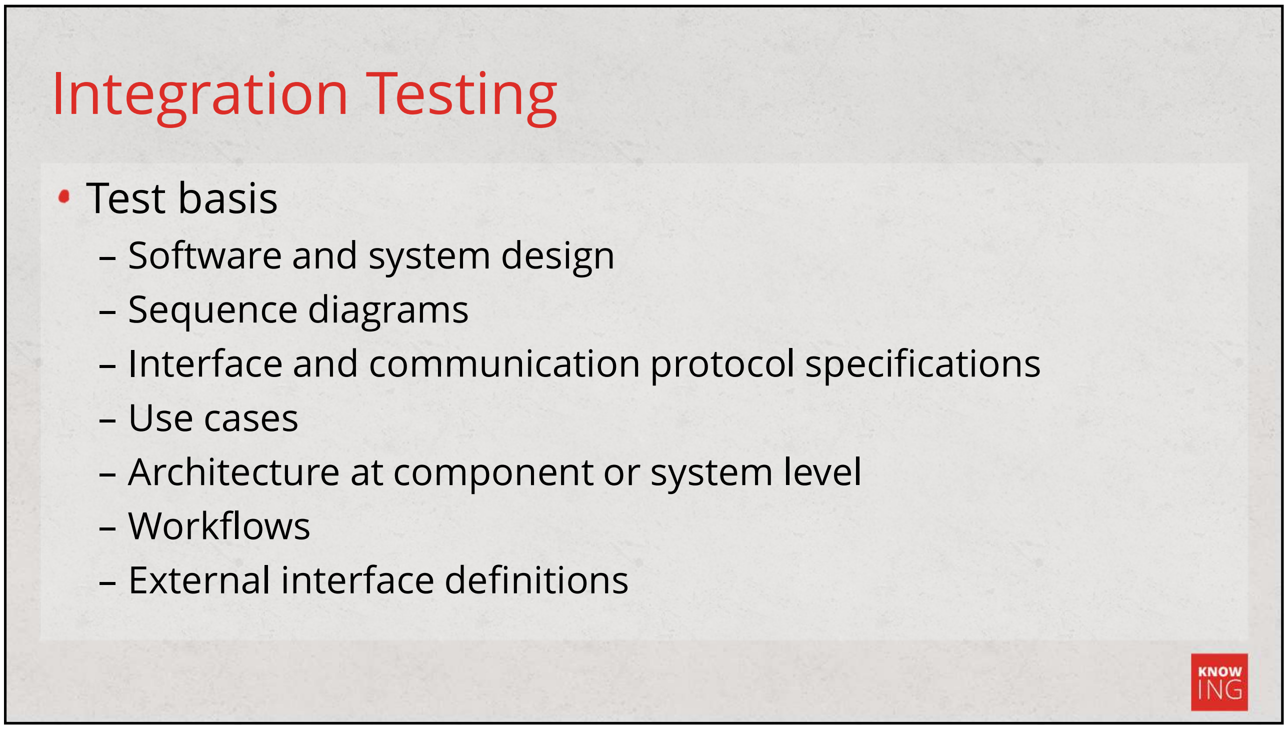
59



Integration Testing

- Objectives
  - Reducing risks
  - Verifying whether the functional and non-functional behaviours of interfaces are as designed and specified
  - Building confidence in the quality of the interface
  - Finding defects
  - Preventing defects from escaping to higher test levels

60

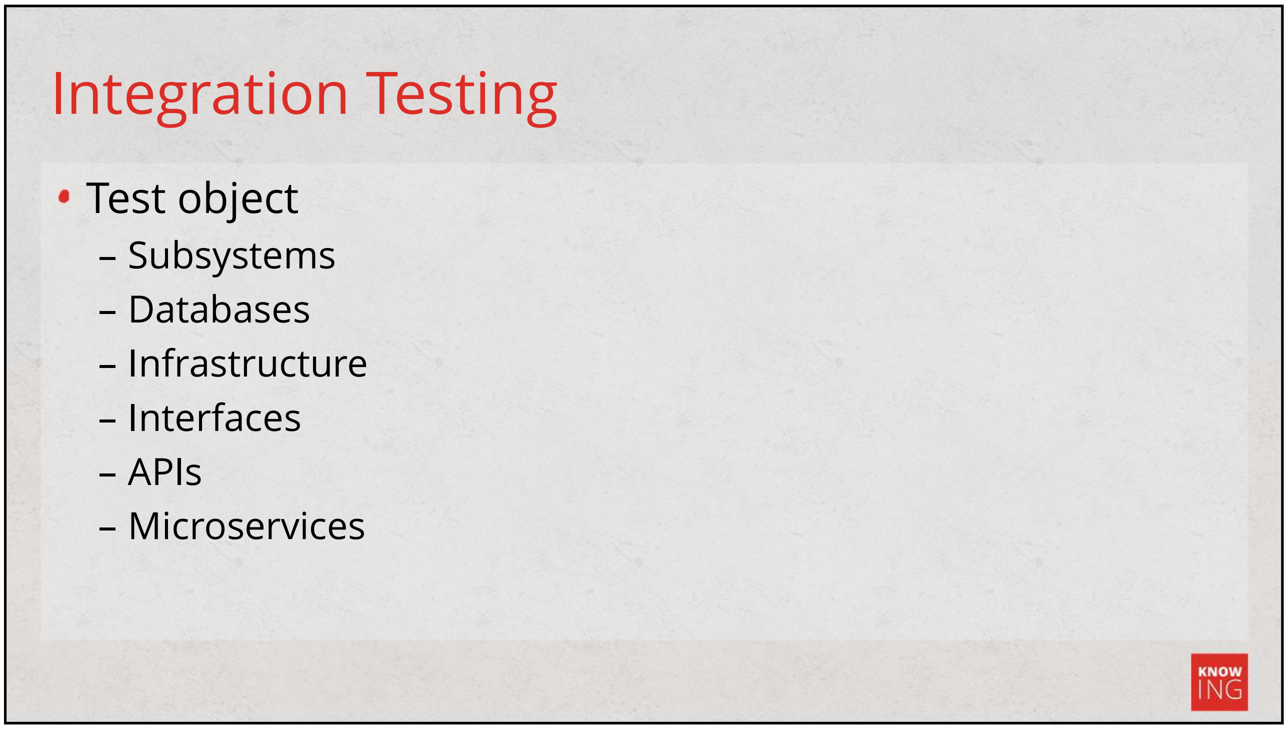


Integration Testing

- Test basis
  - Software and system design
  - Sequence diagrams
  - Interface and communication protocol specifications
  - Use cases
  - Architecture at component or system level
  - Workflows
  - External interface definitions

KNOW ING

61



Integration Testing

- Test object
  - Subsystems
  - Databases
  - Infrastructure
  - Interfaces
  - APIs
  - Microservices

KNOW ING

62

Integration Testing

- Typical defects and failures
  - Inconsistent message structures between systems
  - Incorrect data, missing data, or incorrect data encoding
  - Interface mismatch
  - Failures in communication between systems
  - Unhandled or improperly handled communication failures between systems
  - Incorrect assumptions about the meaning, units, or boundaries of the data being passed between systems
  - Failure to comply with mandatory security regulations

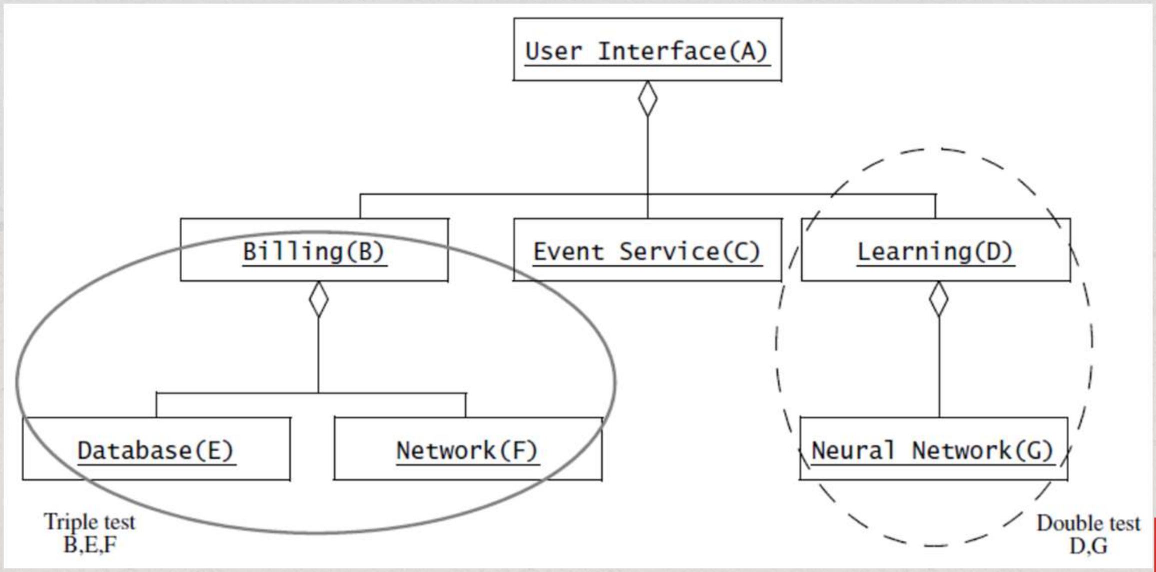
63

Integration Testing

- Approaches & responsibility
  - Both component integration tests and system integration tests should concentrate on the integration itself
  - Component integration testing is often the responsibility of developers and system integration testing is generally the responsibility of testers
  - To detect defects early, integration should normally be incremental rather than ‘big bang’
  - The greater the scope of integration, the more difficult it becomes to isolate defects to a specific component or system, which may lead to increased risk and additional time for troubleshooting

