

CS631G Software Verification

MASTERING TEST DESIGN (W6)

Class instructor: Yuri Chernak, PhD

Outline

2

Section	Outline
Part I. Test Process Overview	<ul style="list-style-type: none">• The conventional test process lifecycle• Conventional test levels (types)• White Box vs. Black Box testing• IEEE Std.829 standard for Test Design Specification
Part II. Model-based Testing with UML and RCT	<ul style="list-style-type: none">• Testing based on Use-Case diagrams• Testing based on Activity diagrams• Testing based on State diagrams• Testing with RCT (entitlements, data flows)
Part III. Black-box Testing Techniques	<ul style="list-style-type: none">• Black-box testing process• Equivalence Class Partitioning• Boundary Value Analysis• Decision Table testing• State-Transition testing
Part IV. Exploratory Testing	<ul style="list-style-type: none">• Exploratory testing explained• Exploratory testing procedure• Exploratory testing Pros and Cons
Part V. HP ALM Test Management Tool	<ul style="list-style-type: none">• HP ALM and qTest overview• Managing inventory of test cases• Managing test execution• Managing defects

Class Objectives

3

The objective of this class is to discuss the test process and test levels, focusing on the conventional test design techniques.

In this lecture we cover UML-based testing, black-box testing, and exploratory testing. The lecture concludes with an overview of the HP ALM test management tool.

At the end of this class you will:

- understand the test process lifecycle and conventional test levels;
- learn UML-based test design techniques;
- learn black-box testing techniques;
- understand the exploratory testing concept, pros and cons;
- learn features of the HP ALM tool.

Part I.

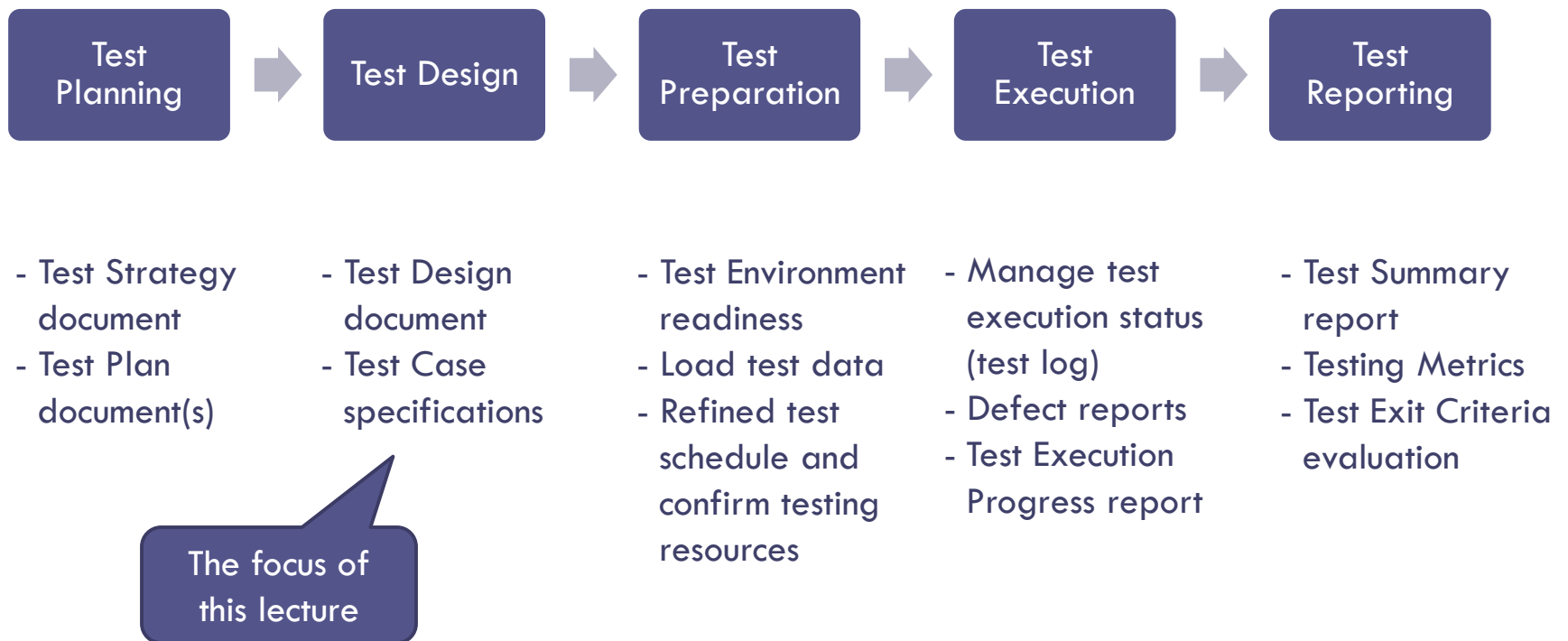
4

Part I. Test Process Overview

The Test Process Lifecycle

5

A conventional test process includes five phases, where each phase has its own activities and deliverables, as shown below.



Two Missions of Software Testing

IEEE Std. 610 Definition - 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 items.

