PACE UNIVERSITY
Seidenberg School of Computer Science and Information Systems

CS631G Software Verification

CHAPTER 8:

INTRODUCTION TO SOFTWARE TESTING

(W5)

Class instructor: Yuri Chernak, PhD

Outline

Outline (continued)

- Requirements Traceability Matrix (RTM)
- Defect reporting and management
- When to stop testing
- Test completion report
- qTest and HP ALM – test management tools
- Software inspections vs. testing
- Context-driven testing
- Exploratory testing explained
- Test metrics
- Key Points
- Exercise

Class Objectives

The objective of this class is to introduce software testing and testing practices.

At the end of this class you will:

- understand testing terms as defined in the IEEE standard;
- understand the levels of testing and test process phases;
- understand test documentation;
- learn about conventional test design techniques;
- learn about defect management and common test metrics.

IEEE Std.610 – Definition of Terms

Testing

1. The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the **system** or component.
2. The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the **features** of the software item.

Software Feature

A distinguishing characteristic of a software item (e.g., performance, portability, or functionality).

Test Plan

A document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who owns each task, and any risks requiring contingency planning (*see Slide 21*).

Test Design Specification

A document specifying the details of the test approach for a software feature or combination of software features and identifying the associated tests (*see Slide 23*).

IEEE Std.610 – Definition of Terms

Test Case
A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

Test Case Specification
A document specifying inputs, predicted (expected) results, and a set of execution conditions for a test item.

Test Incident Report (e.g., Defect Report)
A document reporting on any event that occurs during the testing process which requires investigation.

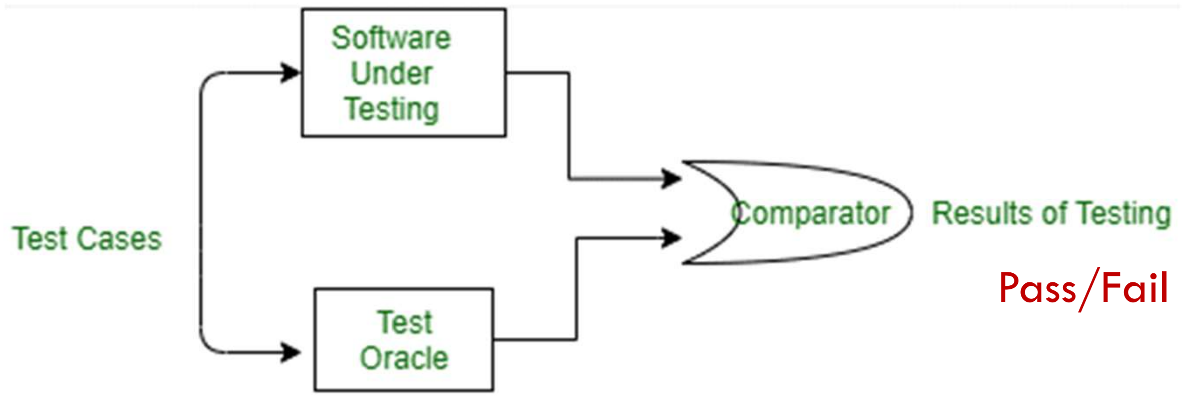
Test Execution Log
A chronological record of relevant details about the execution of tests.

Test Summary Report (a.k.a. Test Completion Report)
A document summarizing testing activities and results. It also contains an evaluation of the corresponding test items and test exit criteria (*see Slides 42, 43*).

What is a Test Oracle?

Test Oracle Definition

- A Test Oracle is a mechanism used in software engineering and software testing to determine whether a test has passed or failed.
- A Test Oracle provides a standard for assessing the correctness of the software's output. It serves as a benchmark for evaluating the correctness of a software system's behavior, i.e., it defines what the expected result should be for a given set of inputs or conditions.
- The use of Test Oracles involves comparing the output(s) of the system under test, for a given test-case input, to the output(s) that the oracle determines that product should have (see the figure below).



Types of Test Oracles

A Test Oracle serves as a benchmark for evaluating the correctness of a software system's behavior, a.k.a., ***test basis***. It enhances the effectiveness of testing efforts by providing an objective and consistent basis for comparison between expected and actual outcomes, supporting quality assurance and the overall software development lifecycle.

Conventional Types of Test Oracles

1. **Specified** oracles are typically associated with formalized approaches to software modeling, e.g., stated software requirements. Specified oracles are commonly used in formal requirements-based testing.
2. **Derived** oracles differentiate correct and incorrect behavior by using information derived from artifacts of the system, i.e., project documentation.
3. **Implicit** oracles rely on the tester’s intuition and experience to determine whether the output of the system under test is correct or not. Implicit oracles are commonly used in exploratory testing.

Challenges Finding Test Oracles for AI Systems

- Testing AI systems poses unique challenges, and finding reliable **Test Oracles** is particularly complex due to the indeterministic nature of artificial intelligence.
- The indeterministic nature of AI systems refers to the fact that, given the same input under identical conditions, an artificial intelligence model or system may produce different outputs on different occasions.

Here are some challenges associated with finding **Test Oracles** for testing AI systems:

1. **Lack of Ground Truth.** AI systems often make predictions or decisions based on complex algorithms and models. Determining the ground truth or the correct output for certain inputs can be challenging, especially when dealing with subjective tasks or ambiguous data.
2. **Evolving Models.** AI models, especially in machine learning, are dynamic and can evolve over time as they learn from new data. This makes it difficult to establish a fixed set of expected outcomes, as the model's behavior may change with each update or retraining.
3. **Complex Decision-making Process.** AI systems, especially in areas like natural language processing and image recognition, often involve complex decision-making processes. Interpreting the rationale behind a specific decision made by the AI can be challenging, making it difficult to define clear expected outcomes.

Four Schools of Testing

Requirements-based Testing

- Requirements-based testing involves examining each requirement and developing a test or multiple tests for it.
- The main assumption of the requirements-based testing is that requirements are available, complete, and accurate, which may not be the case. Thus, testing is only as good as the requirements.
- Requirements-based tests are "black box" (or functional) tests. So long as relevant inputs and/or conditions produce expected results that demonstrate the requirements are met, how the software does it is not of concern.

Risk-based Testing

- Risk-based testing (RBT) is a type of software testing that functions as an organizational principle used to prioritize the tests of features and functions in software, based on the risk of failure, the function of their importance and likelihood or impact of failure.
- When we analyze a feature under test, we ask how and why it can fail, what will be the impact on users, etc.

Four Schools of Testing (continued)

Test-driven Development

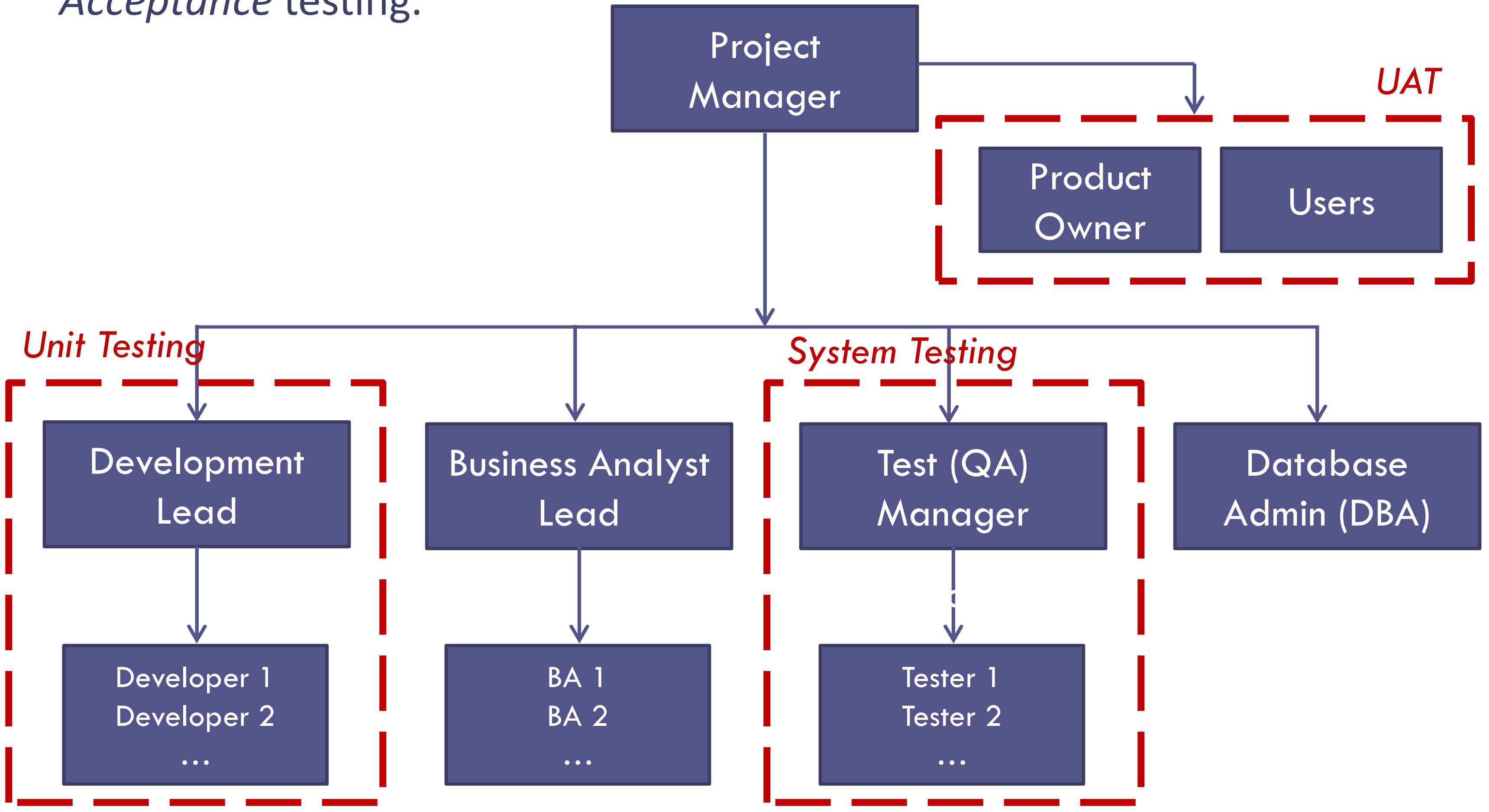
- Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development. TDD was introduced in ***Extreme Programming***.
- Tests are written before code and 'passing' the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don't move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of Agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

Exploratory Testing

- Exploratory testing is an approach to software testing that is concisely described as simultaneous learning, test design and test execution.
- Cem Kaner, who coined the term "exploratory testing" in 1993, now defines exploratory testing as "a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his/her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project."