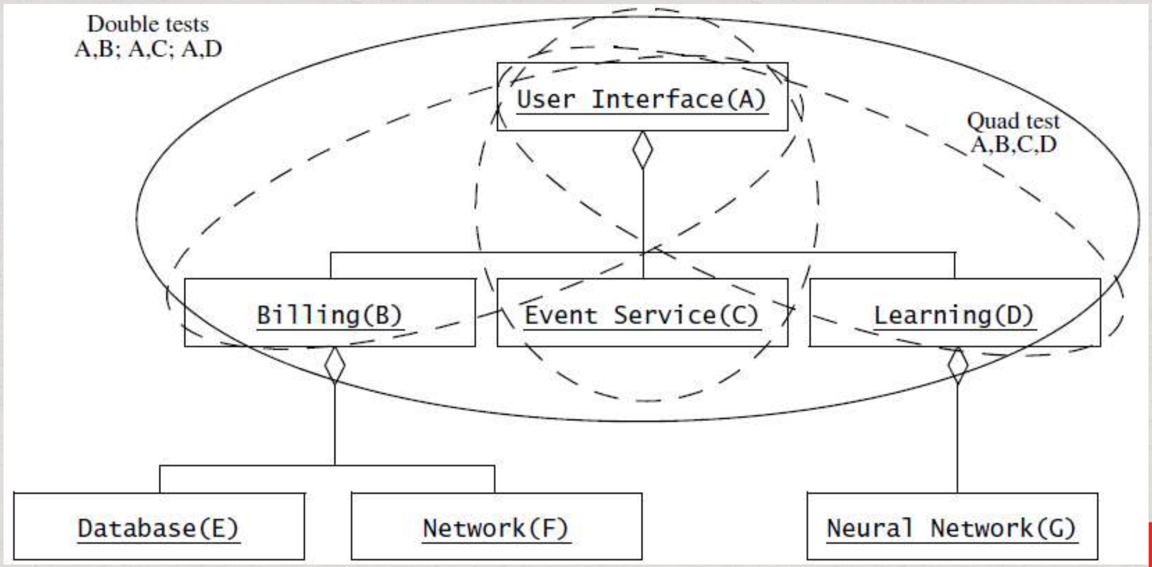
64

Integration Testing



65

Integration Testing



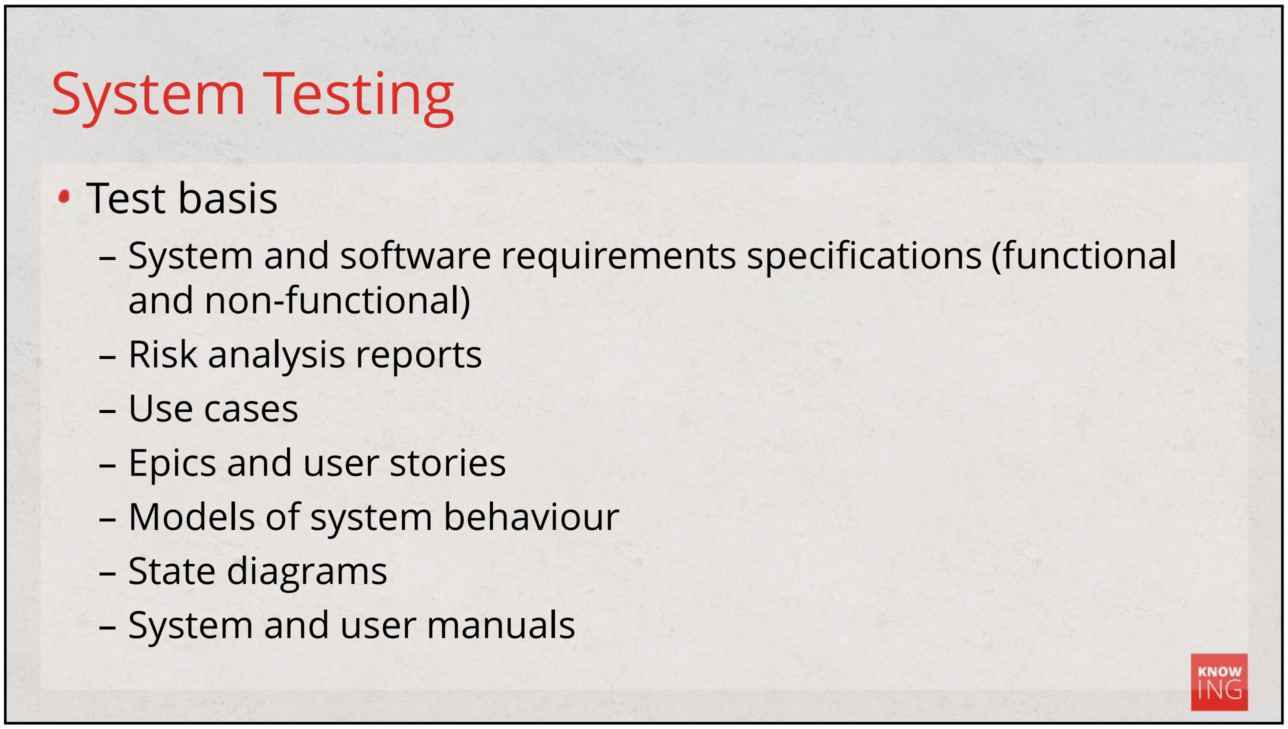
66

System Testing

- Focus on the behaviour and capabilities of a whole system or product
- Consider the end-to-end tasks the system can perform and the non-functional behaviours it exhibits while performing those tasks

System Testing

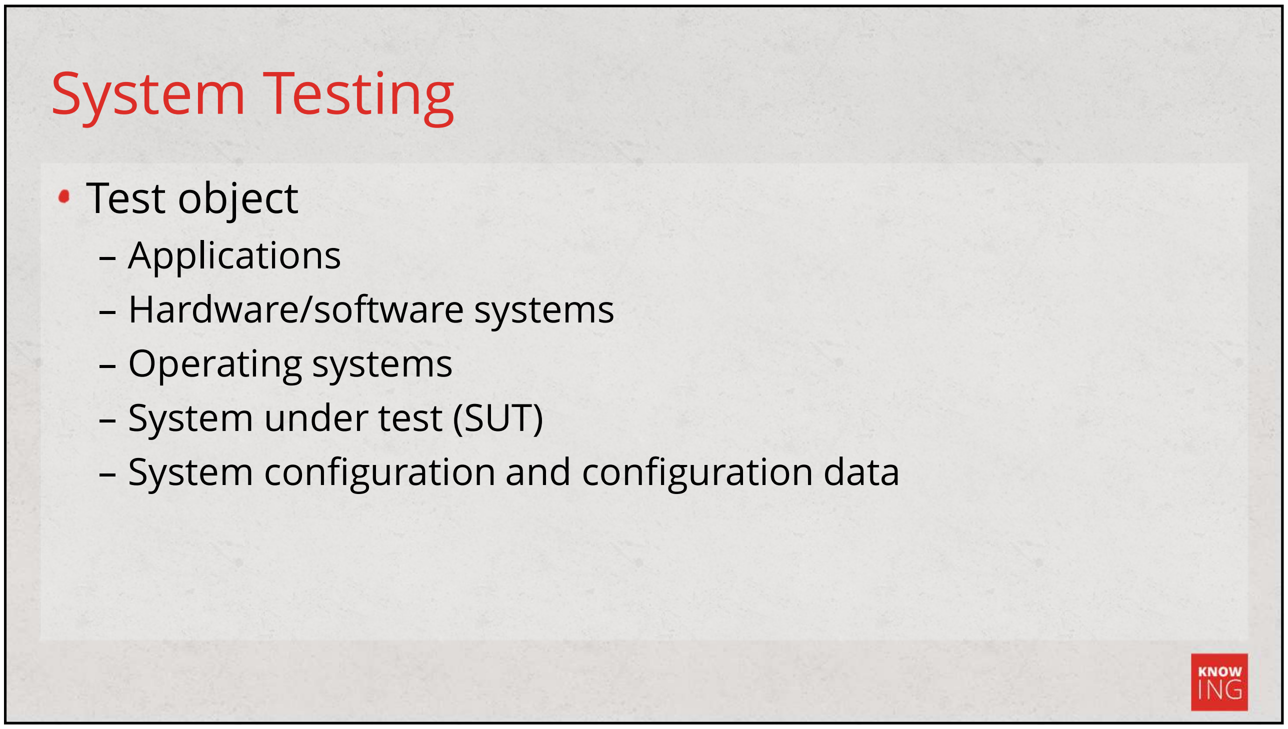
- Objectives
  - Reducing risks
  - Verifying whether the functional and non-functional behaviours of system are as designed and specified
  - Building confidence in the quality of the system as a whole
  - Finding defects
  - Preventing defects from escaping to higher test levels or production
- Test environment should ideally correspond to the final target or production environment



System Testing

- Test basis
  - System and software requirements specifications (functional and non-functional)
  - Risk analysis reports
  - Use cases
  - Epics and user stories
  - Models of system behaviour
  - State diagrams
  - System and user manuals

69



System Testing

- Test object
  - Applications
  - Hardware/software systems
  - Operating systems
  - System under test (SUT)
  - System configuration and configuration data

70

System Testing

- Typical defects and failures
  - Incorrect calculations
  - Incorrect or unexpected system functional or non-functional behaviour
  - Incorrect control and/or data flows within the system
  - Failure to properly and completely carry out end-to-end functional tasks
  - Failure of the system to work properly in the production environment
  - Failure of the system to work as described in system and user manuals

KNOW ING

71

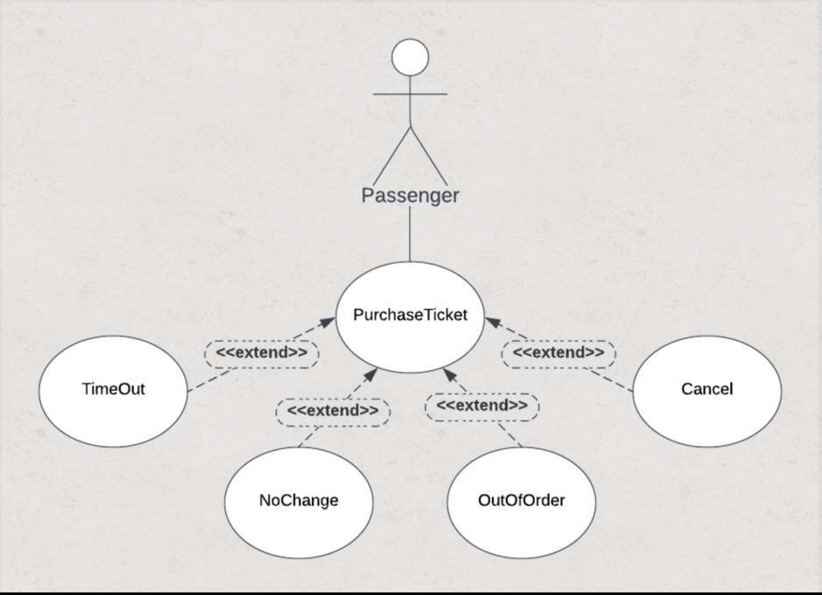
System Testing

- Approaches & responsibility
  - Focus on the overall, end-to-end behaviour of the system as a whole, both functional and non-functional
  - Independent testers typically carry out system testing
  - Defects in specifications can lead to a lack of understanding of, or disagreements about, expected system behaviour
  - Early involvement of testers in user story refinement or static testing activities, such as reviews, helps to reduce the incidence of such situations

KNOW ING

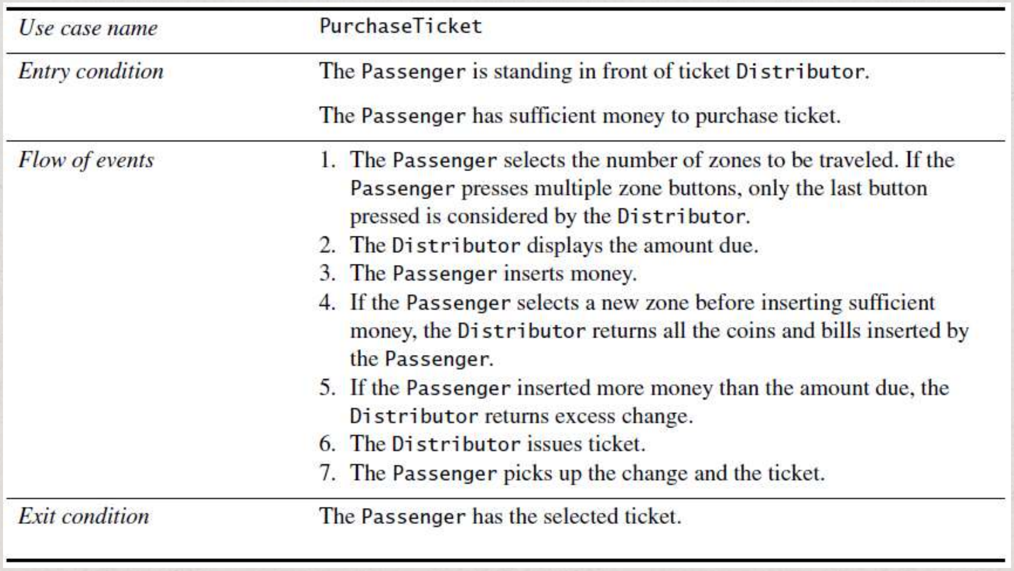
72

System Testing – Functional



73

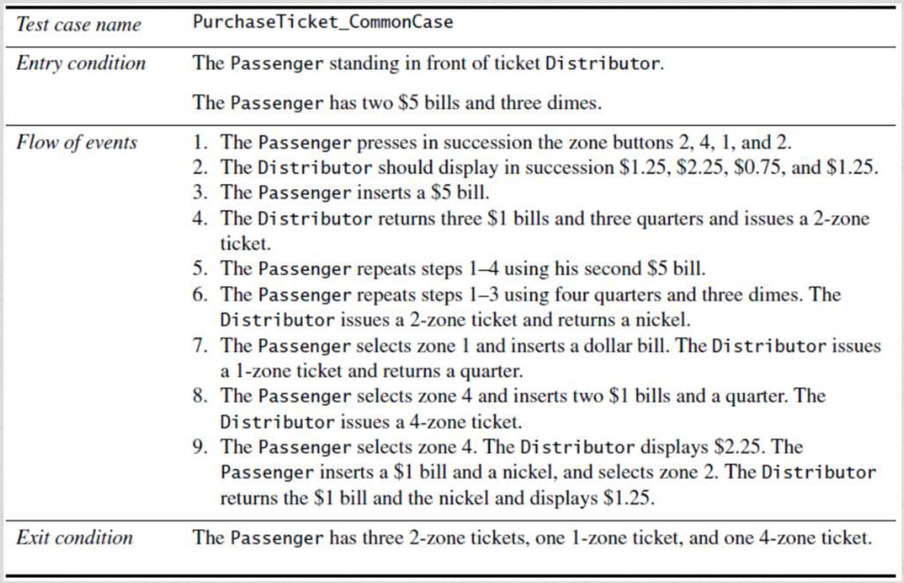
System Testing – Functional



Use case name	`PurchaseTicket`
Entry condition	The `Passenger` is standing in front of ticket `Distributor`. The `Passenger` has sufficient money to purchase ticket.
Flow of events	1. The `Passenger` selects the number of zones to be traveled. If the `Passenger` presses multiple zone buttons, only the last button pressed is considered by the `Distributor`. 2. The `Distributor` displays the amount due. 3. The `Passenger` inserts money. 4. If the `Passenger` selects a new zone before inserting sufficient money, the `Distributor` returns all the coins and bills inserted by the `Passenger`. 5. If the `Passenger` inserted more money than the amount due, the `Distributor` returns excess change. 6. The `Distributor` issues ticket. 7. The `Passenger` picks up the change and the ticket.
Exit condition	The `Passenger` has the selected ticket.