Validation Testing Mission

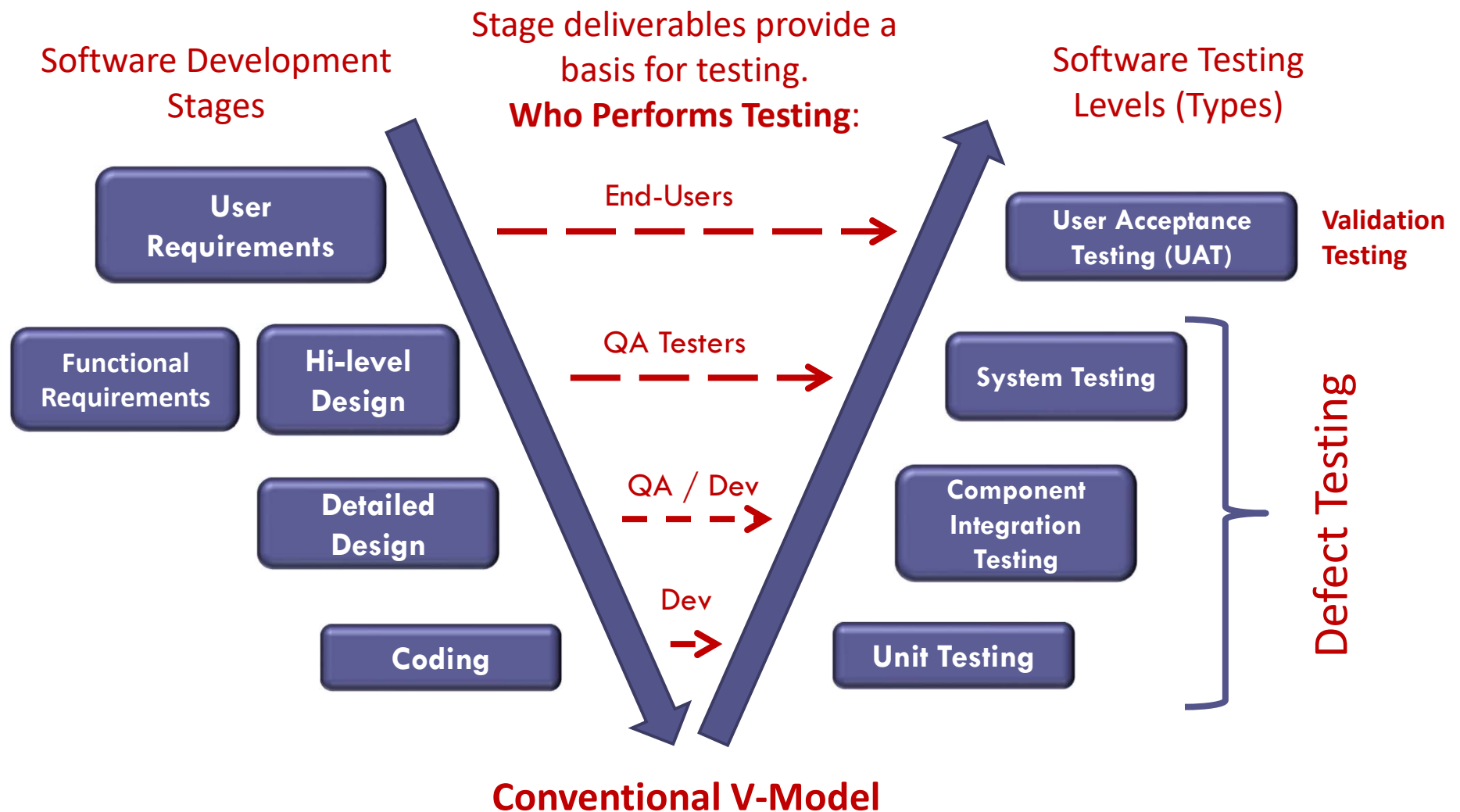
- Mission – evaluate a system's **suitability** to support a customer's business.
- To demonstrate to the developer and the system customer that the software meets its requirements.
- A successful test shows that the system operates as intended.

Defect Testing Mission

- 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.

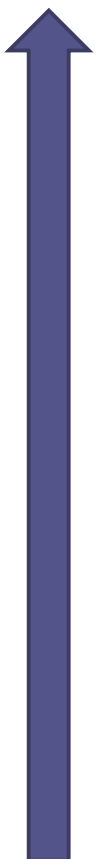
Conventional Test Levels (Types)

7



Test Levels Explained

8



Test Level	Description
Acceptance Testing	<ol style="list-style-type: none">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.
System Testing	Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.
Component Integration Testing	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	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 and fixes have not 'broken' previously working code.
- In a manual testing process, regression testing is expensive, but with the automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program (example – DevOps process).
- Executing a limited sub-set of the regression test suite for a new build is known as "smoke testing".

Performance Testing

- Provides a systematic evaluation of the system's performance characteristics to demonstrate that they meet the operational requirements.
- Performance testing is an engineering discipline and requires special tools and highly-technical skills.

Other Test Types

10

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 or “Systems Integration Testing” (SIT).
- In the financial industry, this is also known as “Front-to-Back” testing.

Operational Acceptance Testing (OAT)

- **OAT** is used to determine the 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 teams and production support groups.

Black-box vs. White-box Testing

11

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.
- Usually used for functional or user acceptance testing.

White-box Testing

- A method of software testing 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.
- Usually used for unit testing.