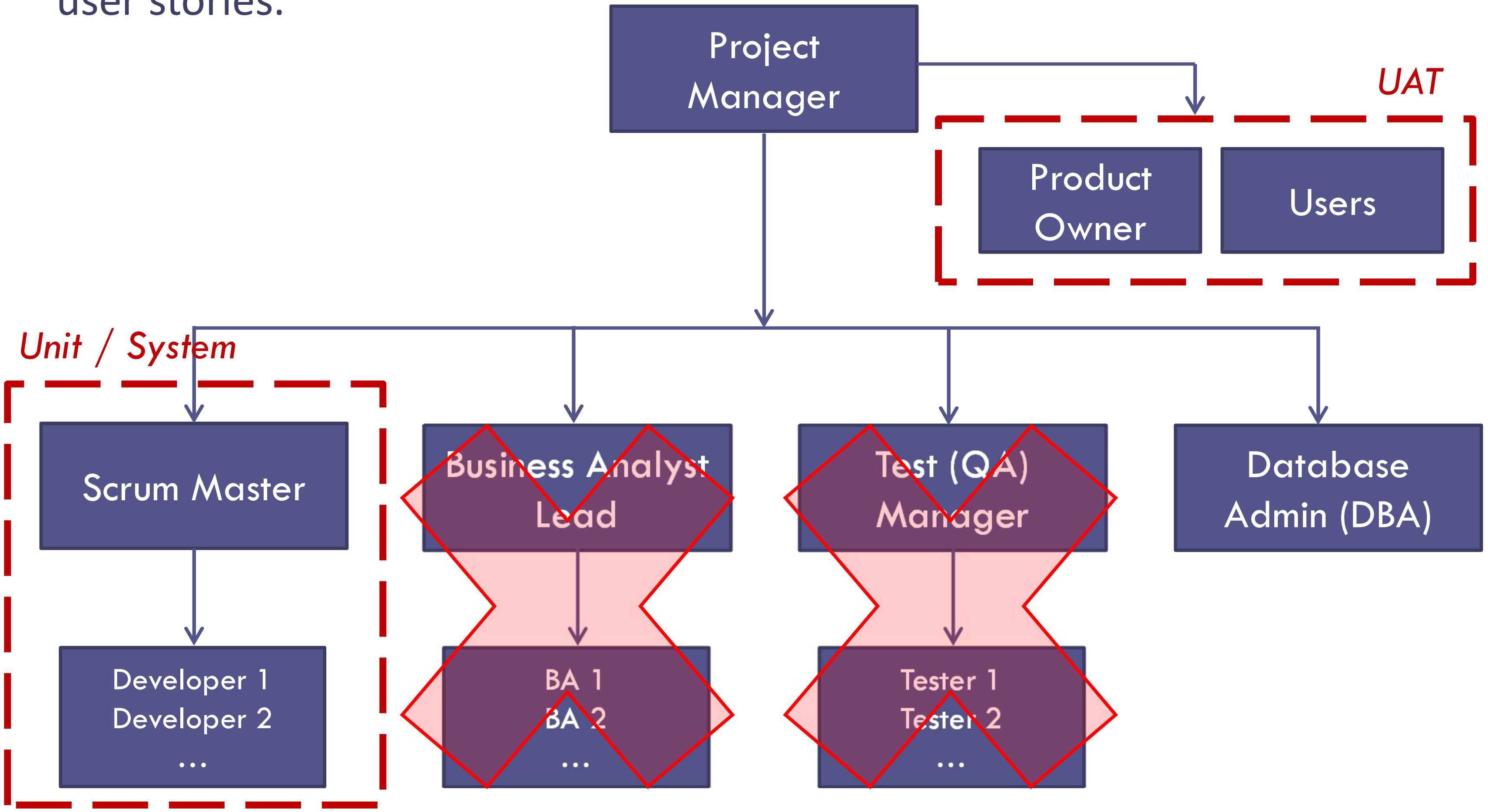
Who Tests Software?

On traditional projects, developers perform *Unit* testing, a QA team performs *System* testing, and the product owner and users perform *User Acceptance* testing.



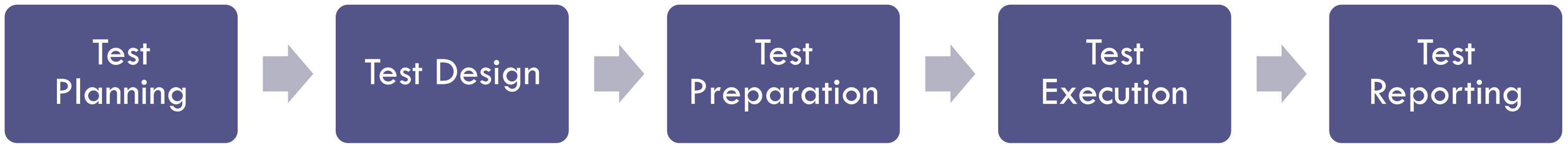
Who Tests Software? (continued)

On Agile projects, the whole development team performs unit and system testing and the product owner/users perform acceptance testing based on user stories.



Conventional Test Process Lifecycle

A conventional test process lifecycle includes five phases, where each phase has its own activities and deliverables:



-Test Strategy document (a.k.a. Master Test Plan)
-Test Plan document

-Test Design document
-Test Case specifications

-Test environment readiness
-Loaded test data
-Refined test schedule and confirmed testing resources

-Test execution status (test log)
-Defect reports
-Test execution progress report

-Test summary report
-Testing metrics

Two Missions of Software Testing

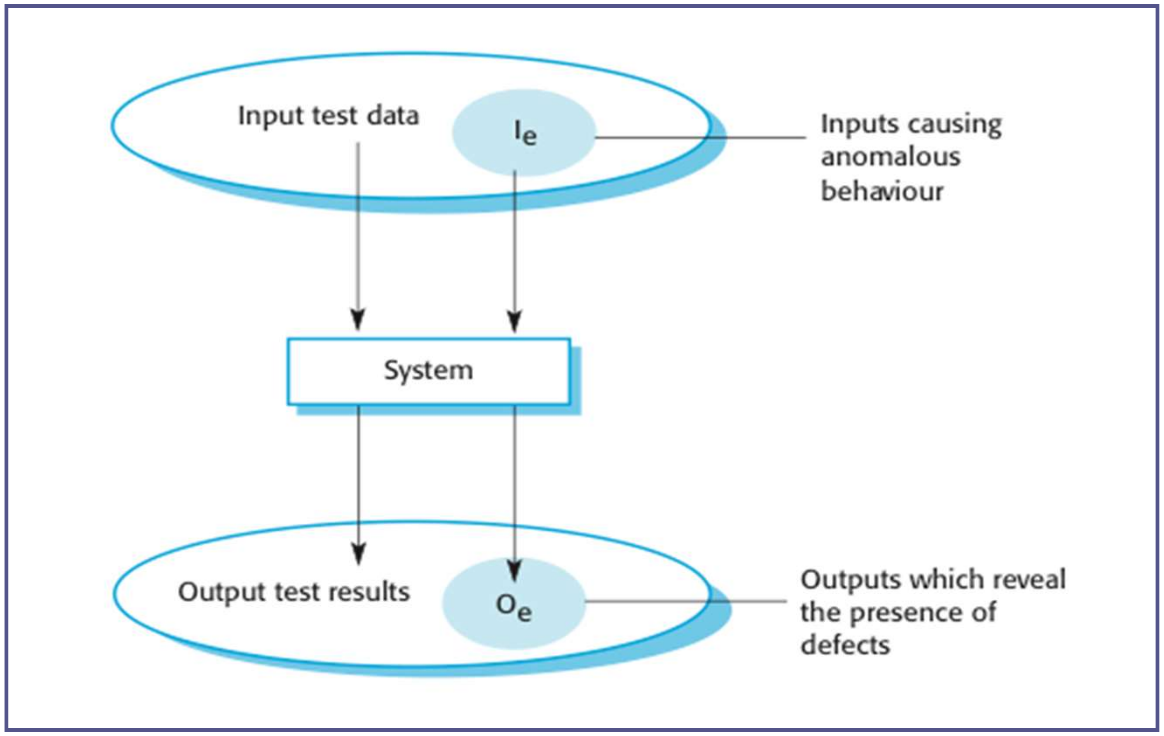
Defect Testing (a.k.a. Stability Testing)

- Mission – evaluate stability of a software product.
- To discover faults or defects in the software where its behavior is incorrect or not in conformance with its specification.
- A successful test is a test that makes the system perform incorrectly, i.e., it is effective to expose a defect in the system.

Validation Testing (a.k.a. User Acceptance Testing)