74

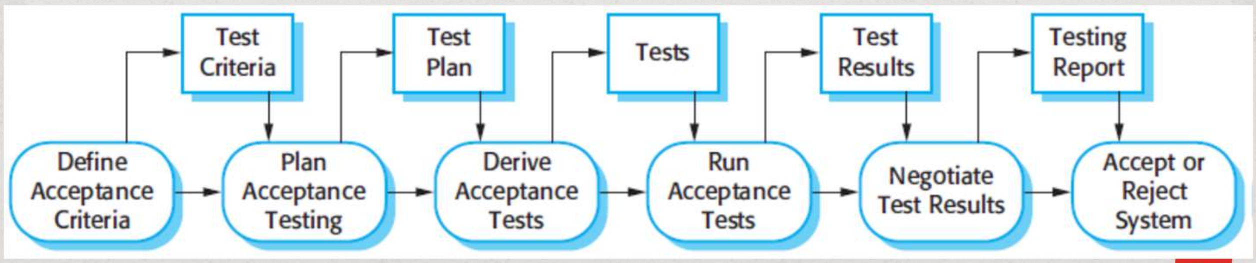
System Testing – Functional

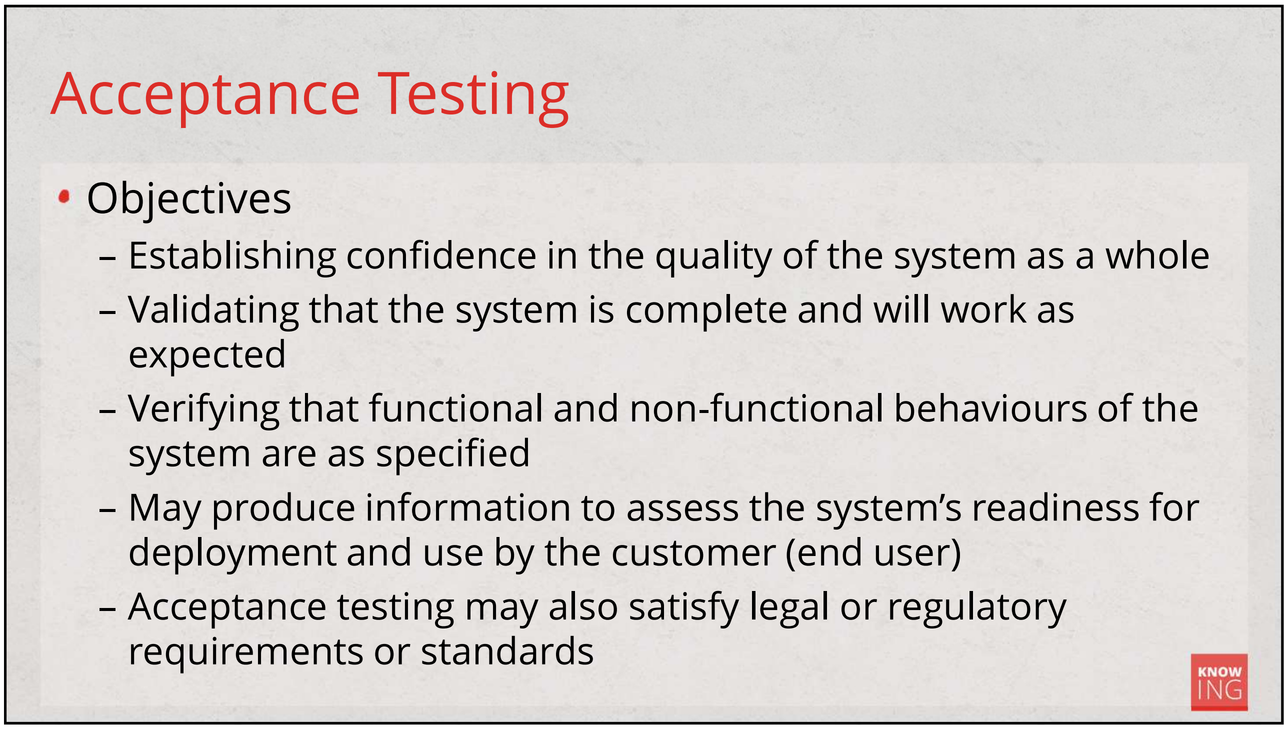


Test case name	PurchaseTicket_CommonCase
Entry condition	The Passenger standing in front of ticket Distributor. The Passenger has two $5 bills and three dimes.
Flow of events	1. The Passenger presses in succession the zone buttons 2, 4, 1, and 2. 2. The Distributor should display in succession $1.25, $2.25, $0.75, and $1.25. 3. The Passenger inserts a $5 bill. 4. The Distributor returns three $1 bills and three quarters and issues a 2-zone ticket. 5. The Passenger repeats steps 1–4 using his second $5 bill. 6. The Passenger repeats steps 1–3 using four quarters and three dimes. The Distributor issues a 2-zone ticket and returns a nickel. 7. The Passenger selects zone 1 and inserts a dollar bill. The Distributor issues a 1-zone ticket and returns a quarter. 8. The Passenger selects zone 4 and inserts two $1 bills and a quarter. The Distributor issues a 4-zone ticket. 9. The Passenger selects zone 4. The Distributor displays $2.25. The Passenger inserts a $1 bill and a nickel, and selects zone 2. The Distributor returns the $1 bill and the nickel and displays $1.25.
Exit condition	The Passenger has three 2-zone tickets, one 1-zone ticket, and one 4-zone ticket.

Acceptance Testing

- Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

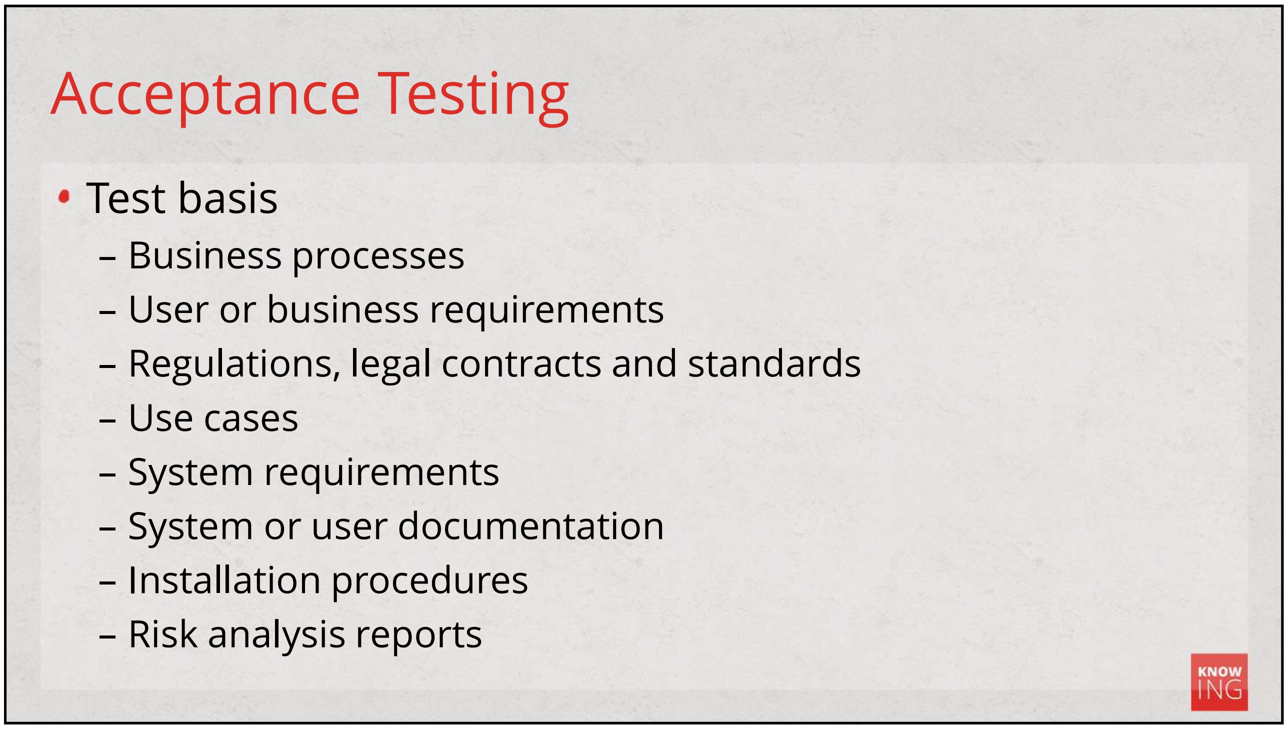




Acceptance Testing

- 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 behaviours of the system are as specified
  - May produce information to assess the system's readiness for deployment and use by the customer (end user)
  - Acceptance testing may also satisfy legal or regulatory requirements or standards

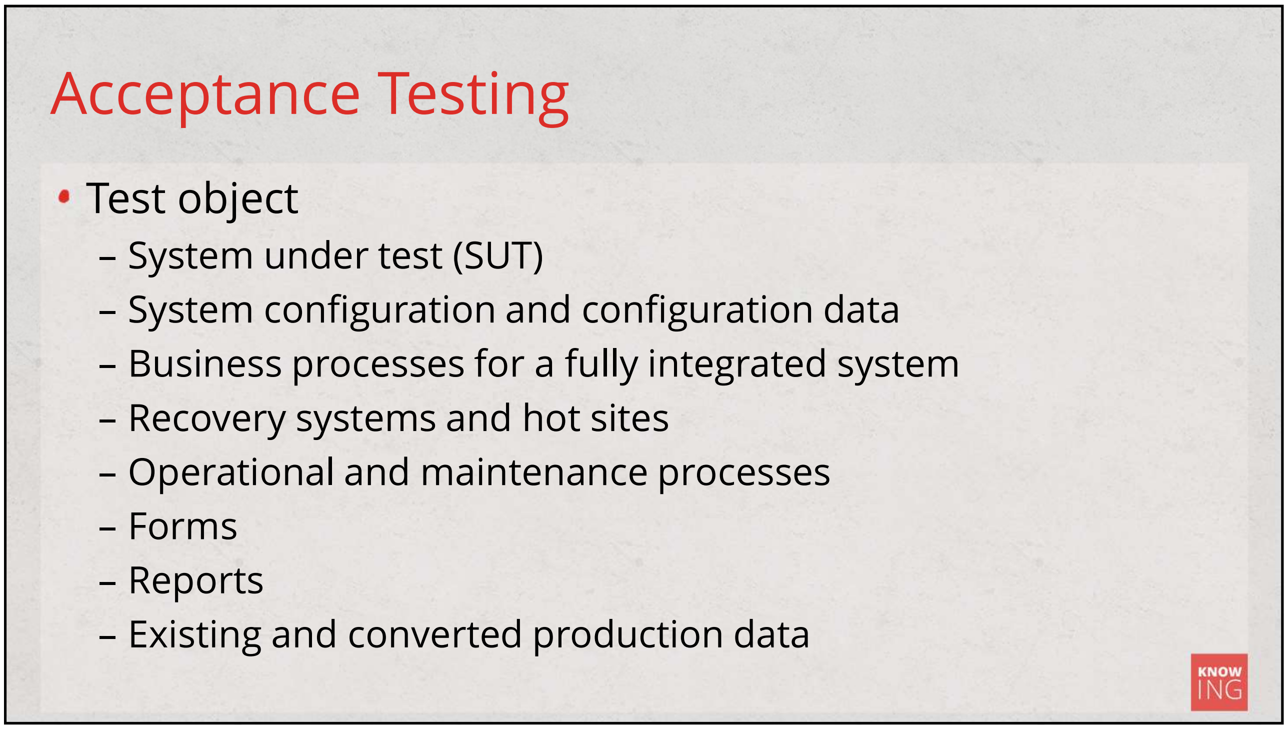
77



Acceptance Testing

- Test basis
  - Business processes
  - User or business requirements
  - Regulations, legal contracts and standards
  - Use cases
  - System requirements
  - System or user documentation
  - Installation procedures
  - Risk analysis reports