Black-box vs. White-box Testing Techniques

12

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
Decision table testing	Branch testing
State-transition testing	Statement coverage
	Decision coverage

Discussed in
Part III

IEEE Std. 829 – Test Design Specification

13

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

14

- A formal test design document is mostly used on **requirements-based testing** projects.
- Requirements can be developed at various levels of detail and abstraction. When requirements are structured by large chunks, e.g., use cases or user stories, identifying test cases can be challenging.
- As a solution, we first develop test ideas for a large requirement, capture and elaborate them in a **test design specification** and then we use this document as a basis for detailing test cases.
- **A test design specification can also benefit Agile projects, where we need to develop test ideas, i.e., story acceptance tests for user stories.**
- Sometimes, a good test design document is sufficient to execute testing and more detailed test cases are not developed.

Part II.

15

Part II. Model-based Testing with UML and RCT

Model-based Testing with UML

16

- UML is a standard notation for specifying system functionality and behavior.
- Hence, UML can be used to support model-based testing, i.e., as a basis for developing test cases.
- Model-based testing can be effective on projects that lack requirements specifications, e.g., Agile development projects.
- There are three types of UML diagrams that can support black-box testing:
 - Use-case diagrams
 - Activity diagrams
 - State diagrams
- In addition to the UML diagrams, an RCT provides an effective basis for test ideas (see Slides 22, 23, 24).

Testing Based on Use-Case Diagrams

17

- A **use-case diagram** is a representation of a user's interaction with the system that shows the relationship between the user (actor) and the different use cases in which the user is involved.
- A use-case diagram can identify various Actors and system use cases, and will often be accompanied by other types of diagrams as well.
- The use-case diagram can support the following test objectives:
 - Identify and validate various actors (user roles) who can execute a given use case, a.k.a. **entitlements testing**;
 - If multiple roles can execute a given use case, explore and test concurrency conditions, i.e., **concurrency testing**;
 - If a given use case has relationships with other use cases, e.g., <<include>> and <<extend>>, **test all usage permutations** of the base use case.

HMS Example: Use Case Diagram

18

UC.1.02 - Update Guest Reservation

Test 1 - Entitlements

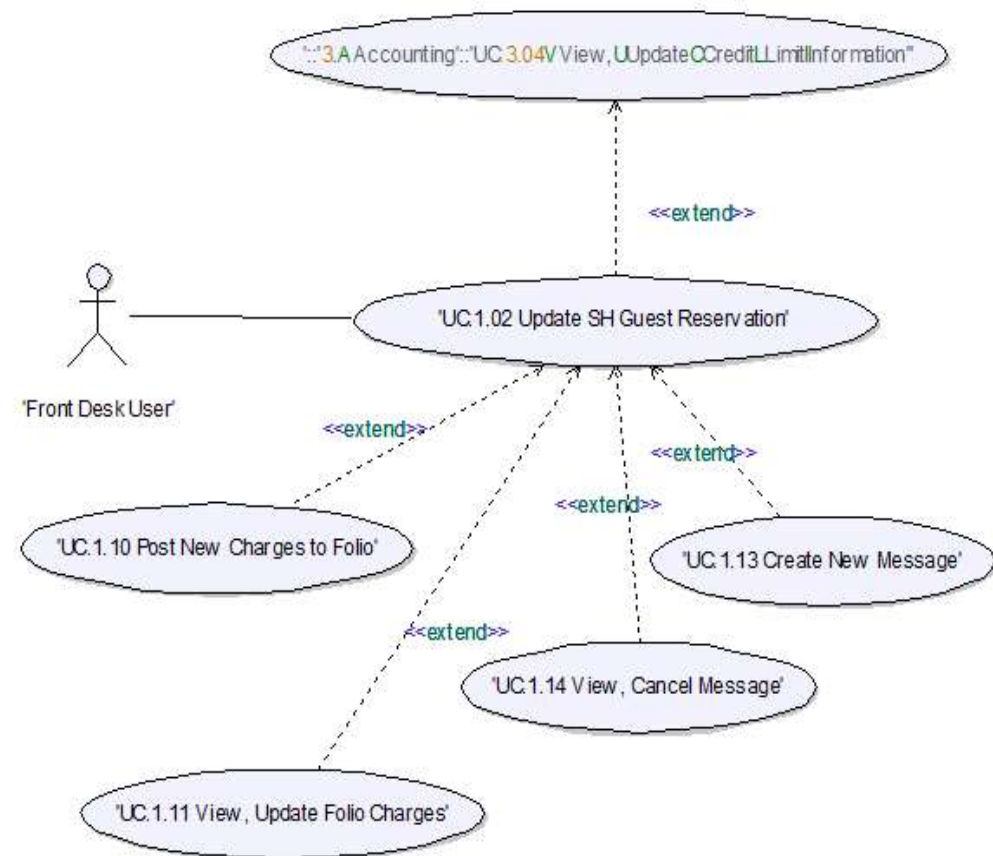
- Validate that Front-Desk Clerk and Property Manager can execute UC.1.02.
- Validate that Accounting User and Housekeeping User cannot execute UC.1.02.