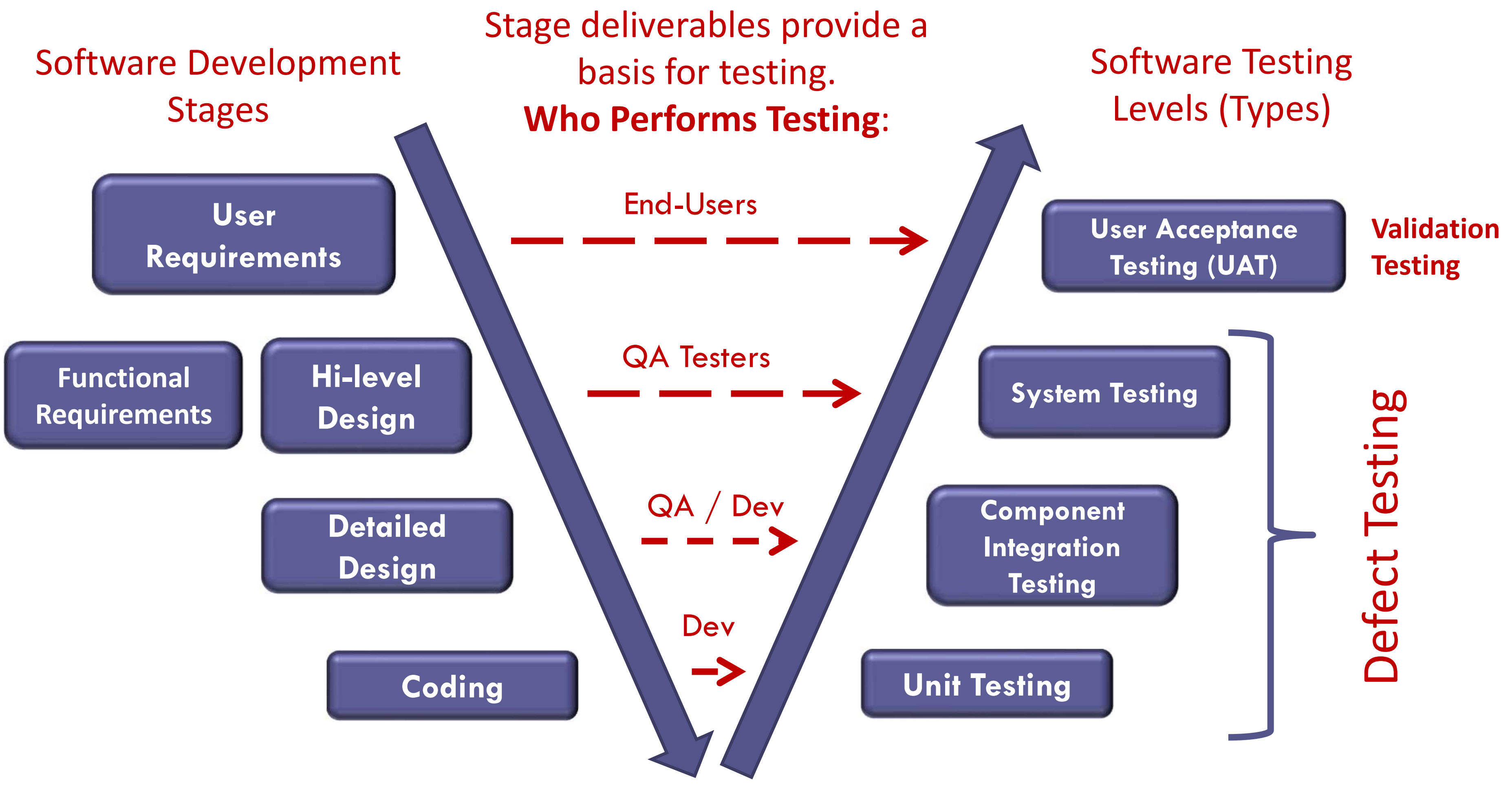
- Mission – evaluate a software product's suitability to support a customer's business.
- To demonstrate to the developer and the system customer that the software meets its requirements.
- A successful test shows that the system operates as intended, can support the customer's business.

Defect Testing Concept


Input test data
I_e
Inputs causing anomalous behaviour
System
Output test results
O_e
Outputs which reveal the presence of defects

- When designing test cases, we select challenging inputs that are more effective in finding defects.
- Testing can only show the presence of defects, not their absence.

V-Model: Conventional Test Levels (Types)


Software Development Stages
Stage deliverables provide a basis for testing.
Who Performs Testing:
Software Testing Levels (Types)
User Requirements
Functional Requirements
Hi-level Design
Detailed Design
Coding
End-Users
QA Testers
QA / Dev
Dev
User Acceptance Testing (UAT)
System Testing
Component Integration Testing
Unit Testing
Validation Testing
Defect Testing

Conventional V-Model

Test Levels Explained

Test Level	Performed By	Description
Acceptance Testing	End Users (sometimes with the assistance of business analysts)	1. Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. 2. Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component for production.
System Testing	QA team	Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.
Component Integration Testing	QA/Developers	Testing of individual hardware or software components or groups of related components. In integration, we assemble units together into subsystems and finally into systems.
Unit Testing	Developers	A unit is the smallest piece of software that a developer creates. Testing of individual hardware or software units or groups of related units.

Other Test Types

Regression Testing

- Regression testing is testing the system to check that changes have not ‘broken’ previously working code.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.

Smoke Testing

- Executing a limited sub-set of regression tests for a new build is known as “smoke testing”.
- Smoke testing, also known as build verification testing or build acceptance testing, is a preliminary testing phase conducted on a software build to determine whether it is stable enough for more in-depth testing.
- Smoke testing is focused on the most critical and basic functionalities of the software. It doesn't delve into detailed testing but aims to catch showstopper issues that would prevent further testing.

Performance Testing

- Provides a systematic evaluation of the system’s performance characteristics to demonstrate they meet the operational (a.k.a. non-functional) requirements.
- Performance testing is an engineering discipline and requires special tools and highly-technical skills.

Other Test Types

End-to-End (E2E) Testing

- In large IT departments, many systems are interconnected and exchange data.
- When planning testing for a given system, we need to analyze the impact on external systems.
- The test strategy should include the impacted external systems and this type of testing is known as “E2E” testing.
- In the financial industry, this is also known as “Front-to-Back” (F2B) testing.

Operational Acceptance Testing (OAT)

- **OAT** is used to conduct operational readiness (pre-release) of a product, service or system as part of a quality management system.
- **OAT** is a type of non-functional software testing, common in IT departments with **segregation of duties** between the application development teams and production management and support groups.

Test Automation

- In software testing, test automation is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes.
- Test automation can be implemented for various test types – unit testing, smoke testing, functional regression testing, and performance testing.
- Developing the automated testing scripts is a software development project.
- Automated software scripts require maintenance that can be very expensive.
- Test automation is a special skill on the IT job market.
- Five best automation tools: http://automated-360.com/automation-tools/5-best-test-automation-tools/

IEEE Std. 829 – Test Plan Document

Test Plan Purpose - to prescribe the scope, approach, resources, and schedule of the testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with this plan.

A Test Plan document sections:

a) Test plan identifier
b) Introduction
c) Test items
d) Features to be tested
e) Features not to be tested
f) Approach
g) Item pass/fail criteria
h) Suspension criteria and resumption requirements
i) Test deliverables
j) Testing tasks
k) Environmental needs
l) Responsibilities
m) Staffing and training needs
n) Schedule
o) Risks and contingencies
p) Approvals

Information in these sections changes for different releases.

Test Plan document is W10 deliverable. See examples in Classes course portal.

When to Produce a Formal Test Plan

- A detailed test plan is most useful for new development projects:
    - On critical new development projects, stakeholders need a clear understanding how a software product will be tested, how the test project is planned, what will be the testing deliverables, how do you know when to stop testing, what will be the basis for sign off, etc.
- For maintenance projects, the test process is already known and proven, hence, most of the standard test plan sections do not provide value for the project stakeholders.
- On Agile projects, a formal test plan document is not used. Instead, a direct communication with users is considered more important. A Road Map (iteration planning document) can also be used to define the functional scope of testing.

IEEE Std. 829 – Test Design Specification

Document Section	Description
Test design specification identifier	Specify the unique identifier assigned to this test design specification. Supply a reference to the associated test plan, if it exists.
Features to be tested	Identify the test items and describe the features and combinations of features that are the object of this design specification. For each feature or feature combination, a reference to its associated requirements in the item requirement specification or design description should be included.
Approach refinements	Specify refinements to the approach described in the test plan. Include specific test techniques to be used. The method of analyzing test results should be identified (e.g., comparator programs or visual inspection). Specify the analysis that provides a rationale for test case selection.
Test identification	List the identifier and a brief description of each test case associated with this design. A particular test case may be identified in more than one test design specification.
Feature pass/fail criteria	Specify the criteria to be used to determine whether the feature or feature combination has passed or failed.

When to Produce a Formal Test Design Document

- A formal Test Design document is mostly used on requirements-based testing projects.
- Requirements can be developed at various level of detail and abstraction.
- When requirements are specified by large chunks, e.g., use cases or user stories, identifying test cases can be challenging.
- As a solution, first we can develop test ideas for a large requirement, capture them in a test design specification and then use this document as a basis for detailing test cases.
- Sometimes, a good test design document is sufficient to execute testing and detailed test cases are not developed.

Test Design document is Week 11 deliverable. See examples in Classes, course portal.

Use-Case Testing Example

Use Case: 01.01 Create Guest Reservation

Use-Case Description

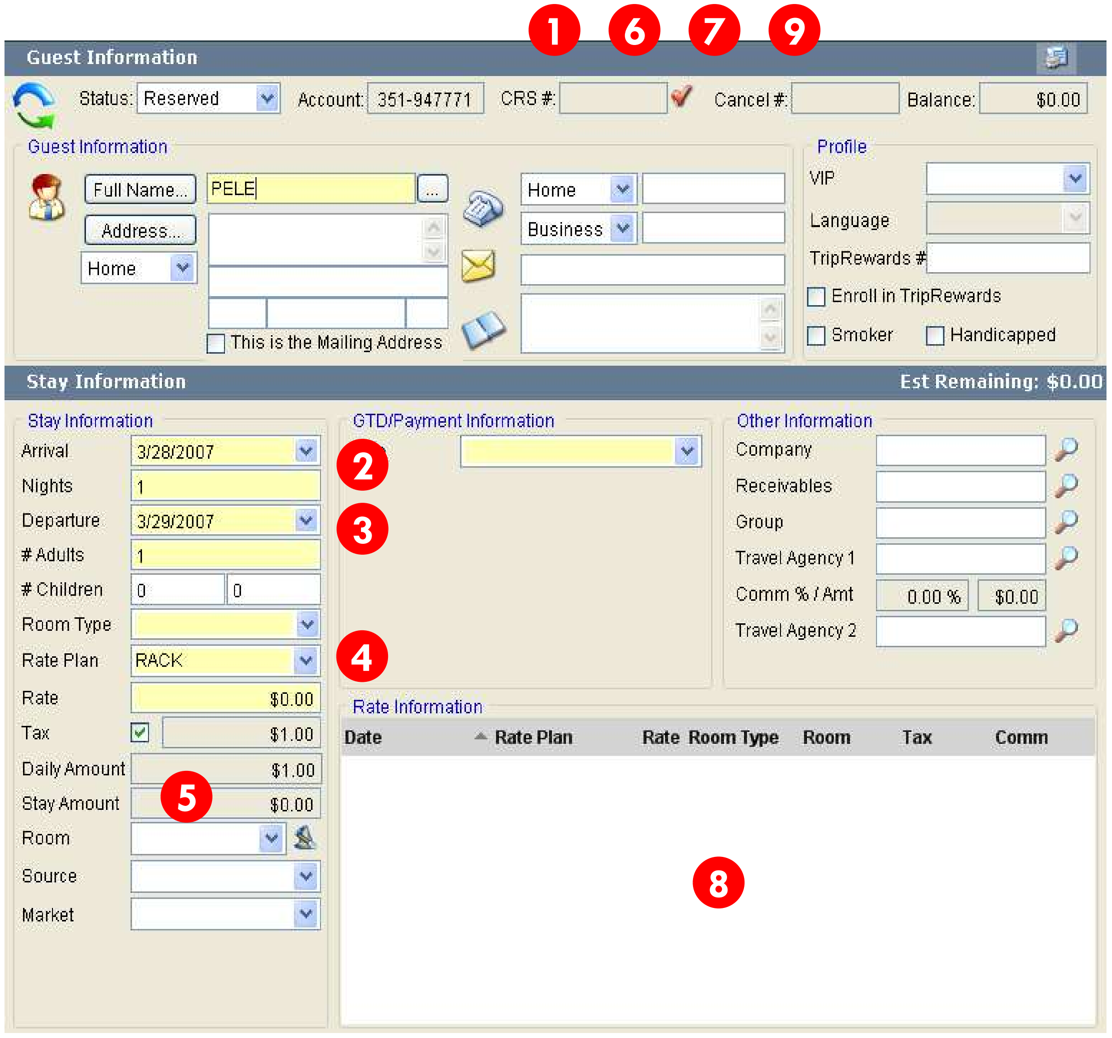
1. The use case begins when the Front Desk Clerk intends to create a new guest reservation (or check-in a walk-in guest). [Entitlements]
2. User enters a guest's personal, stay and payment information. [Field Validation]
3. System provides the Rate & Plan information and available room inventory based on the stay information. [System Interface]
4. User selects the room type and rate plan. [Data-Driven Defaults]
5. System displays the stay amount. [Calculations]
6. User submits the reservation.
7. System creates a new reservation and sends the local reservation to the Central Reservation system. [Connectivity, Data Flow Out, System Interface]

Crosscutting Concerns

- The use-case scenario is tangled with supplementary requirements.
- A Test Design specification can help us better identify test cases.