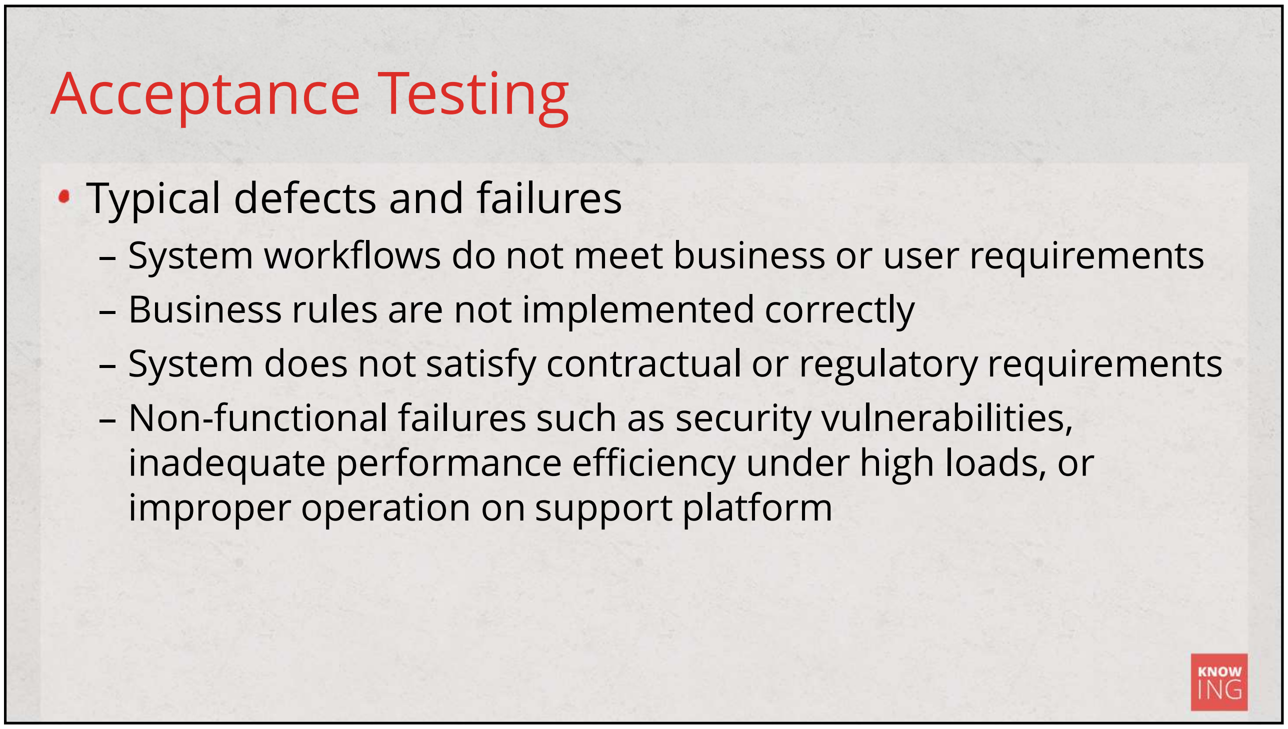
78



Acceptance Testing

- Test object
  - System under test (SUT)
  - 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

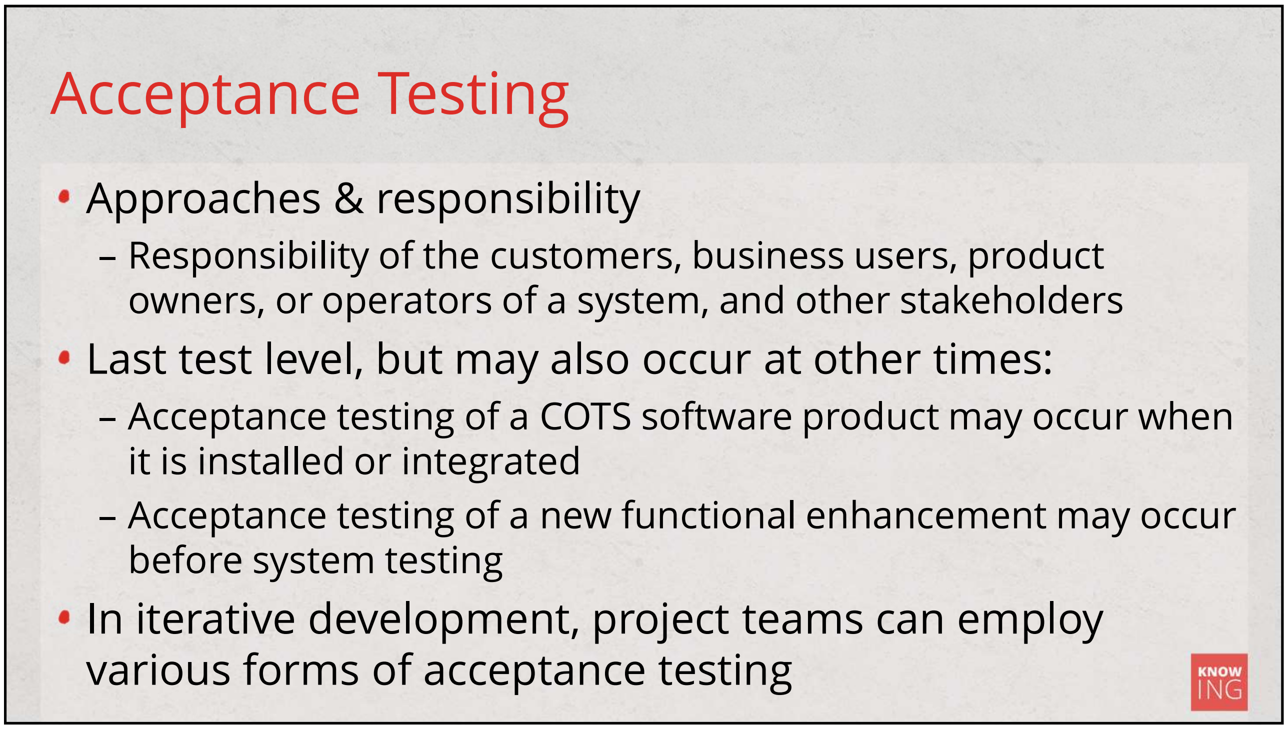
79



Acceptance Testing

- Typical defects and 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 efficiency under high loads, or improper operation on support platform

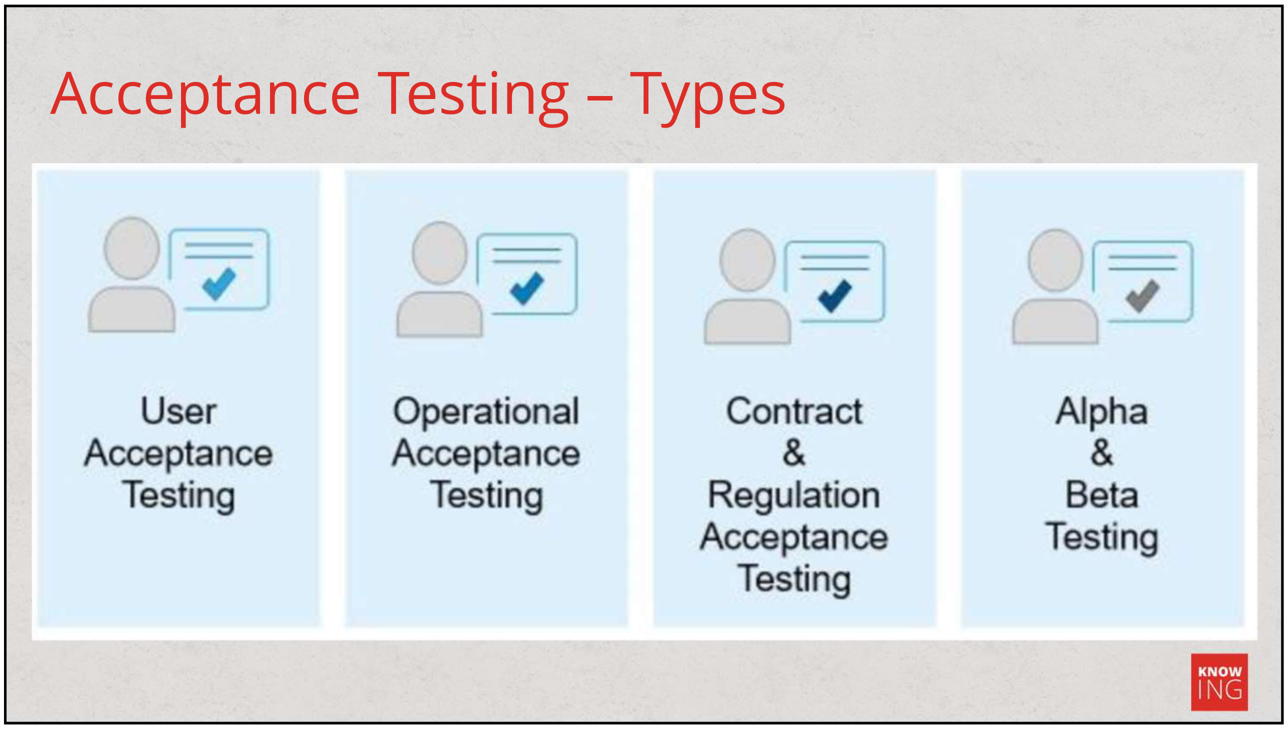
80



Acceptance Testing

- Approaches & responsibility
  - Responsibility of the customers, business users, product owners, or operators of a system, and other stakeholders
- Last test level, but may also occur at other times:
  - Acceptance testing of a COTS software product may occur when it is installed or integrated
  - Acceptance testing of a new functional enhancement may occur before system testing
- In iterative development, project teams can employ various forms of acceptance testing

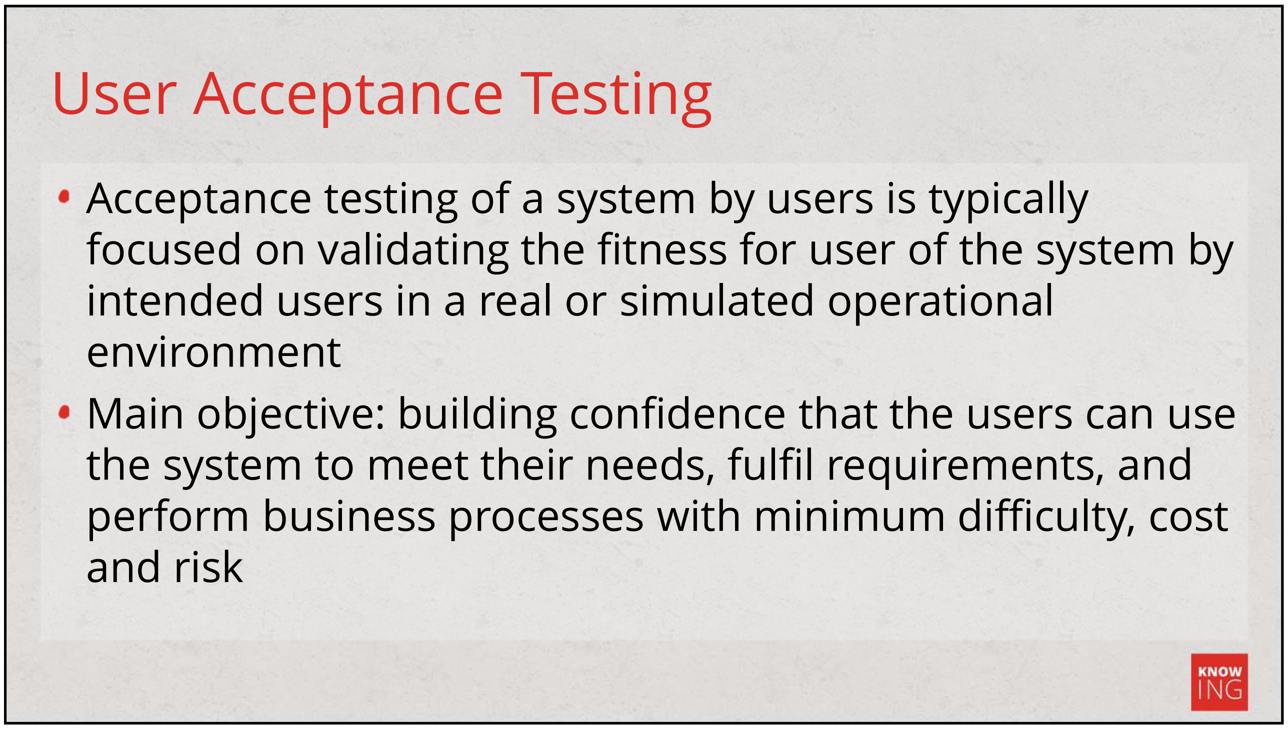
81



Acceptance Testing – Types

User Acceptance Testing	Operational Acceptance Testing	Contract & Regulation Acceptance Testing	Alpha & Beta Testing