Test 2 – Concurrency

Validate concurrent update of the same reservation by FD Clerk and Prop. Manager

Test 3 – Core Functionality

- Post new folio charges while updating a reservation
- Update folio charges while updating a reservation
- Create new message while updating a reservation
- Cancel a message while updating a reservation



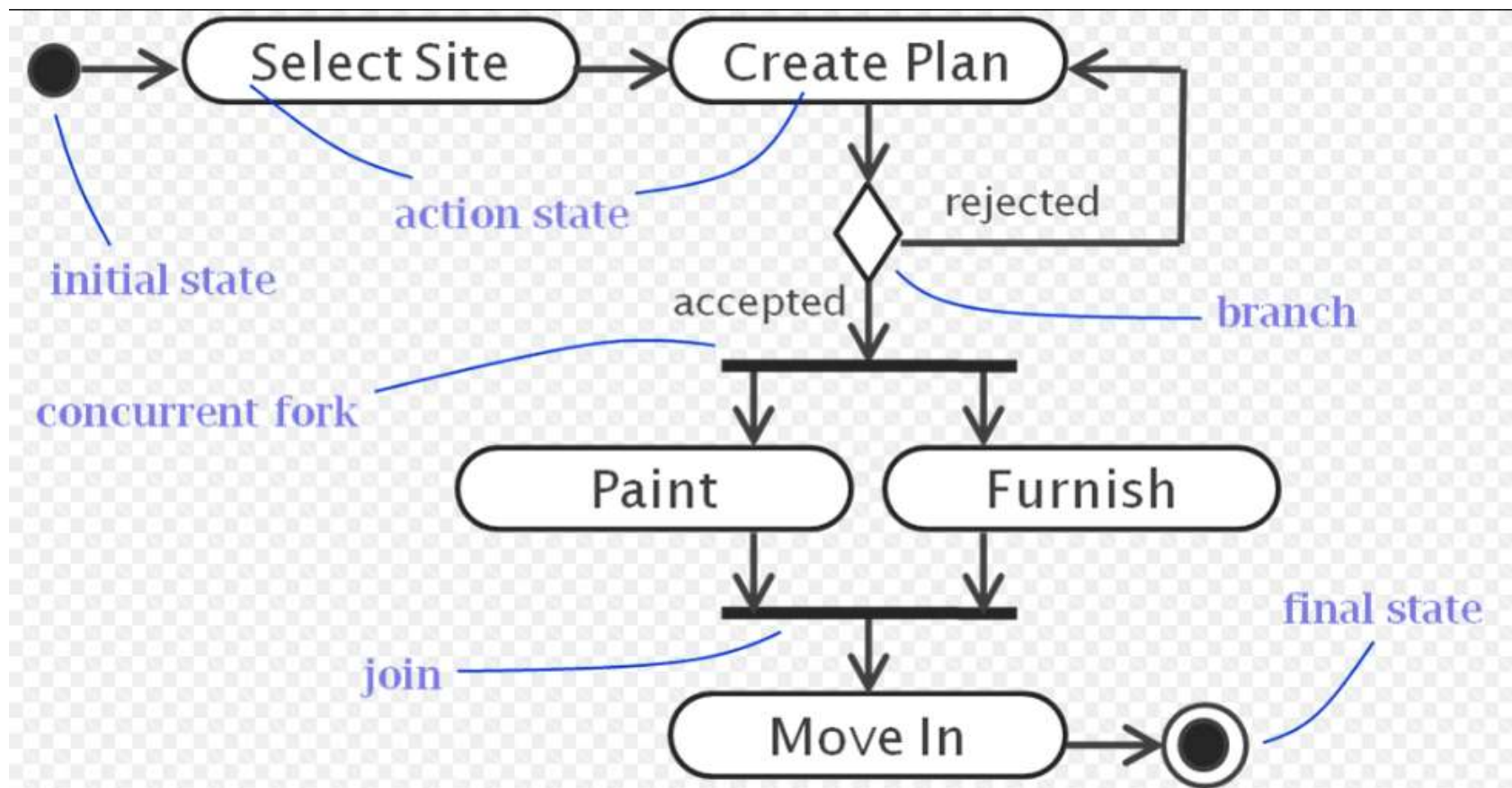
Testing Based on Activity Diagrams

19

- **Activity diagrams** are graphical representations of workflows of stepwise activities and actions with support for choice, iteration, and concurrency.
- In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows).
- Activity diagrams show the overall flow of control, which is similar to source code control flow used as a basis for traditional white-box testing.
- Test ideas that can be derived from analyzing the activity diagram:
 - Test **all activities** in the control flow (*equivalent to the statement coverage in white-box testing*);
 - Test all **activity paths** in the control flow (*equivalent to the control flow testing in white-box testing*);

Activity Diagram Example

20



Testing Based on State Diagrams

21

- **UML state machine diagrams** depict the various states that an object may be in and the transitions between those states. In fact, in other modeling languages, it is common for this type of a diagram to be called a state-transition diagram or simply a state diagram.
- A state represents a stage in the behavior pattern of an object, and like UML activity diagrams it is possible to have initial states and final states.
- An initial state, also called a creation state, is the one that an object is in when it is first created, whereas a final state is the one in which no transitions lead out of.
- A transition is a progression from one state to another and will be triggered by an event that is either internal or external to the object.
- State diagram-based testing is discussed later, see slides 37-39.

Crosscuts: Model-based Testing with RCT

22

- A **Requirements Composition Table (RCT)** represents a holistic view of an application's functionality in the form of a binary relation.
- For each core feature, the RCT provides a list of applicable crosscuts that should be tested in the context of a given core feature.

Applicable crosscuts present a
checklist of test ideas.