Using an RCT for Test Design

	RCT Fragment Concern Types	Create Guest Reservation
	Core Functionality	1
	GUI - User Interface	1
	Crosscutting Concerns	
1	ET - Entitlements	1
2	FV - Field Validation	1
3	DDV - Data-Dependency Validation	1
4	DDD - Data-Driven Defaults	1
5	CL - Calculations	1
	CC - Concurrency	0
6	CN - Connectivity	1
	TST - Transaction Status	0
	DF-In - Data Flow In	0
7	DF-Out - Data Flow Out	1
8	SI-In - System Interface In	1
9	SI-Out - System Interface Out	1



Guest Information

Status:	Reserved
Account:	351-947771
CRS #:	
Cancel #:	
Balance:	$0.00

Guest Information

Full Name...	PELE
Address...	
Home	
Business	

☐ This is the Mailing Address

Profile

VIP	
Language	
TripRewards #	

☐ Enroll in TripRewards
☐ Smoker ☐ Handicapped

Stay Information Est Remaining: $0.00

Stay Information

Arrival	3/28/2007
Nights	1
Departure	3/29/2007
# Adults	1
# Children	0 0
Room Type	
Rate Plan	RACK
Rate	$0.00
Tax	✓ $1.00
Daily Amount	$1.00
Stay Amount	$0.00
Room	
Source	
Market	

GTD/Payment Information

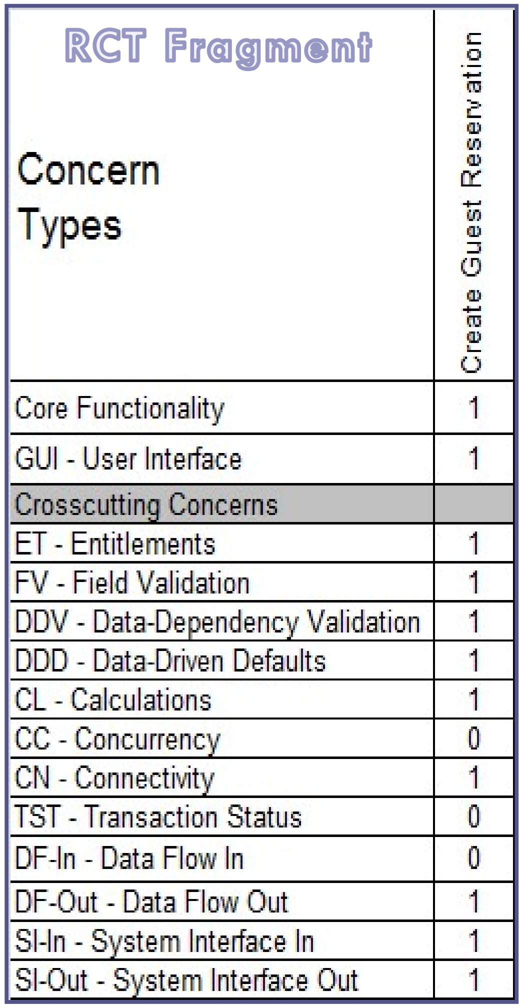
Other Information

Company	
Receivables	
Group	
Travel Agency 1	
Comm % / Amt	0.00 % $0.00
Travel Agency 2	

Rate Information

Date	Rate Plan	Rate	Room Type	Room	Tax	Comm

Using an RCT for Test Design (continued)



RCT Fragment

Concern Types	Create Guest Reservation
Core Functionality	1
GUI - User Interface	1
Crosscutting Concerns	
ET - Entitlements	1
FV - Field Validation	1
DDV - Data-Dependency Validation	1
DDD - Data-Driven Defaults	1
CL - Calculations	1
CC - Concurrency	0
CN - Connectivity	1
TST - Transaction Status	0
DF-In - Data Flow In	0
DF-Out - Data Flow Out	1
SI-In - System Interface In	1
SI-Out - System Interface Out	1

Test Design Specification (IEEE Std. 829)

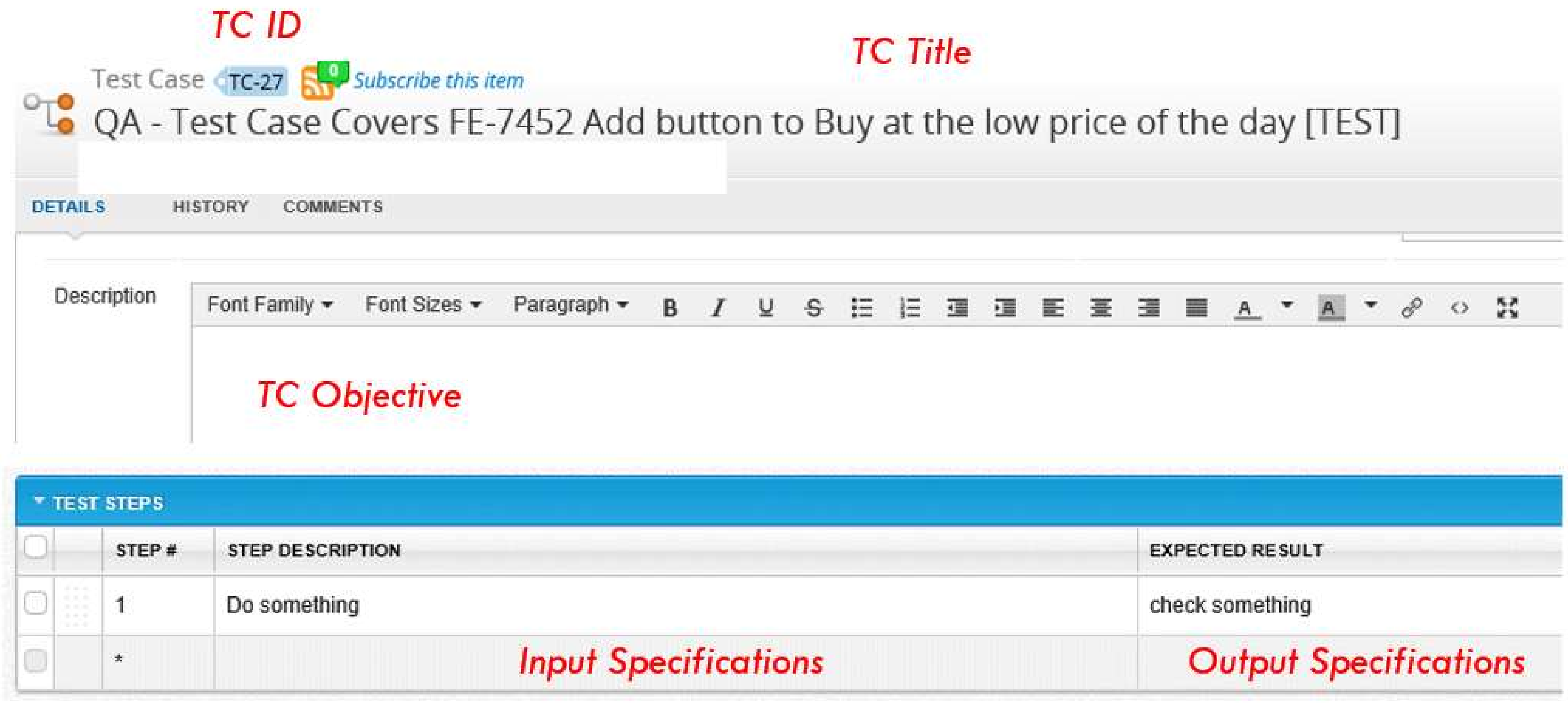
Document Section	Guidelines
Test design specification identifier	TD_01.01 Create Guest Reservation
Features to be tested	Core functionality, GUI, all applicable crosscuts (see the RCT).
Approach refinements	Describe your approach to designing and executing testing for this use case.
Test identification	List test case ID and titles, identified for this use case.
Feature pass/fail criteria	A given feature has passed testing, if all related test cases passed test execution.

IEEE Std. 829 – Test Case Specification

Document Section	Description
Test Case Specification Identifier	Specify the unique identifier assigned to this test case specification.
Test Items	Identify and briefly describe a test objective, i.e., a feature(s) to be exercised by this test case.
Input Specifications	Specify each input required to execute the test case.
Output Specifications (Expected Results)	Specify all of the outputs required, i.e., expected results. Provide the exact value for each expected output.
Environmental Needs (optional)	Specify the characteristics of the hardware or test data required to execute this test case.
Special Procedural Requirements (optional)	Describe any special constraints (e.g., test data) on the test procedures that execute this test case.
Inter-case Dependencies (optional)	List the identifiers of test cases that must be executed prior to this test case.

A Test Design document is intended to identify test ideas and clarify an approach to testing, whereas a Test Case specification is more detailed, it specifies input conditions and expected results for a given test idea.

qTest Tool: Test Case Screen Example



QA - Test Case Covers FE-7452 Add button to Buy at the low price of the day [TEST]

Description

STEP #	STEP DESCRIPTION	EXPECTED RESULT
1	Do something	check something
*		

Black-box vs. White-box Testing

There are two fundamental approaches to software testing – black-box and white-box testing.

Black-box Testing

- A method of software testing that examines the functionality of an application without the knowledge of its internal structure or workings.
- Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.
- This approach is used for **Functional and User Acceptance** testing.