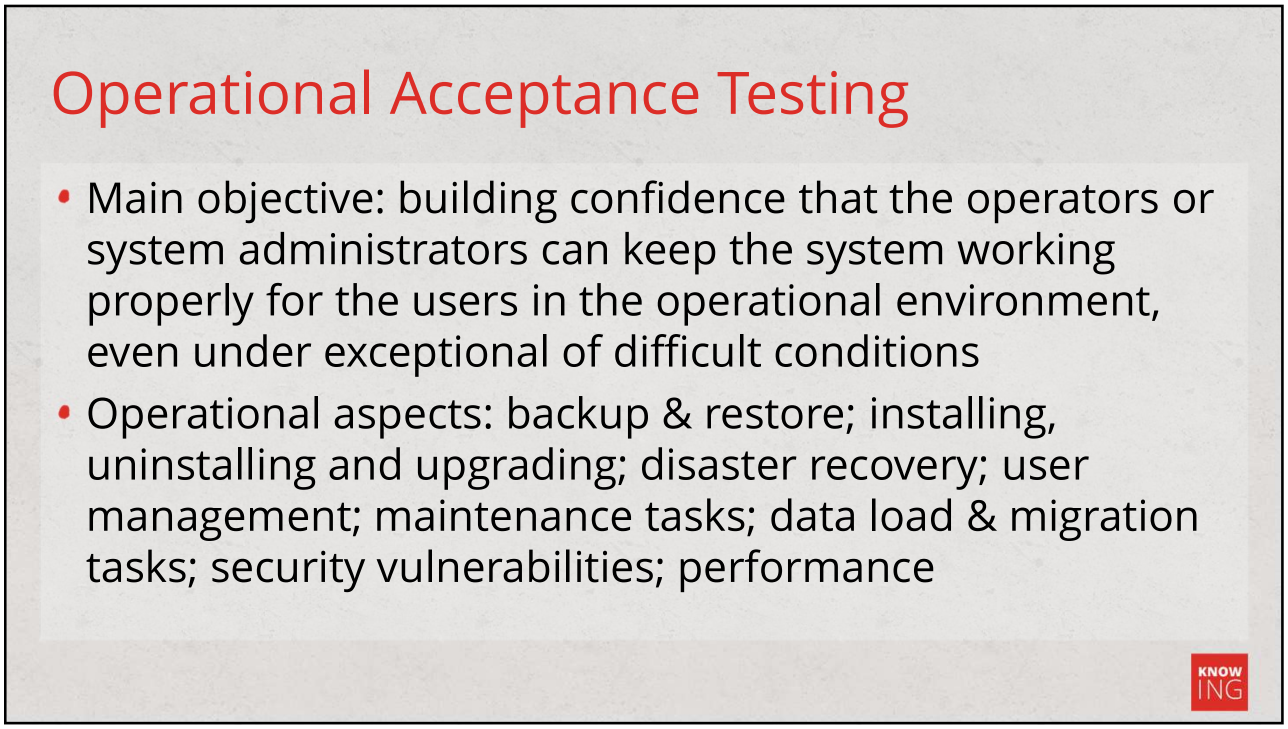
82



User Acceptance Testing

- Acceptance testing of a system by users is typically focused on validating the fitness for user of the system by intended users in a real or simulated operational environment
- Main objective: building confidence that the users can use the system to meet their needs, fulfil requirements, and perform business processes with minimum difficulty, cost and risk

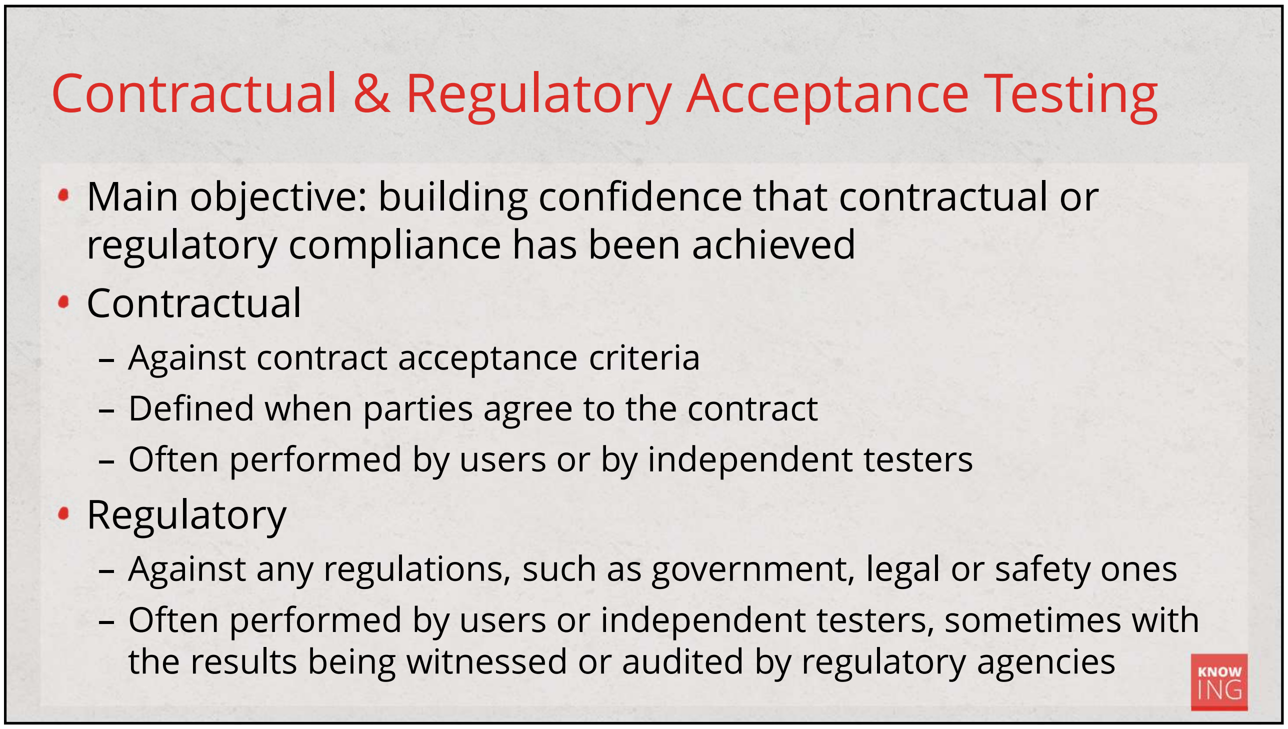
83



Operational Acceptance Testing

- Main objective: building confidence that the operators or system administrators can keep the system working properly for the users in the operational environment, even under exceptional of difficult conditions
- Operational aspects: backup & restore; installing, uninstalling and upgrading; disaster recovery; user management; maintenance tasks; data load & migration tasks; security vulnerabilities; performance

84

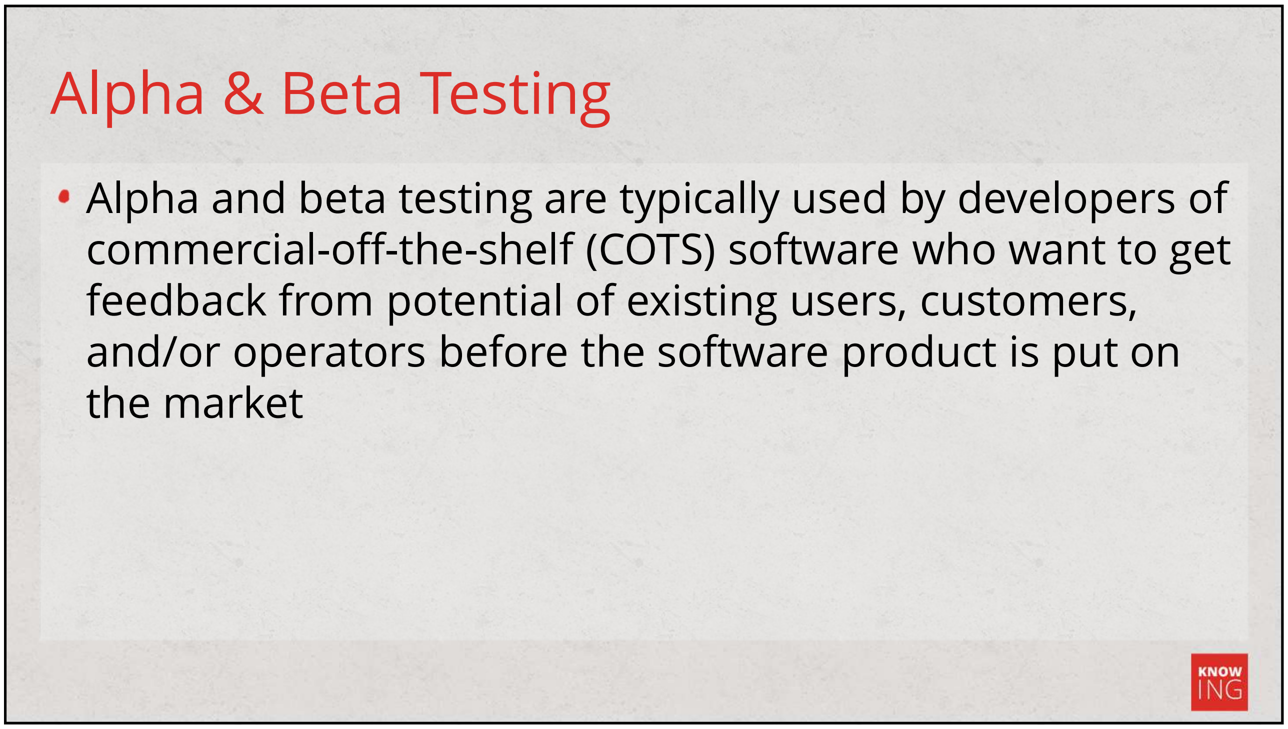


Contractual & Regulatory Acceptance Testing

- Main objective: building confidence that contractual or regulatory compliance has been achieved
- Contractual
  - Against contract acceptance criteria
  - Defined when parties agree to the contract
  - Often performed by users or by independent testers
- Regulatory
  - Against any regulations, such as government, legal or safety ones
  - Often performed by users or independent testers, sometimes with the results being witnessed or audited by regulatory agencies

KNOW ING

85

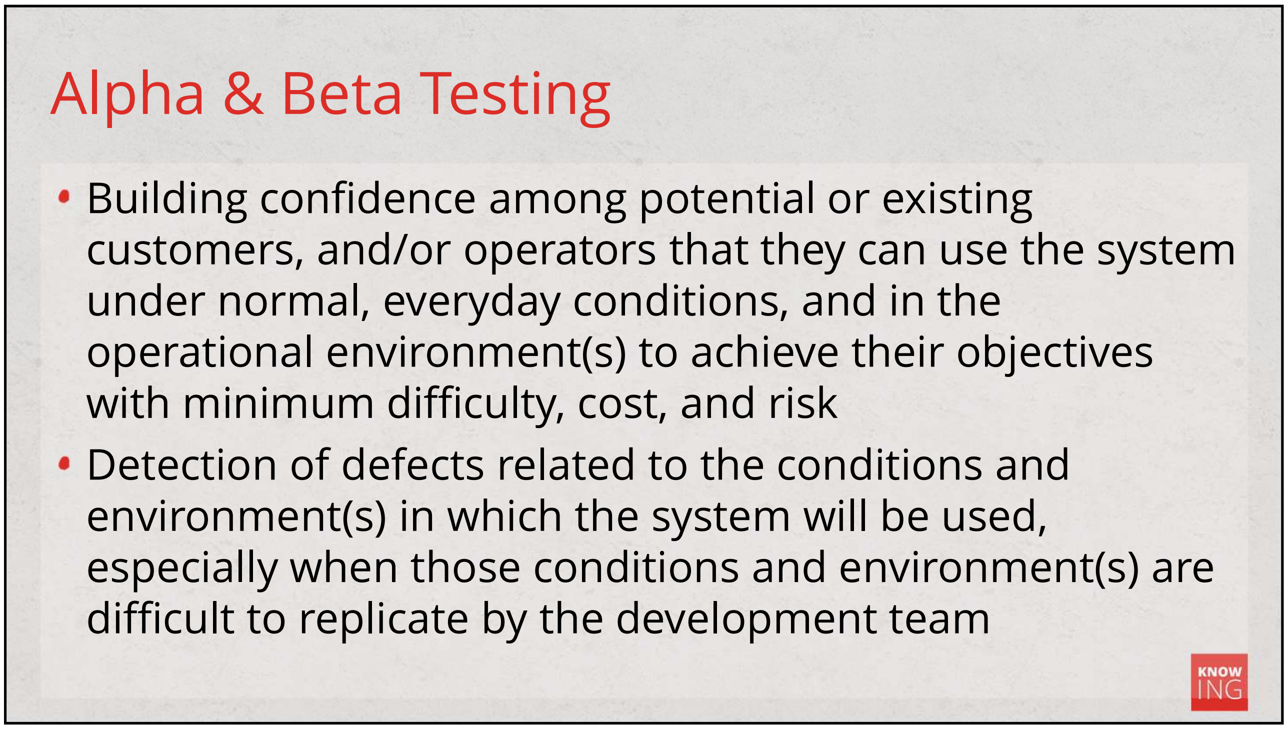


Alpha & Beta Testing

- Alpha and beta testing are typically used by developers of commercial-off-the-shelf (COTS) software who want to get feedback from potential of existing users, customers, and/or operators before the software product is put on the market