01. Front Office Module																
Concern Types	UC.1.01 Create Guest Reservation	UC.1.02 Update Guest Reservation	UC.1.03 Cancel Guest Reservation	UC.1.04 Create Company Account	UC.1.05 Update Company Account	UC.1.06 Close Company Account	UC.1.08 Check-In Guest	UC.1.09 Check-Out Guest	UC.1.10 Post New Charges to Folio	UC.1.11 View, Update Folio Charges	UC.1.12 Quick Posting of Folio Charges	UC.1.13 Create New Message	UC.1.14 View, Cancel Message	UC.1.15 Create Group	UC.1.16 View, Update Group	UC.1.17 Cancel, Close Group
Core Functionality	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
GUI - User Interface	1	1	1	1	0	0	1	1	1	1	1	1	0	1	0	0
Crosscutting Concerns																
ET - Entitlements	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FV - Field Validation	1	1	0	1	1	0	1	1	1	0	1	1	0	1	1	0
DDV - Data-Dependency Validation	1	1	0	0	0	0	1	0	1	0	1	1	1	0	1	1
DDD - Data-Driven Defaults	1	1	0	1	1	0	1	1	0	0	0	0	0	1	1	0
CL - Calculations	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0	0
CC - Concurrency	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
CN - Connectivity	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0
TST - Transaction Status	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1
DF-In - Data Flow In	0	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1
DF-Out - Data Flow Out	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
SI-In - System Interface In	1	1	0	0	0	0	1	0	1	1	1	0	0	0	0	0
SI-Out - System Interface Out	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1

Entitlements: Model-based Testing with RCT

23

- **Entitlements testing** requires a list of all user roles in the system and exercising a core feature with each of these system roles.
- The expected result is validated according to the Entitlements specification (below).

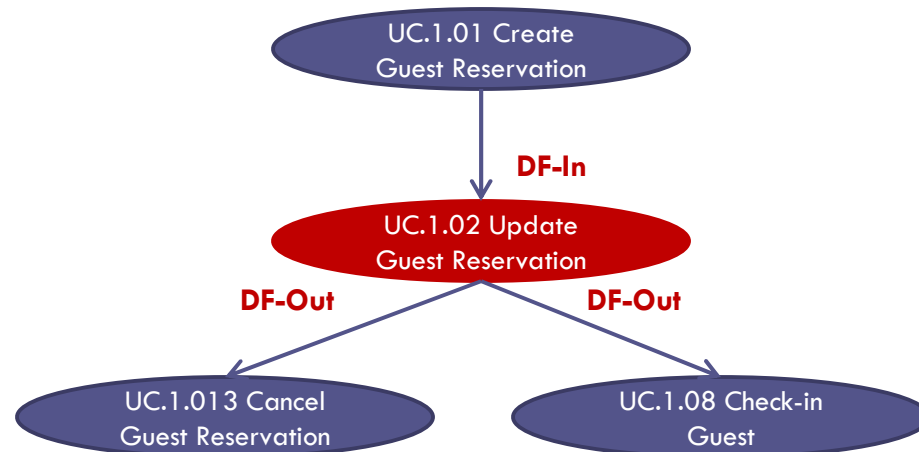
[illegible]

DF-In, DF-Out: Model-based Testing with RCT

24

Concern Types	UC.1.01 Create Guest Reservation	UC.1.02 Update Guest Reservation
Core Functionality	1	1
GUI - User Interface	1	1
Crosscutting Concerns		
ET - Entitlements	1	1
FV - Field Validation	1	1
DDV - Data-Dependency Validation	1	1
DDD - Data-Driven Defaults	1	1
CL - Calculations	1	1
CC - Concurrency	0	1
CN - Connectivity	1	1
TST - Transaction Status	0	1
DF-In - Data Flow In	0	1
DF-Out - Data Flow Out	1	1
SI-In - System Interface In	1	1
SI-Out - System Interface Out	1	1

- **RCT-based data flow testing** requires the exact specification where the data is coming from (DF-In) and where it goes to (DF-Out) from a given core feature.
- Such a specification can be captured in the form of a data flow diagram.
- The up-stream core features can be used to generate test data; the down-stream features can be used to validate expected results.



Part III.

25

Part III. Black-box Testing Techniques

Black-box Testing Overview

26

- **Black-box testing** is a strategy in which testing is commonly based on requirements specifications.
- Unlike the white-box testing, the black-box testing requires no knowledge of the internal structure or implementation of the software under test.

Advantage of Black-box Testing

- Formal black-box testing helps a tester design tests that are both efficient and effective in finding defects.
- Black-box testing helps maximize the return on our testing investment.

Disadvantages of Black-box Testing

- Requirements are always incomplete. Hence, when designing tests solely on requirements, tester can never be sure of how much of the system has been tested.
Black-box testing is only as good as the requirements.
- Exhaustive input testing is impossible; testers use only a subset of input data.

Black-box Testing Process

27

The general black-box testing activities include:

- The requirements specifications are analyzed, and test objectives and purpose are defined.
- **Valid inputs (test data) are chosen**, based on the specification, to determine that the system processes them correctly.
- **Invalid inputs are also chosen** to verify that the system handles properly challenging inputs.
- Expected outputs (test Oracles to match with the actual system responses) for those inputs are determined and confirmed with developers or users.
- Test design completeness is demonstrated using an RTM (requirements traceability matrix), **which shows that all requirements are covered by test cases.**
- The tests are executed and actual results are compared with the expected results.
- A tester records the execution status for each executed test in a test execution log. If a test failed, the tester reports a defect, in which the failed test case is referenced and the issue is described and may be supported with evidence (screen shots, etc.).
- When a bug is fixed, a defect is retested and closed and the status of the related test case is changed to *Passed*.