White-box Testing

- A method of testing software that tests internal structures or workings of an application, as opposed to its functionality.
- In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases.
- This approach is used for **Unit** testing.

Black-box vs. White-box Testing Techniques

When designing test cases for each of the two approaches, some formal test design techniques can be used:

Black-box Techniques	White-box Techniques
Equivalence class partitioning	Control flow testing
Boundary-value analysis	Data flow testing
Cause-effect graphing	Branch testing
Decision table testing	Statement coverage
State-transition testing	Decision coverage

For a detailed discussion of these techniques, see the book:
L. Copeland "A Practitioner's Guide to Software Test Design"

Black-box: Equivalence Class Testing

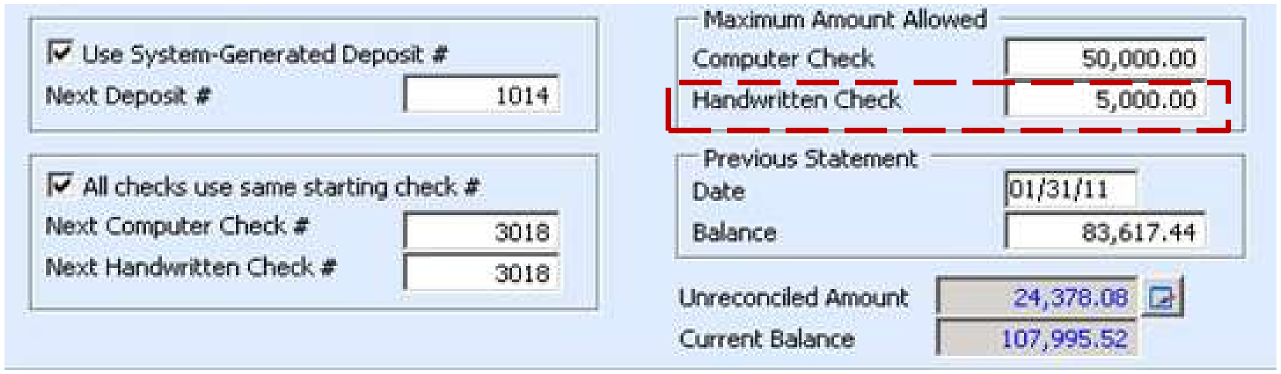
- Equivalence Class Partitioning is a software testing technique that divides the range of input data of a software feature into partitions of equivalent data classes from which test cases can be derived.
- This technique is used to reduce the number of test cases, where test cases are designed to cover each partition at least once.
- An advantage of this approach is reduction in the time required for testing a software due to lesser number of test cases.
- Equivalence partitioning is typically applied to the inputs of a tested component, i.e., to the data entry fields on an application screen.
- The equivalence partitions are usually derived from the supplementary requirements for data-entry inputs.

Black-box: Boundary Value Testing

- Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values in a range.
- The boundary value analysis technique is a variant of the equivalence class partitioning technique, which focuses on the boundaries of each equivalence class, e.g., on, above, and below each class.
- The boundary value analysis technique is effective as many defects are commonly found around the boundaries of data entry values.

Example of a Test Case Specification

Accounting application, Bank Reconciliation module.
Requirement – a Max amount allowed for handwritten checks is a mandatory field, its value should not exceed $5,000.00.



☑ Use System-Generated Deposit #	
Next Deposit #	1014
☑ All checks use same starting check #	
Next Computer Check #	3018
Next Handwritten Check #	3018

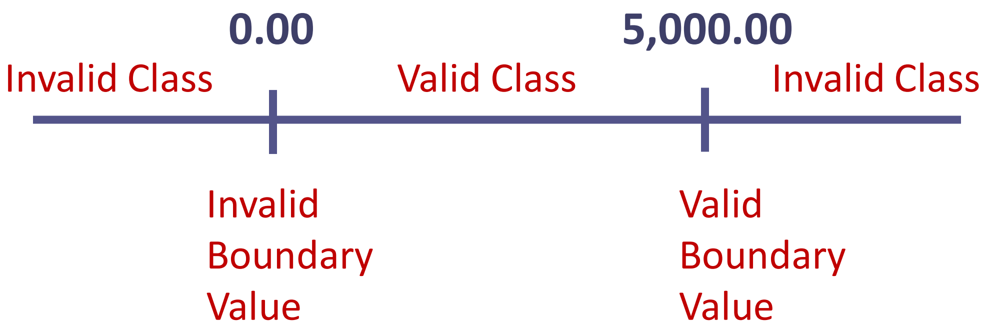
Maximum Amount Allowed

Computer Check	50,000.00
Handwritten Check	5,000.00

Previous Statement

Date	01/31/11
Balance	83,617.44
Unreconciled Amount	24,378.08
Current Balance	107,995.52

Test Case specification is Week 12, 13 deliverable.
See examples in Classes course portal.



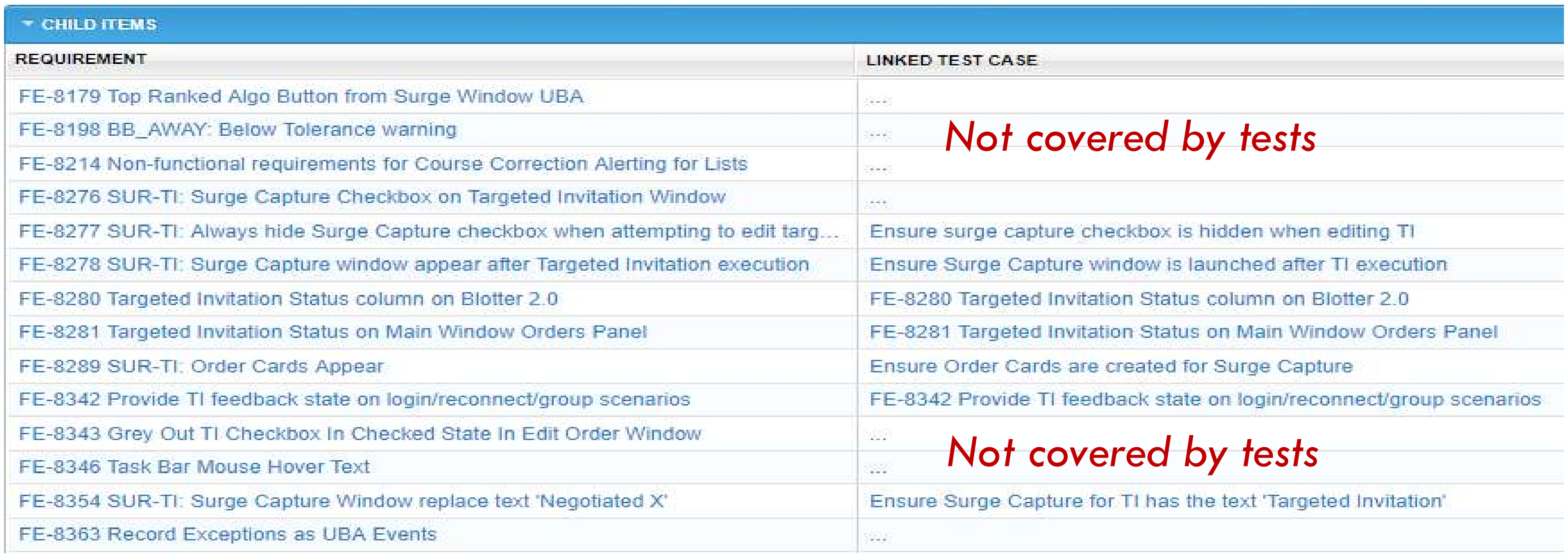
List of test cases (inputs):

1. Blank field – not accepted
2. 0.00 – not accepted
3. 1,000.00 – accepted
4. 5,000.00 – accepted
5. 5,000.01 – not accepted

Requirements Traceability Matrix

- In requirements-based testing, test cases should be designed for each requirement included in the testing scope.
- A requirements traceability matrix (RTM) is used to assess test coverage and demonstrate that a set of test cases is complete, i.e., every requirement has a least one test case associated.
- RTM is commonly created and maintained using a test management tool, e.g., qTest (see an example below).

RTM Example



CHILD ITEMS

REQUIREMENT	LINKED TEST CASE
FE-8179 Top Ranked Algo Button from Surge Window UBA	....
FE-8198 BB_AWAY: Below Tolerance warning	....
FE-8214 Non-functional requirements for Course Correction Alerting for Lists	....
FE-8276 SUR-TI: Surge Capture Checkbox on Targeted Invitation Window	....
FE-8277 SUR-TI: Always hide Surge Capture checkbox when attempting to edit targ...	Ensure surge capture checkbox is hidden when editing TI
FE-8278 SUR-TI: Surge Capture window appear after Targeted Invitation execution	Ensure Surge Capture window is launched after TI execution
FE-8280 Targeted Invitation Status column on Blotter 2.0	FE-8280 Targeted Invitation Status column on Blotter 2.0
FE-8281 Targeted Invitation Status on Main Window Orders Panel	FE-8281 Targeted Invitation Status on Main Window Orders Panel
FE-8289 SUR-TI: Order Cards Appear	Ensure Order Cards are created for Surge Capture
FE-8342 Provide TI feedback state on login/reconnect/group scenarios	FE-8342 Provide TI feedback state on login/reconnect/group scenarios
FE-8343 Grey Out TI Checkbox In Checked State In Edit Order Window	....
FE-8346 Task Bar Mouse Hover Text	....
FE-8354 SUR-TI: Surge Capture Window replace text 'Negotiated X'	Ensure Surge Capture for TI has the text 'Targeted Invitation'
FE-8363 Record Exceptions as UBA Events	....