KNOW ING

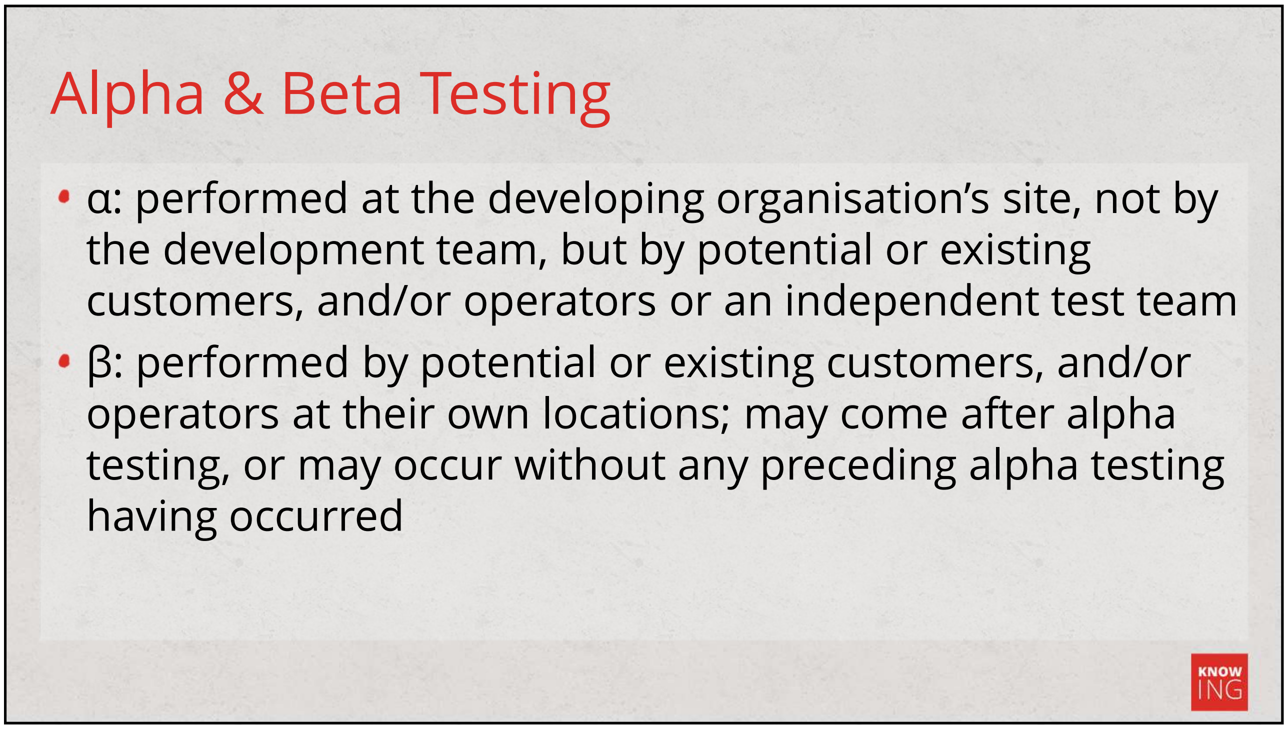
86



Alpha & Beta Testing

- Building confidence among potential or existing customers, and/or operators that they can use the system under normal, everyday conditions, and in the operational environment(s) to achieve their objectives with minimum difficulty, cost, and risk
- Detection of defects related to the conditions and environment(s) in which the system will be used, especially when those conditions and environment(s) are difficult to replicate by the development team

87



Alpha & Beta Testing

- α: performed at the developing organisation’s site, not by the development team, but by potential or existing customers, and/or operators or an independent test team
- β: performed by potential or existing customers, and/or operators at their own locations; may come after alpha testing, or may occur without any preceding alpha testing having occurred

88

Test Types

89

Testing Types

- Evaluating **functional** quality characteristics
- Evaluating the effects of **change**
- Evaluating where the **structure** or architecture of the component or system is correct
- Evaluating **non-functional** quality characteristics

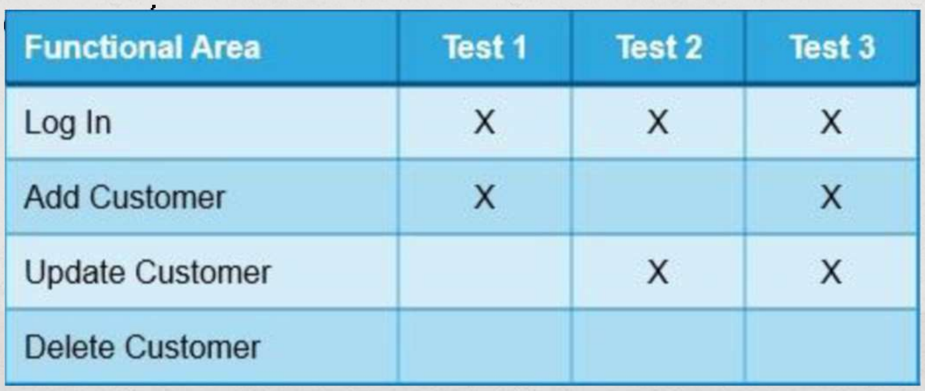
90

Functional Testing

- Involve tests that evaluate functions that the system should perform
- Functional requirements may be described in work products, or they may be undocumented
- Functional tests should be performed at all test levels though the focus is different at each level
- Functional testing considers the behaviour of the software, so black-box techniques may be used

Functional Testing - Coverage

- Functional coverage: extent to which some type of functional element has been exercised by tests, and is expressed as a percentage of the type(s) of element being



Functional Area	Test 1	Test 2	Test 3
Log In	X	X	X
Add Customer	X		X
Update Customer		X	X
Delete Customer			

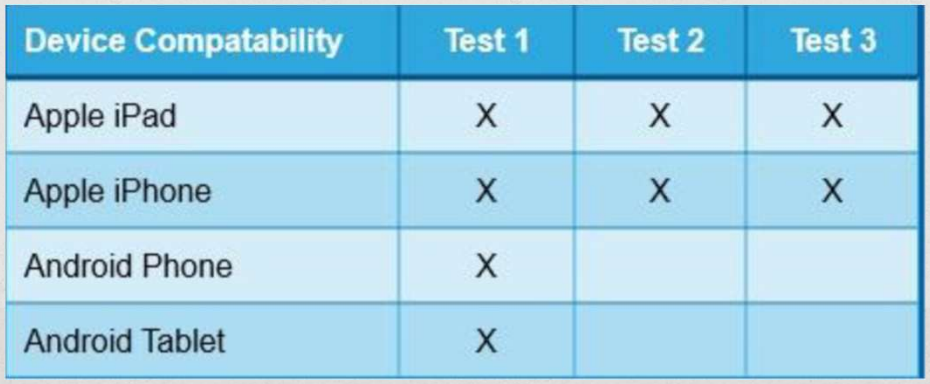
Non-Functional Testing

- To measure characteristics of systems and software that can be quantified on a varying scale, such as response time for performance testing
- Testing of “how well” the system behaves
- May be performed at all test levels
- Consider external behaviour of software and mostly use black-box test design techniques

93

Non-Functional Testing – Coverage

- Extent to which some type of non-functional element has been exercised by tests, and is expressed as a percentage of the type(s) of element being covered

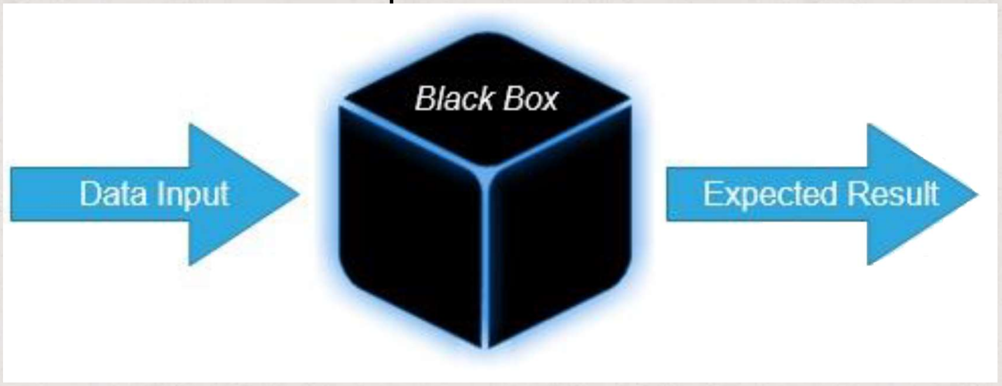


Device Compatability	Test 1	Test 2	Test 3
Apple iPad	X	X	X
Apple iPhone	X	X	X
Android Phone	X		
Android Tablet	X		

94

Black-Box Testing

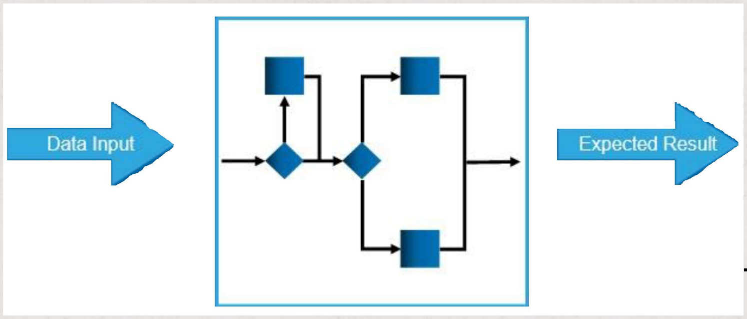
- If actual result = expect result then √