Equivalence Class Partitioning

28

- Equivalence class testing is most commonly used in testing data entry field edits.
- Equivalence class testing is a technique used to reduce the number of test cases to a manageable level while still maintaining reasonable test coverage.
- The main motivation for using this technique is to avoid redundancy.
- The important aspect of equivalence classes is that they form a partition of a set, where the partition refers to a collection of mutually disjoint subsets whose union is the entire set.
- The fact that the entire set is represented assures completeness, and the disjoint sets assure non-redundancy.
- The key to equivalence class testing is the choice of equivalence partitions that determines the classes.
- Equivalence class testing is accomplished by using one (or few) input in a test case from each equivalence class that represents the entire class.

Equivalence Class Assumptions

29

The equivalence class partitioning technique is based on two assumptions:

1. If one test case in an equivalence class detects a defect, all other test cases (inputs from the same partition) are likely to detect the same defect.
2. If one test case in an equivalence class does not detect a defect and pass testing, no other test cases in the same equivalence class are likely to detect the defect.

Equivalence class testing steps

1. Identify the equivalence classes from related requirements.
2. Create a test case for each equivalence class.

HMS Example: Equivalence Class Testing

30

A screenshot of a hotel reservation form. The fields include: Property (14927), # Rooms (1), Arrival (3/10/2007), Nights (1), Departure (3/11/2007), # Adults (1), # Children (0), Company, Group, Room Type, Rate Plan (RACK), and Rate (\$0.00). The # Adults field is highlighted with a red dashed border.

Requirement “Number of Adults”

- When creating a reservation, the user enters the number of adults, which is a mandatory field.
- The field valid values are in the range 1 – 10. When the value is over 4, the system displays a warning message and asks the user for confirmation the number of adults.

Equivalence Classes (Partitions)



Test Inputs

- Valid: 2, 8 (with a warning message)
- Invalid: no value (blank field), (-1), 11

Boundary Value Analysis

31

- Equivalence class testing is the most basic test design technique that leads us to an idea of boundary value testing.
- **Boundary value analysis** is based on testing at the boundaries between partitions (equivalence classes).
- Boundary value testing is effective because that is where many defects hide.

Boundary value testing steps

1. Identify equivalence classes.
2. Identify the boundaries of each equivalence class; **decide whether the boundary is a valid or invalid condition.**
3. Create test cases for each boundary value by choosing one point on the boundary, one point just below, and one point just above the boundary.

HMS Example: Boundary Value Testing

32

Equivalence Classes (Partitions)



Property: 14927
Rooms: 1
Arrival: 3/10/2007
Nights: 1
Departure: 3/11/2007
Adults: 1
Children: 0 0
Company:
Group:
Room Type:
Rate Plan: RACK
Rate: \$0.00

Identifying Test Cases

1. Equivalence classes – 2 valid and 2 invalid
2. Boundary values:
 - a) 0 – invalid
 - b) 4 – valid
 - c) 10 – valid
3. Selected test cases:
 - a) Boundary 0: invalid inputs 0, -1, valid input 1
 - b) Boundary 4: valid inputs 3, 4 (no warning), 5 (warning)
 - c) Boundary 10: valid inputs 9, 10, invalid input 11

Example: Accounting Software System

33

Vendor Maintenance

Update Delete Copy Clear Close By Vendor #

Vendor # AFF1 Company Aero Furniture Factory, Inc.

Information Contact Activity NotePad Analysis Settings GL Accounts (1) GL Accounts (2)

Alias

Address 6200 Lincoln Avenue Building A

Create Date 01/01/10

Discount % 20.00%

Credit Limit 100,000.00

“Discount %” Field Requirement

Vendors can be given discount in the range from 0% to 20%.

- What are the valid and invalid equivalence classes?
- What are the boundaries for these partitions?

Decision Table Testing

34

- **Decision tables** are an effective technique to capture complex system requirements, e.g., business rules, etc.
- Decision tables document all possible “conditions” (inputs) and all possible “actions” (outputs or system responses). They have been used since the early 1960s to represent and analyze complex logical relationships or business rules.
- To identify test cases with decision tables, we interpret conditions as inputs, and actions as outputs (expected results). The decision rules then represent test cases.
- The decision table technique can be used for testing data dependency of related fields. In this case conditions will represent equivalence class of each data entry field and the actions will represent system responses to the input value combinations.
- Decision tables can have a graphical representation, **known as Cause-Effect graphing**, that can also be used as a basis for test design.

Decision Table: Example 1

35

- Decision tables are a precise yet compact way to model complex business rules and their corresponding actions.
- Each column (rule) is a test case.

Printer troubleshooter

		Rules							
Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

Decision Table: Example 2

36

A computer supplier firm will give discounts if payments are made within 18 days. The discounting policy is defined as follows:

- if the amount of the order for computer supplies is greater than \$1000, give a 4% discount;
- If the amount is between \$500 and \$1000, give a 2% discount;
- if the amount is less than \$500, do not apply any discount.
- all orders made via the Web automatically receive an extra 5% discount. Any special order is exempt from all discounting.