Not covered by tests

Not covered by tests

White-box: Common Testing Techniques

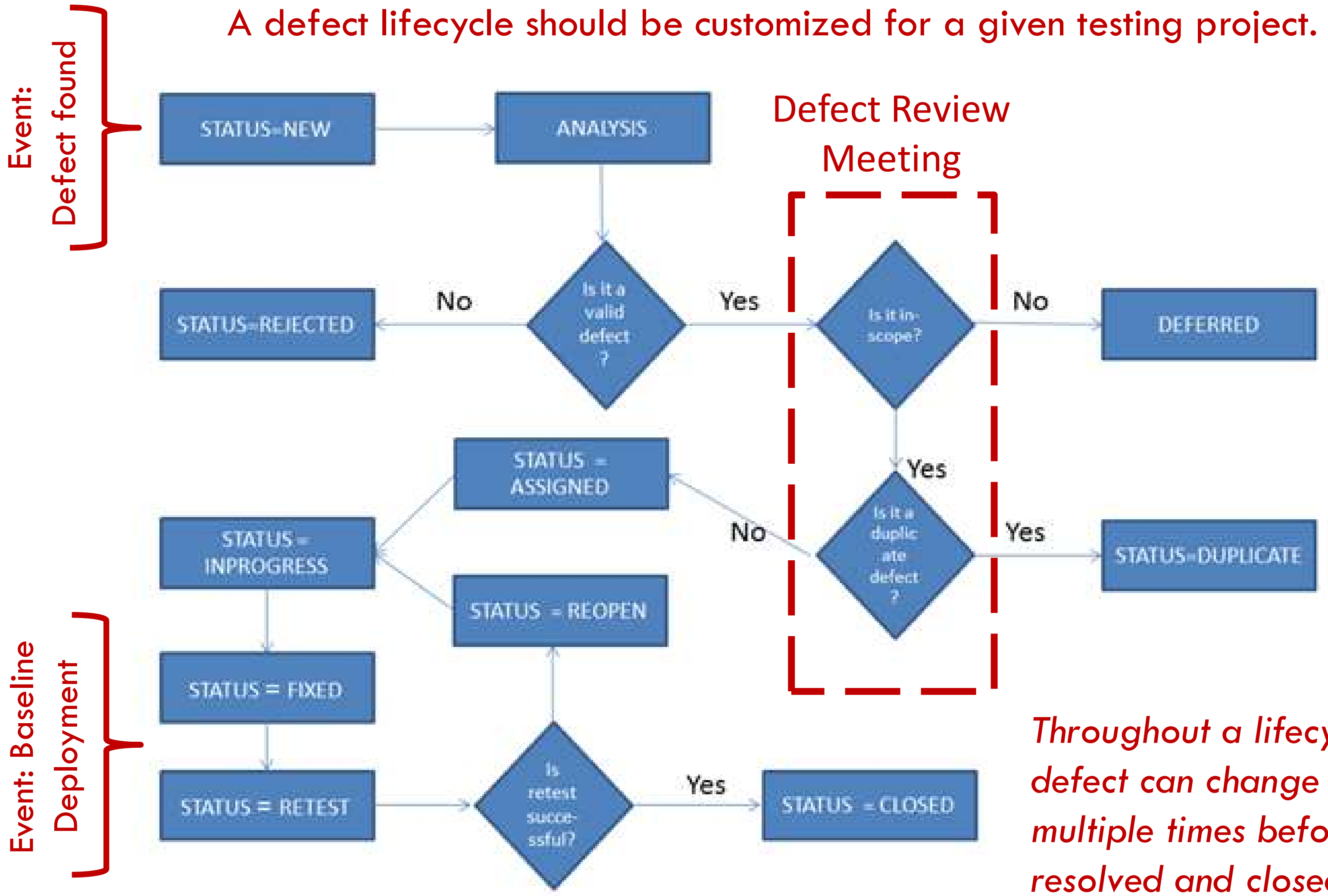
Technique	Description
Control-flow testing	Control-flow testing is based on a control-flow graph. The path through the graph is analyzed and provides a basis for designing test cases. It applies to almost all software and it is mostly applied to relatively small programs or segments of larger programs.
Branch testing	This technique is running a series of tests to ensure that all branches in source code are tested at least once.
Statement coverage	This technique is aimed at exercising all programming statements with minimal tests.
Path coverage	This technique corresponds to testing all possible paths which means that each statement and branch is covered.
Data-flow testing	In data-flow testing, the control flow-graph is annotated with information about how the program **variables** are **defined** and **used**.

Reporting Software Defects

- The most important mission of testers is finding and reporting software defects.
- Defects are found in deliverables produced by other team members, e.g., business analysts (requirements) or developers (source code).
- Once a defect is found, it needs to be communicated to a party who is the owner of the deliverable.
- However, it is not always evident where a fix should be applied, and some investigation of the defect origin is commonly needed.
- Hence, defect reporting is a communication process where a given defect changes hands from the person who found it to the next party who investigates it, then to the party who fixes it, etc.
- Once the defect is fixed, it goes back to the Tester for re-testing and closing.
- On large projects, the number of new defects reported daily can be from 15 to 30 defects. On such projects effective defect management becomes a critical activity:
  - Defect Management = New Defect Prioritization + Defect Status Tracking

Defect Lifecycle Example

A defect lifecycle should be customized for a given testing project.



Throughout a lifecycle, a defect can change hands multiple times before it is resolved and closed.

Tips to Write Effective Defect Reports

Tips	Descriptions
It’s all in Defect Title	The defect title should always be meaningful and self-descriptive enough to tell about the defect. Keep the title short & should properly convey the defect summary without further reading. Each defect should be clearly identifiable by the defect title.
Steps To Reproduce The Defect	Software defects are very sensitive & they occur when there is a condition in a software product which does not meet a software requirement or end-user expectations. While reporting the defect, you should add the proper steps to reproduce the defect.
Be Specific	The defect description should be to the point and do not write down the essay about the problem.
Defect Linking	Add the references to the specifications or other related defect number while adding the defect. For example, in Jira defects can be linked to their related user stories.
Don’t take business decisions, only pass the Information	You should be objective while reporting defects. You should add the actual result in defect & the expected result if it is specified in the requirement specification document. Don’t take business decisions if anything is not clear , ask for additional information, nut do not pass any kind of judgments by your own.
Capture ***Severity*** and ***Priority*** of defects	Severity & Priority defect attributes play a vital role in the defect management. The **Severity** describes the impact of the bug on application functionality. The **Priority** indicates how quickly a bug should be fixed.

When to Stop Testing

- The longer you test software, the more likely you will be finding defects.
- How do you know when to stop testing?
  - The rule is usually defined in a **Test Plan** document as the Test Exit criteria that should be agreed with a project manager and key project stakeholders.
  - The data to evaluate the Test Exit criteria and the evaluation conclusion are usually presented in a Test Completion Report (see next slide).

Exit Criteria Example for System Testing

Test Design Completeness =>	• All functional requirements are covered by test cases.
Test Execution Completeness =>	• All test cases have been executed.
Product Stability Assessment =>	• No defects of Critical & High severity remain open.

Test Summary (a.k.a. Test Completion) Report

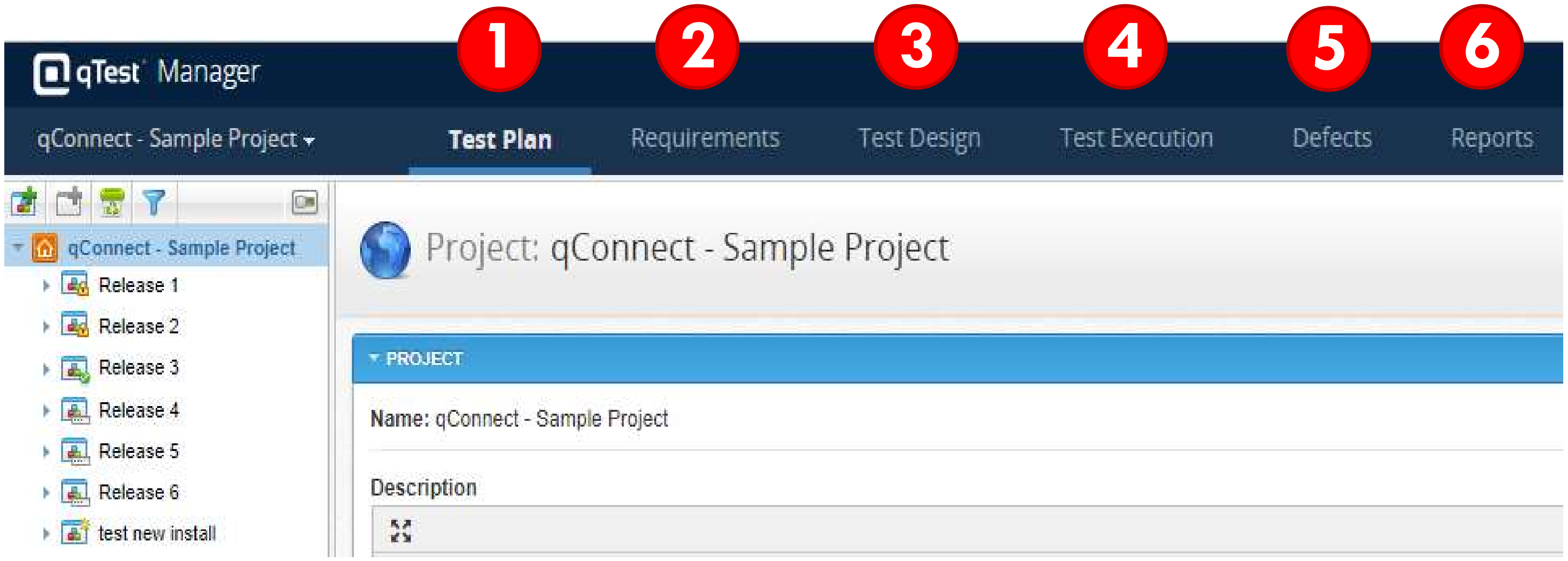
IEEE Std.829 definition:

Document Section	Description
Test summary report identifier	Specify the unique identifier assigned to this test summary report.
Summary	Summarize the evaluation of the test items. Identify the items tested, indicating their version/revision level. Indicate the environment in which the testing activities took place. For each test item, supply references to the following documents if they exist: test plan, test design specifications, test procedure specifications, test item transmittal reports, test logs, and test incident reports.
Variances	Report any variances of the test items from their design specifications. Indicate any variances from the test plan, test designs, or test procedures. Specify the reason for each variance.
Comprehensive assessment	Evaluate the comprehensiveness of the testing process against the comprehensiveness criteria specified in the test plan if the plan exists. Identify features or feature combinations that were not sufficiently tested and explain the reasons.