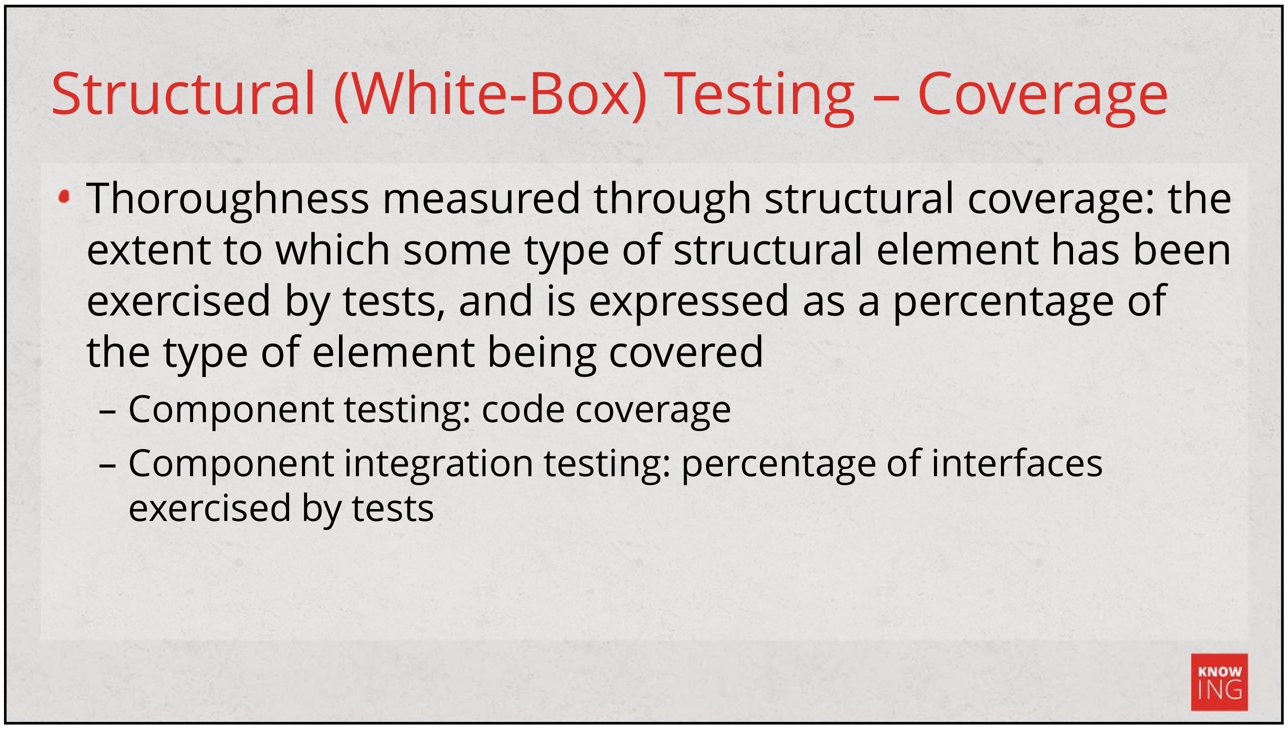
– To be studies in Week 8

Structural Testing a.k.a. White-Box Testing

- Derive tests based on the system's internal structure or implementation: code, architecture, work flows, and/or data flows within the system



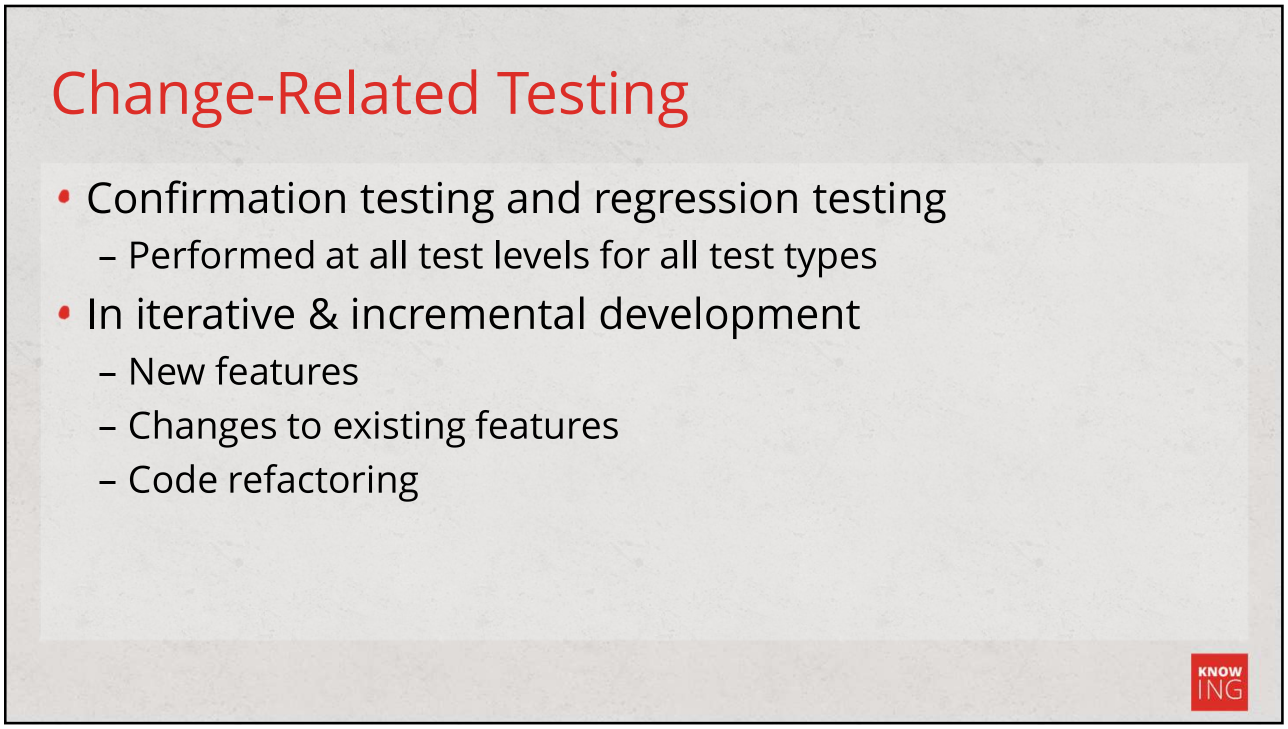
– To be studies in Week 9



Structural (White-Box) Testing – Coverage

- Thoroughness measured through structural coverage: the extent to which some type of structural element has been exercised by tests, and is expressed as a percentage of the type of element being covered
  - Component testing: code coverage
  - Component integration testing: percentage of interfaces exercised by tests

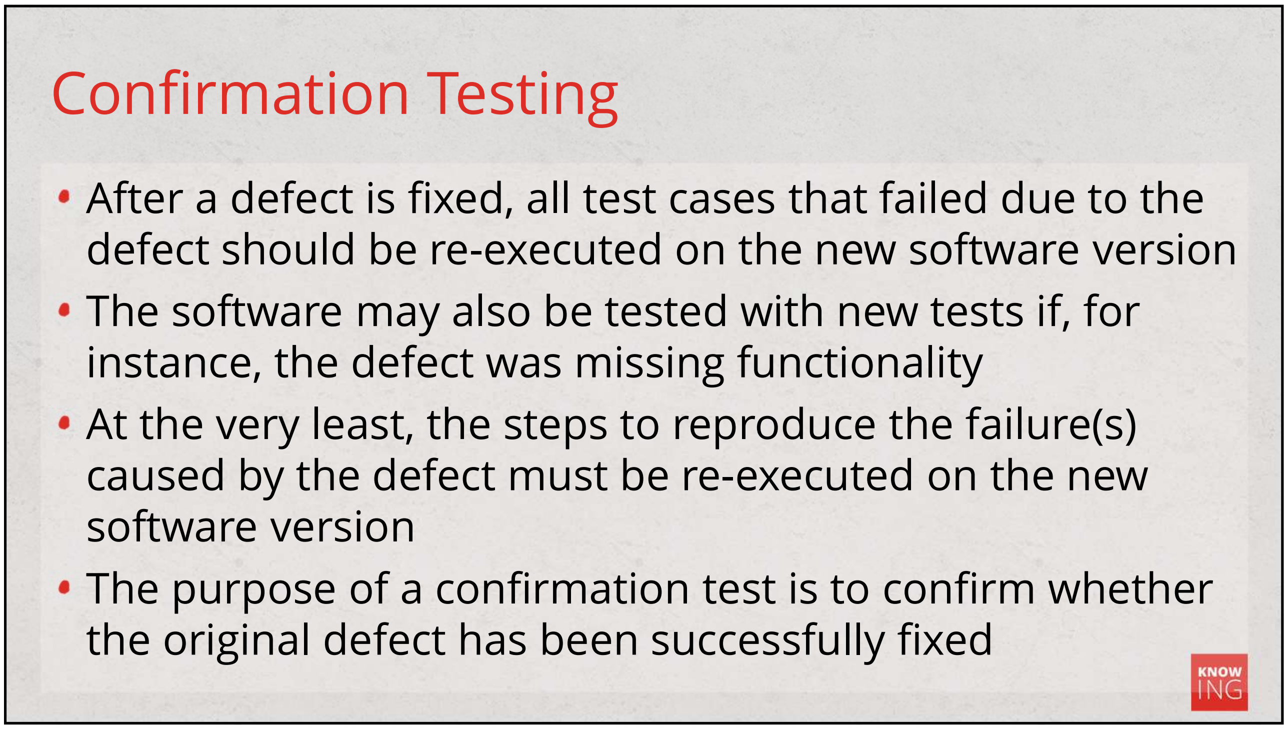
97



Change-Related Testing

- Confirmation testing and regression testing
  - Performed at all test levels for all test types
- In iterative & incremental development
  - New features
  - Changes to existing features
  - Code refactoring

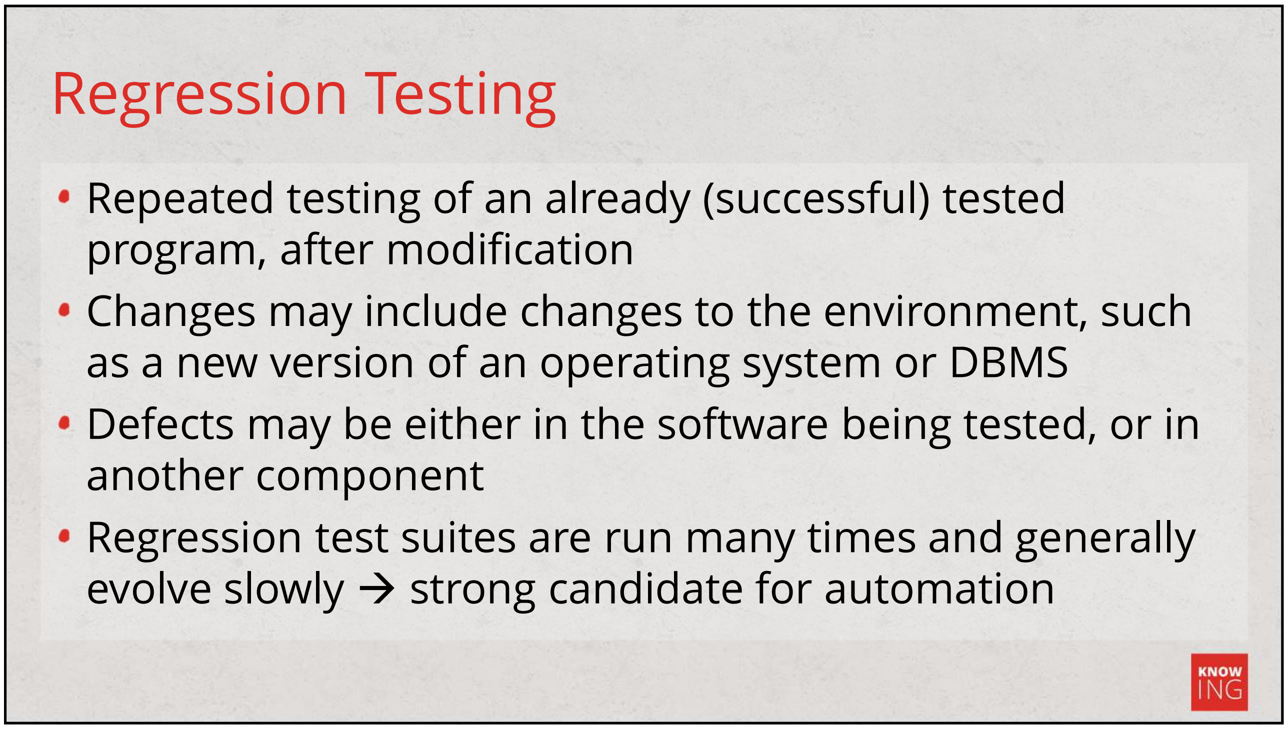
98



Confirmation Testing

- After a defect is fixed, all test cases that failed due to the defect should be re-executed on the new software version
- The software may also be tested with new tests if, for instance, the defect was missing functionality
- At the very least, the steps to reproduce the failure(s) caused by the defect must be re-executed on the new software version
- The purpose of a confirmation test is to confirm whether the original defect has been successfully fixed