Conditions	Rules						
special order	Y	–	N	N	N	N	N
payment within 18 days	–	N	Y	Y	Y	Y	Y
ordered via the web	–	–	N	N	–	Y	Y
order between £500 and £1000	–	–	Y	N	N	Y	N
order > \$1000	–	–	N	Y	N	N	Y
Actions							
no discount	X	X			X		
2% discount			X			X	
4% discount				X			X
extra 5% discount						X	X

State-Transition Testing

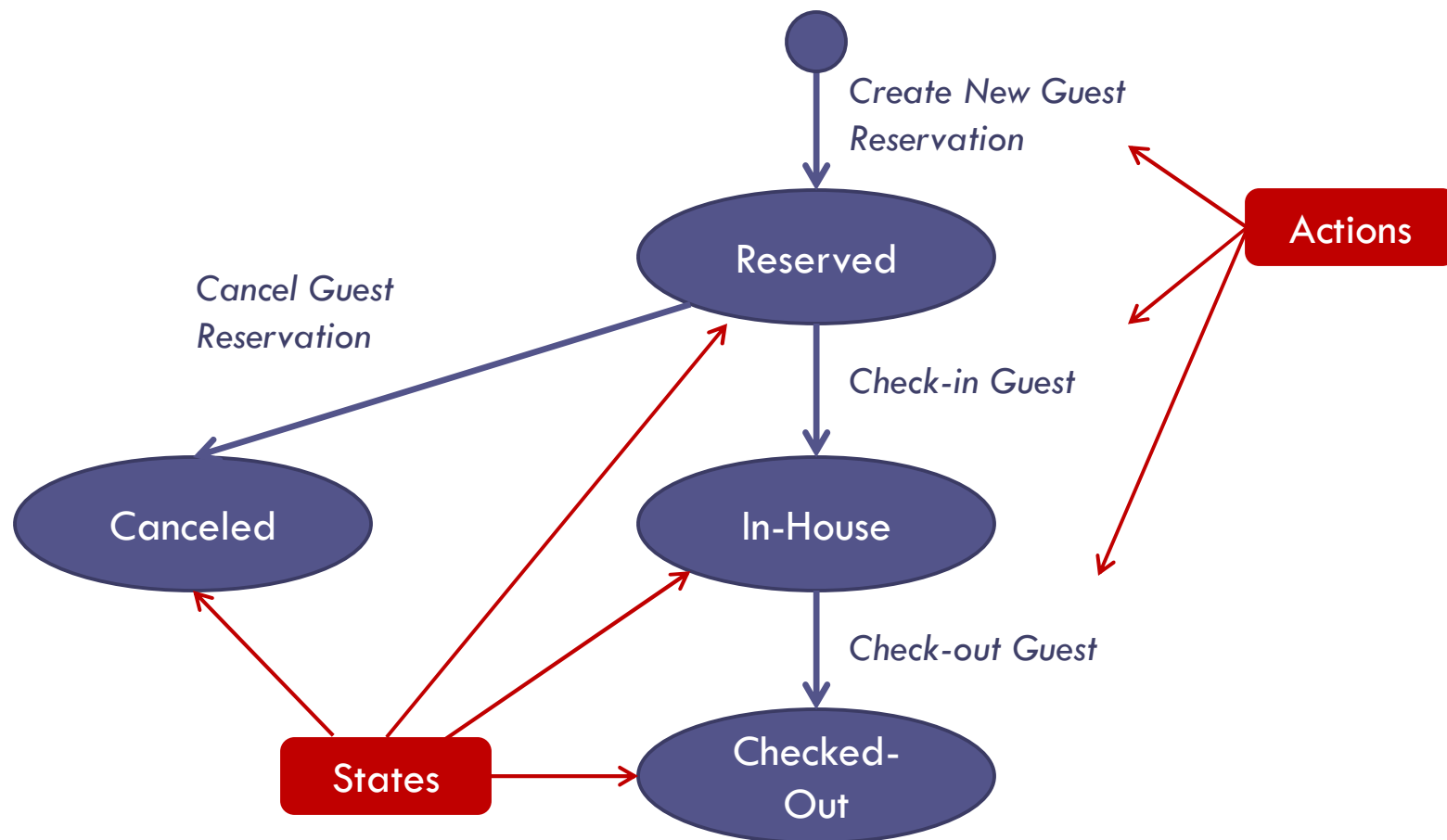
37

- **State-transition diagrams**, like decision tables, are an excellent technique to capture some system requirements. A common example is testing a transaction that changes statuses under certain conditions.
- Business rules defined using a state-transition diagram can also be defined in an equivalent state-transition table. Either a diagram or a table can be used as a basis for designing test cases.
- When designing test cases, four types of coverage can be considered:
 1. **State Coverage**: create a test set that covers all states in the diagram;
 2. **Event Coverage**: create a test set of tests that trigger all events at least once;
 3. **Path Coverage**: create a test set that covers all paths through the transition diagram, if it has branches;
 4. **Transition Coverage**: create a test set that covers all transitions at least once.

HMS Example: State -Transition Diagram

38

A guest's reservation state changes depending on the action performed:



HMS Example: State -Transition Testing

39

Test 1. Validate the user can perform all events (actions)

- Create a new guest reservation
- Check-in guest
- Check-out guest
- Cancel a guest's reservation

Test 2. Validate reservation transitions

- “Reserved” status can be changed to “Canceled” status
- “Reserved” status can be changed to “In-House” status
- “In-House” status can be changed to “Checked-out” status
- “In-House” status cannot be changed to “Canceled” status (*negative TC*)
- “Check-out” status cannot be changed to “Canceled” status (*negative TC*)

Part IV.