Test Summary Report (continued)

Document Section	Description
Summary of results	Summarize the results of testing. Identify all resolved incidents and summarize their resolutions. Identify all unresolved incidents.
Evaluation	Provide an overall evaluation of each test item including its limitations. This evaluation shall be based upon the test results and the item level pass/fail criteria. An estimate of failure risk may be included.
Summary of activities	Summarize the major testing activities and events. Summarize resource consumption data, e.g., total staffing level, total machine time, and total elapsed time used for each of the major testing activities.
Approvals	Specify the names and titles of all persons who must approve this report. Provide space for the signatures and dates.

qTest – Test Management Tool



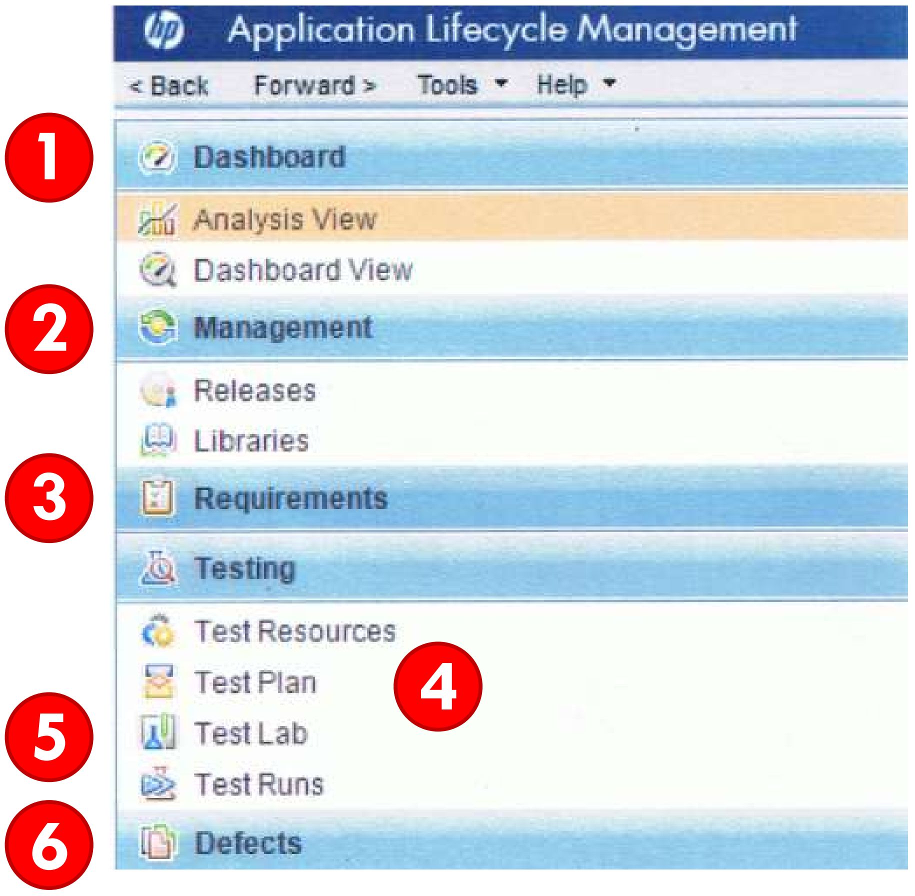
qTest Modules:

1. **Test Plan** – managing releases
2. **Requirements** – maintain requirements as a test basis
3. **Test Design** – create, maintain test case inventory
4. **Test Execution** – create test runs, capture test execution statuses, link defects
5. **Defects** – report new defects, manage existing defects
6. **Reports** – canned management reports

HP ALM – Test Management Tool

- On large projects, test management tools are used to create and manage test assets.
- HP Application Lifecycle Management (ALM) tool is most common on the market and includes six modules:

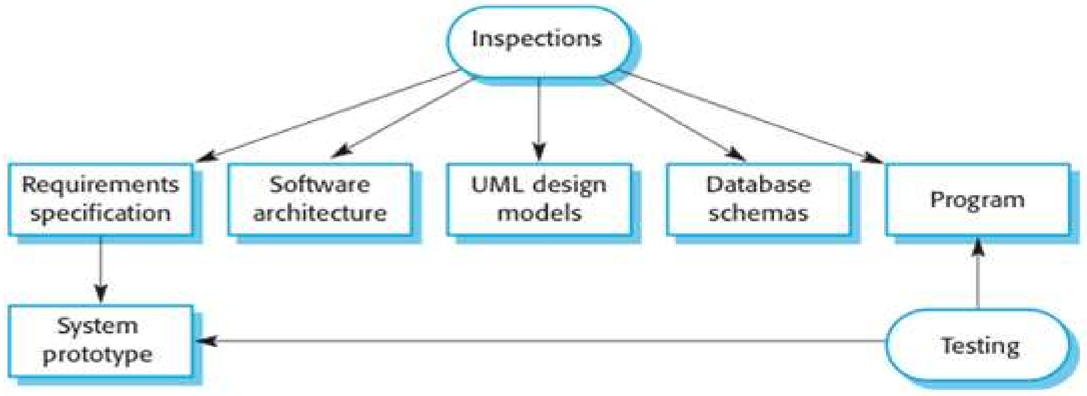
1. **Dashboard** – to produce project metrics
2. **Management** – to define releases and test cycles
3. **Requirements** – to manage the inventory of requirement, scope of testing, traceability
4. **Test Plan** – to manage the inventory of test cases
5. **Test Lab** – to manage the test execution log
6. **Defect** – to report and manage defects



Software Inspections vs. Testing

Inspections and Testing

- Software inspections. Concerned with analysis of the static system representation to discover problems (**static verification**). May be supplemented by tool-based analysis of documents and code. Was popular in the IT industry in the 1990s. Was commonly conducted on Waterfall projects.
- Software testing. Concerned with exercising and observing product behaviour (**dynamic verification**). The system is executed with test data and its operational behaviour is observed.



Formal Inspection Process

Inspection in software engineering refers to peer review of any work product by trained individuals who look for defects using a well-defined process.

The stages in the inspections process are:

- **Planning:** The inspection is planned by the moderator.
- **Overview meeting:** The author describes the background of the work product.
- **Preparation:** Each inspector examines the work product to identify possible defects.
- **Inspection meeting:** During this meeting the reader reads through the work product, part by part and the inspectors point out the defects for every part.
- **Rework:** The author makes changes to the work product according to the action plans from the inspection meeting.
- **Follow-up:** The changes by the author are checked to make sure everything is correct.

The Inspection Process Roles

A formal inspection process includes the following roles:

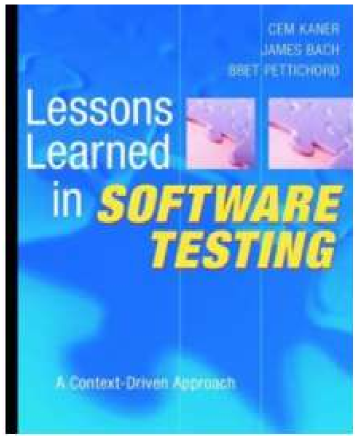
- **Author:** The person who created the work product being inspected.
- **Moderator:** This is the leader of the inspection. The moderator plans the inspection and coordinates it.
- **Reader:** The person reading through the documents, one item at a time. The other inspectors then point out defects.
- **Recorder/Scribe:** The person that documents the defects that are found during the inspection.
- **Inspector:** The person that examines the work product to identify possible defects.

Context-Driven School of Testing

- The context-driven school of software testing advocates continuous and creative evaluation of testing opportunities in light of the potential information revealed and the value of that information to the organization right now.
- The context-driven school of software testing advocates testing in a way that conforms to the context of the project, as opposed to testing in a way that follows some fixed notion of "best practice".
- It declares that the human part of that context is most important, and that good testing is ultimately a matter of skill, not procedure.
- Context-driven software testing is a set of values about test methodology. It is not itself a test technique.
- Context-driven testing could arguably be called agile testing because the principles it recommends (see next slide) are analogous to those defined in the Agile Manifesto.

Principles of the Context-Driven Testing

1. The value of any practice depends on its context.
2. There are good practices in context, but there are no best practices.
3. People, working together, are the most important part of any project's context.
4. Projects unfold over time in ways that are often not predictable.
5. The product is a solution. If the problem is not solved, the product does not work.
6. Good software testing is a challenging intellectual process.
7. Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products.


CEM KANER
JAMES BACH
BRET PETTICHORD
Lessons Learned in SOFTWARE TESTING
A Context-Driven Approach

Reference - http://context-driven-testing.com/

Exploratory Testing Explained

- Exploratory testing has always been performed by skilled testers. In the early 1990s, ad hoc testing was too often synonymous with sloppy and careless work.
- As a result, a group of test methodologists (now calling themselves the Context-Driven School) began using the term "exploratory" seeking to emphasize the dominant thought process involved in unscripted testing, and to begin to develop the practice into a teachable discipline.
- This new terminology was first published by Cem Kaner in his book *Testing Computer Software* and expanded upon in another book *Lessons Learned in Software Testing*.
- Exploratory testing can be as disciplined as any other intellectual activity.

C. Kaner’s exploratory testing tutorial is available on the course portal (Classes), in the Supplementary Materials module.

Exploratory Testing Pros and Cons

Benefits and Drawbacks

- The advantages of exploratory testing are that less preparation is needed, important bugs are found quickly, and at execution time, the approach tends to be more intellectually stimulating than execution of scripted tests.
- The disadvantages are that a) tests invented and performed on the fly can't be reviewed in advance, hence, they can't prevent errors in test cases, and b) it can be difficult to show exactly which tests have been executed and measure, report the test execution progress.

Exploratory testing can effectively complement the scripted (requirements-based) testing, especially, when we know that requirements are not complete.

Test Metrics

- Metrics in software development are important means to provide stakeholders with visibility into various aspects of project progress.
- Testing metrics are commonly used to provide visibility to the project’s stakeholders into test design and test execution progress and software product stability (defect metrics).
- On large projects, collecting data and producing metrics requires using test management tools.
- A standard set of metrics provided by a Test Manager and communicated to the project’s stakeholders on a regular basis is known as a *Test Management Dashboard*.

Types of Test Metrics

Test Design Metrics

- Test coverage, i.e., the ratio of requirements covered by tests to the total number of requirements in the scope of testing. This metric is useful to provide visibility into test design progress and completion.
- Distribution of tests by application modules, or functional areas. This metric is useful to plan resources and schedule across various modules.

Test Execution Metrics

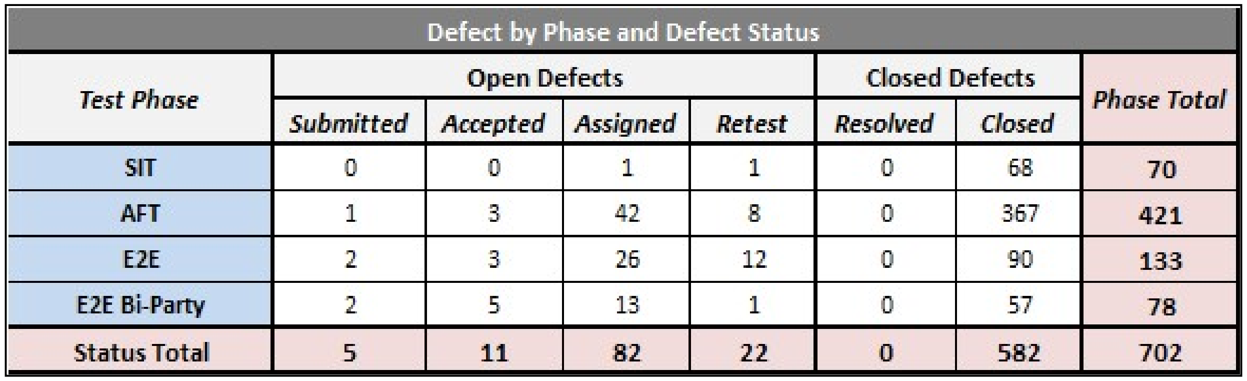
Most common and important test execution metrics.