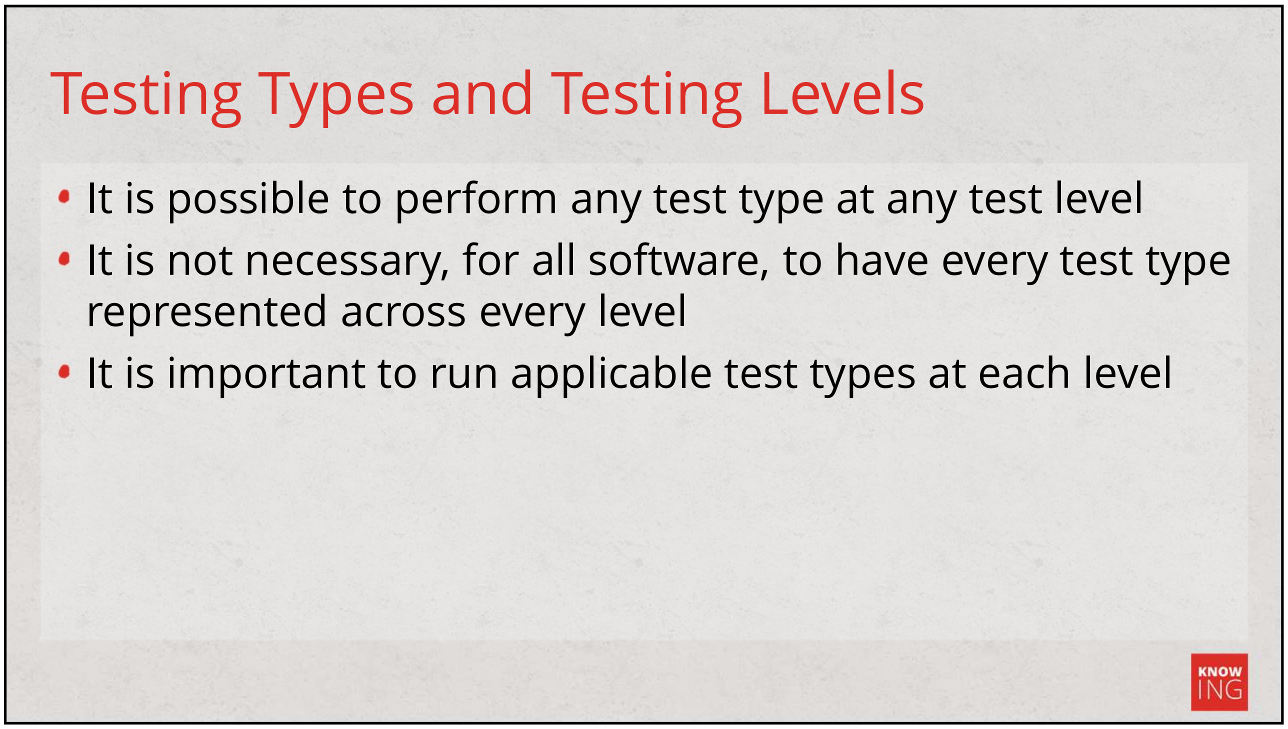
99



Regression Testing

- Repeated testing of an already (successful) tested program, after modification
- Changes may include changes to the environment, such as a new version of an operating system or DBMS
- Defects may be either in the software being tested, or in another component
- Regression test suites are run many times and generally evolve slowly → strong candidate for automation

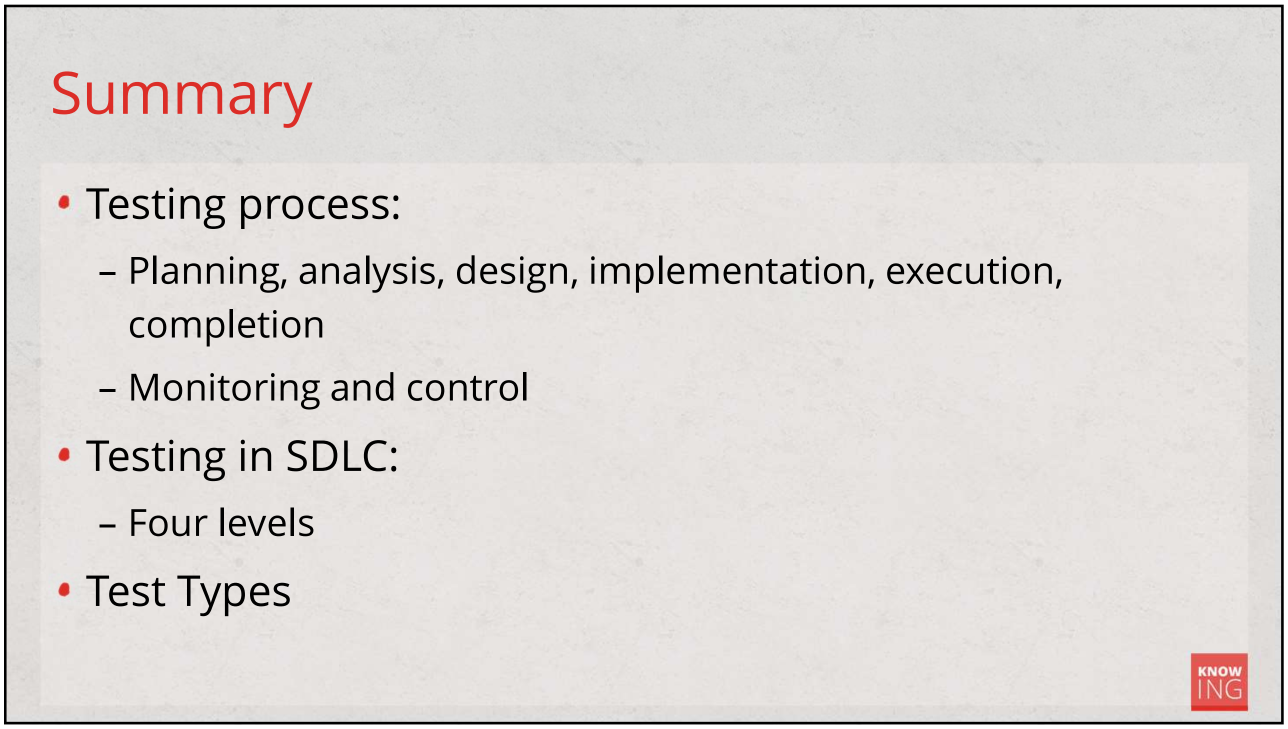
100



Testing Types and Testing Levels

- It is possible to perform any test type at any test level
- It is not necessary, for all software, to have every test type represented across every level
- It is important to run applicable test types at each level

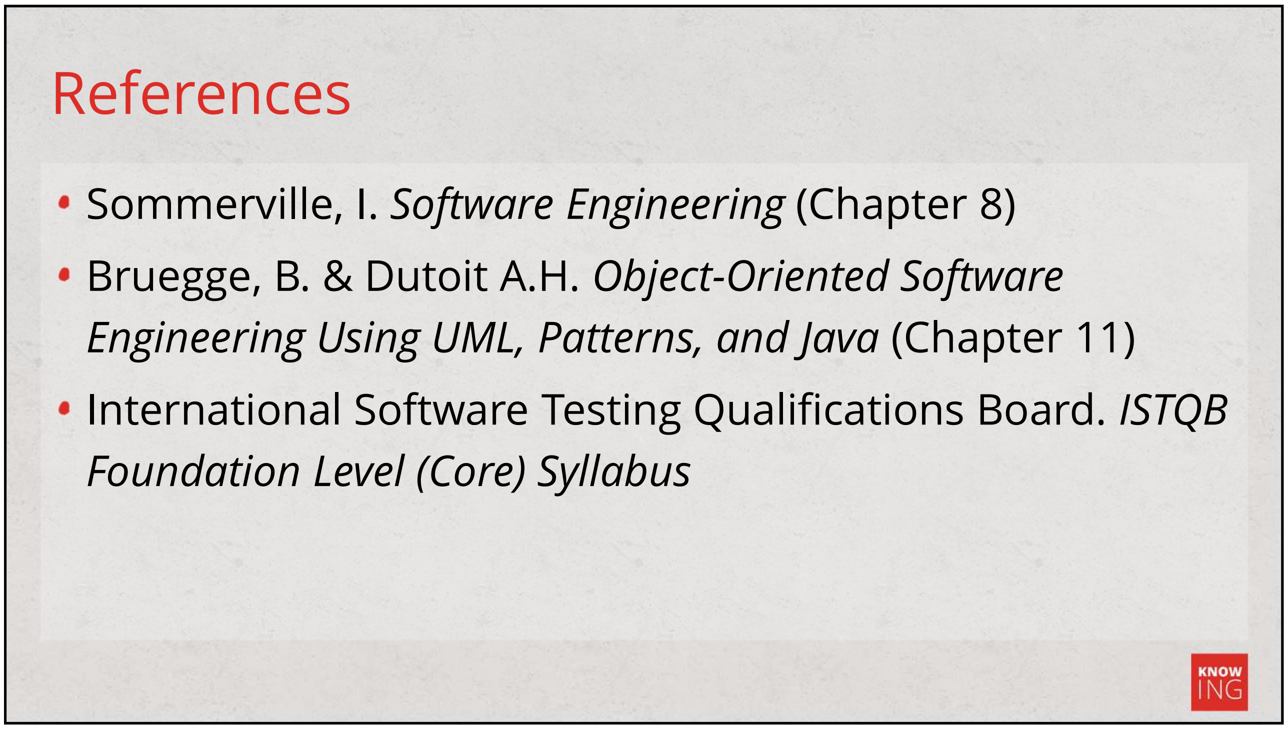
101



Summary

- Testing process:
  - Planning, analysis, design, implementation, execution, completion
  - Monitoring and control
- Testing in SDLC:
  - Four levels
- Test Types

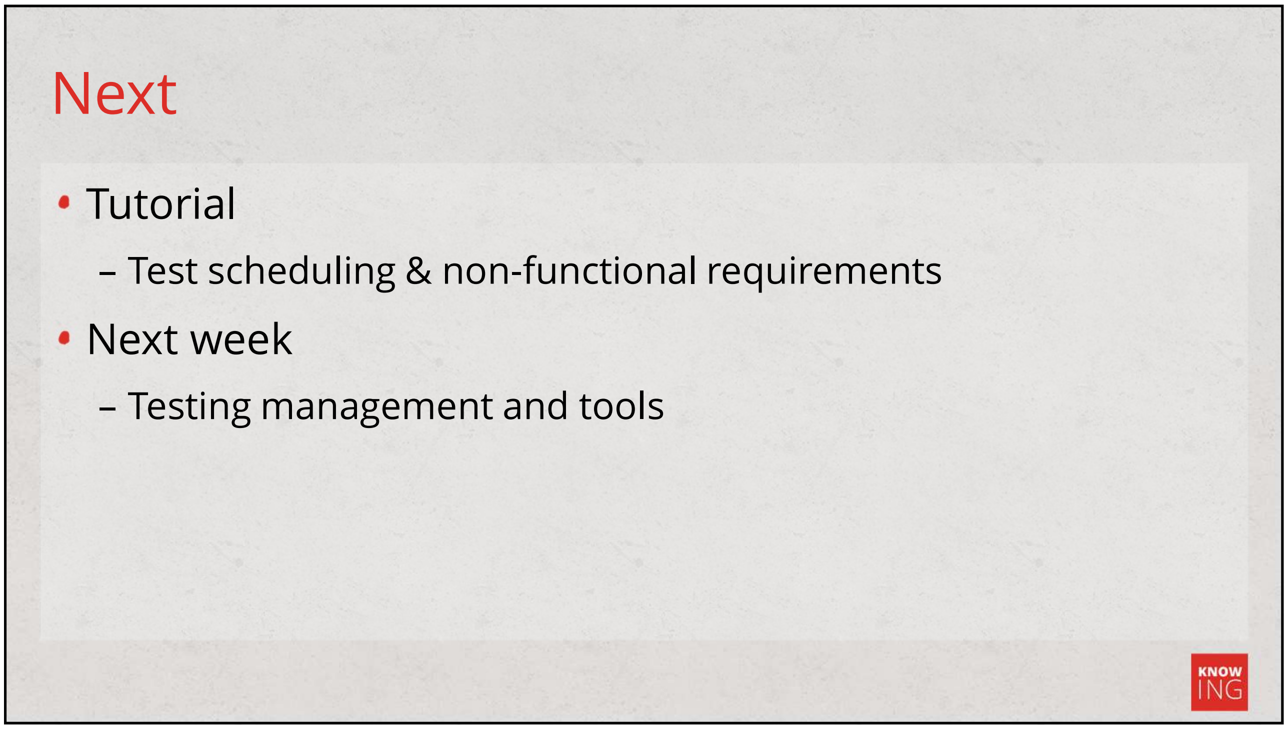
102



References

- Sommerville, I. *Software Engineering* (Chapter 8)
- Bruegge, B. & Dutoit A.H. *Object-Oriented Software Engineering Using UML, Patterns, and Java* (Chapter 11)
- International Software Testing Qualifications Board. *ISTQB Foundation Level (Core) Syllabus*

103



Next

- Tutorial
  – Test scheduling & non-functional requirements
- Next week
  – Testing management and tools

104