- Distribution of tests by the execution status – *Pass, Fail, No Run*.
- Test execution progress – a chart showing the trend of the number of executed tests.
- Defect Metrics – show a trend of open defects, defect distribution by priority, functional areas, resolution teams, etc.

Examples of Defect Metrics

Producing and communicating defect metrics can help a test manager and project stakeholders assess how effective the testing is and when it can be completed.

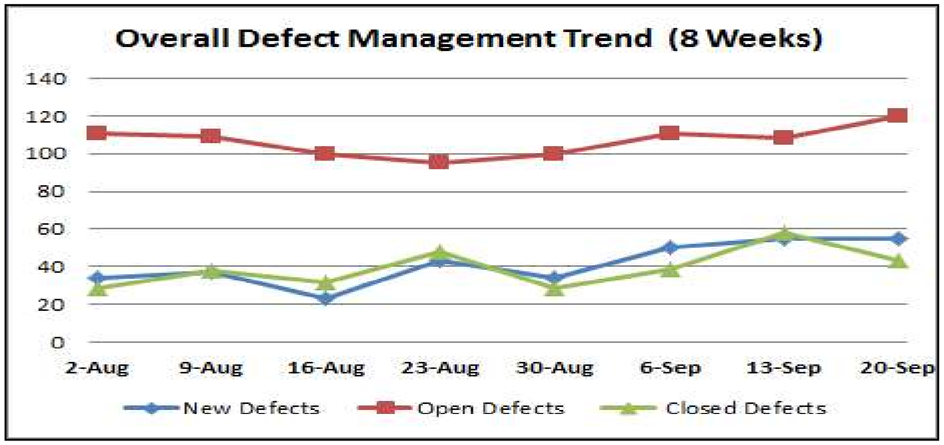
Distribution of defects by test type and status.



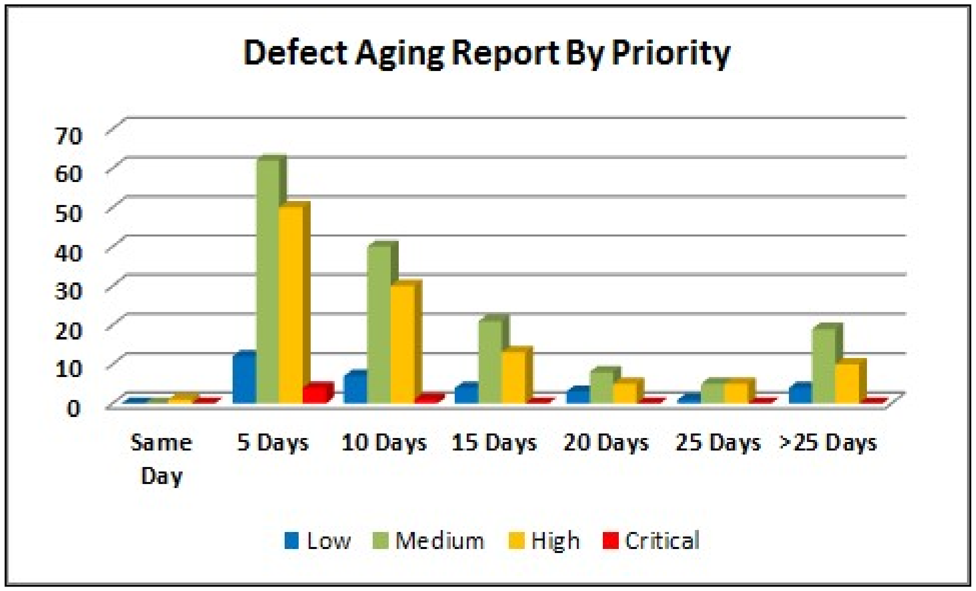
Defect by Phase and Defect Status

Test Phase	Open Defects				Closed Defects		*Phase Total*
	Submitted	*Accepted*	*Assigned*	*Retest*	*Resolved*	*Closed*	
SIT	0	0	1	1	0	68	70
AFT	1	3	42	8	0	367	421
E2E	2	3	26	12	0	90	133
E2E Bi-Party	2	5	13	1	0	57	78
Status Total	5	11	82	22	0	582	702

Trend of defects by status.

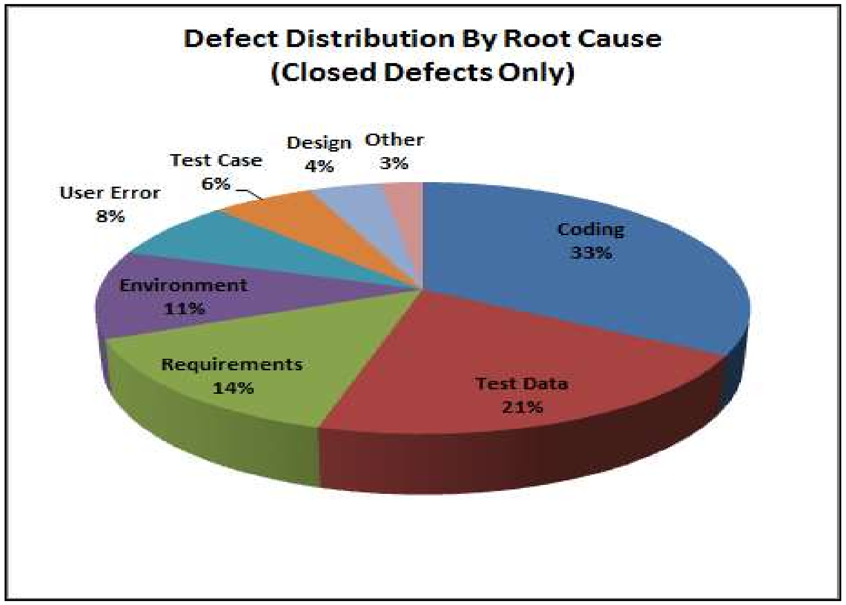


Examples of Defect Metrics (continued)



The **defect aging** report can help manage commitments to investigate and fix defects.

The **defect root-cause** report can help understand the main reasons why defects were introduced.



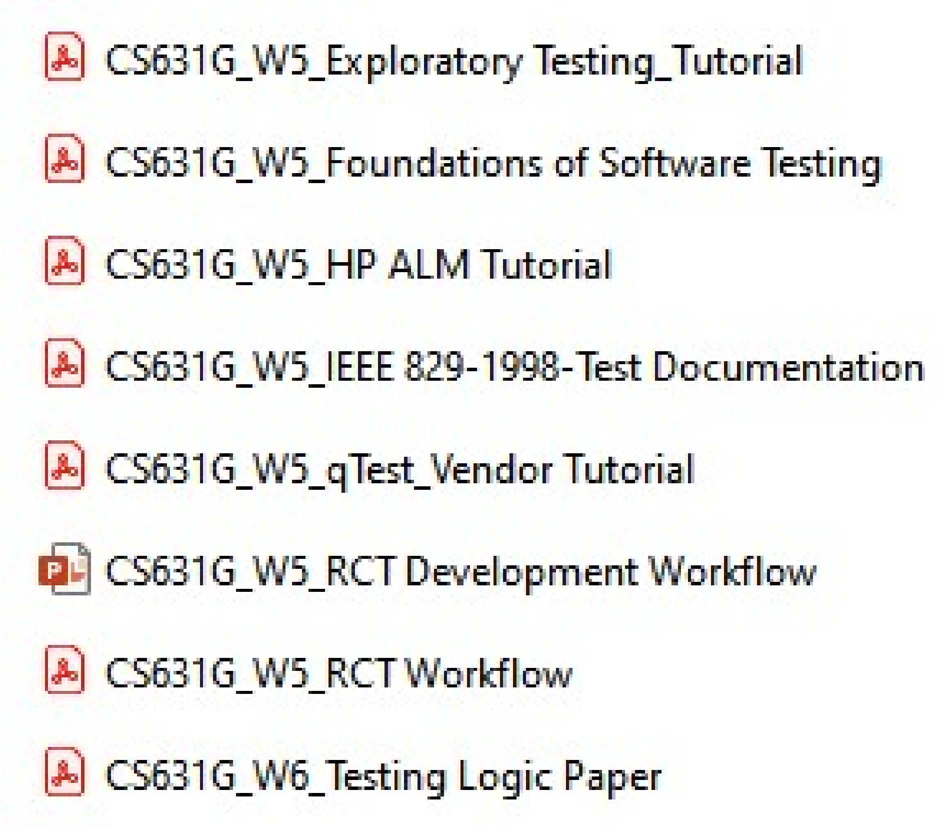
Key Points

- When testing software, you should try to ‘break’ the software by using experience and guidelines to choose types of test cases that have been effective in discovering defects in other systems.
- Wherever possible, you should develop automated tests for regression testing. The tests are embedded in a program that can be run every time a change is made to a system.
- Test-first development is an approach to development where tests are written before the code to be tested.
- Acceptance testing is a user testing process where the aim is to decide if the software is good enough to be deployed and used in its operational environment.

Exercises

1. Explain what software testing is.
2. What are the main schools of testing?
3. Explain what exploratory testing is.
4. Describe the two distinct missions of testing.
5. Describe the conventional test process lifecycle.
6. Explain what regression testing is.
7. Explain in which context you would consider the E2E testing.
8. Explain what you need to include in a test plan document.
9. What are the specifics of use-case-driven testing?
10. What is the difference between the white-box and black-box testing approaches?
11. Name conventional black-box test design techniques.
12. Name conventional white-box test design techniques.
13. What is the purpose of a test summary (completion) report?
14. What is the difference between software testing and software inspections?
15. Give examples of test metrics that can be used on large software projects.

Appendix: Supplementary Materials



CS631G_W5_Exploratory Testing_Tutorial

CS631G_W5_Foundations of Software Testing

CS631G_W5_HP ALM Tutorial

CS631G_W5_IEEE 829-1998-Test Documentation

CS631G_W5_qTest_Vendor Tutorial

CS631G_W5_RCT Development Workflow

CS631G_W5_RCT Workflow

CS631G_W6_Testing Logic Paper