40

Part IV. Exploratory Testing

Exploratory Testing Explained

41

- **Exploratory testing** can be a powerful technique when it is performed by skilled testers. However, 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 later was expanded upon in other books, e.g., *Lessons Learned in Software Testing*, etc.
- Exploratory testing can be as disciplined as any other intellectual activity.

Exploratory Testing Explained (continued)

42

Exploratory testing is a style of software testing that:

- emphasizes the personal freedom and responsibility of the individual tester and continually optimizes the value of 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.

Exploratory Testing Procedure

43

An exploratory testing process includes the following steps:

1. Create a mental model of the system's functionality (test basis);
2. Design one or more tests that would disprove, or challenge the model;
3. Execute these tests and observe outcomes;
4. Evaluate the outcomes against the model;
5. Repeat the process until the model is proved or disproved.



Exploratory testing focuses on “defect testing” as opposed to “validation testing”.

Exploratory Testing Pros and Cons

44

Benefits and Drawbacks

- **The advantages** of exploratory testing are that less preparation is needed, important bugs can be 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 or report the test execution progress.

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

Scripted vs. Exploratory Testing

45

- Scripted testing provides a division of labor – planning, test design, and test execution.
- Whereas, exploratory testing is simultaneous planning, test design, and execution.
- Scripted testing requires a test basis in the form of written software requirements. If requirements are incomplete, it undermines the scripted testing effectiveness, completeness, and its ability to find defects.
- With scripted testing we can demonstrate test coverage and better estimate the test execution resources and schedule.
- Estimating the necessary resources and schedule for exploratory testing can be challenging.
- With scripted testing, we can review test specifications and ensure their quality. This may not be possible in the case of exploratory testing.
- Scripted testing allows creating, evolving, and automating a regression test suite, which is not possible with un-documented exploratory testing.
- Exploratory testing is not suitable for regulatory projects that are subject to IT audits.

Part V.

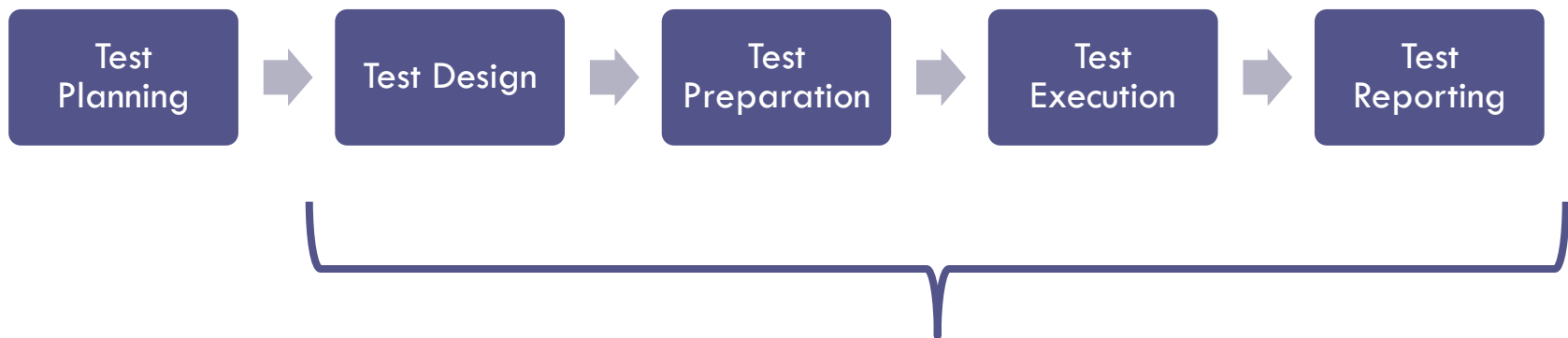
46

Part V. Managing Test Design and Execution with HP ALM Tool

Using a Test Management Tool

47

On large software projects, testers commonly use test management tools to manage their test assets.



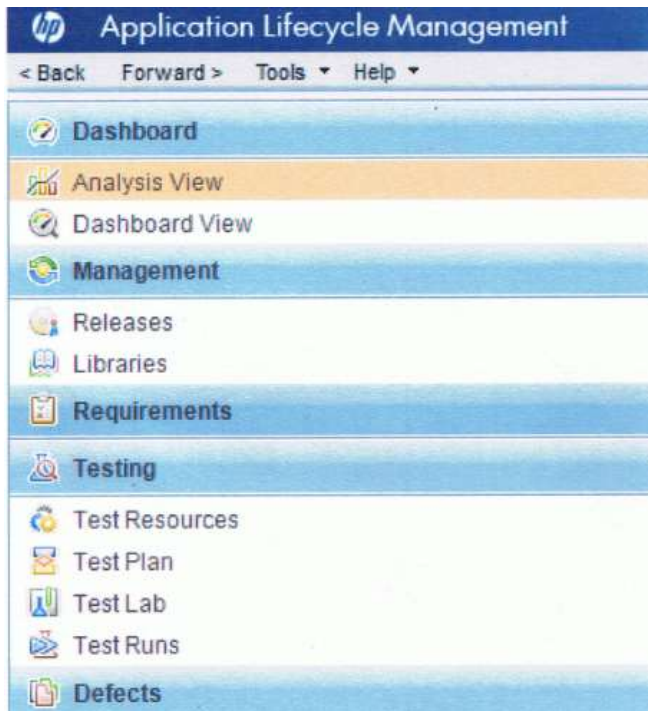
The test process phases where a test management tool can be used.

Test Management Tools – HP ALM and qTest

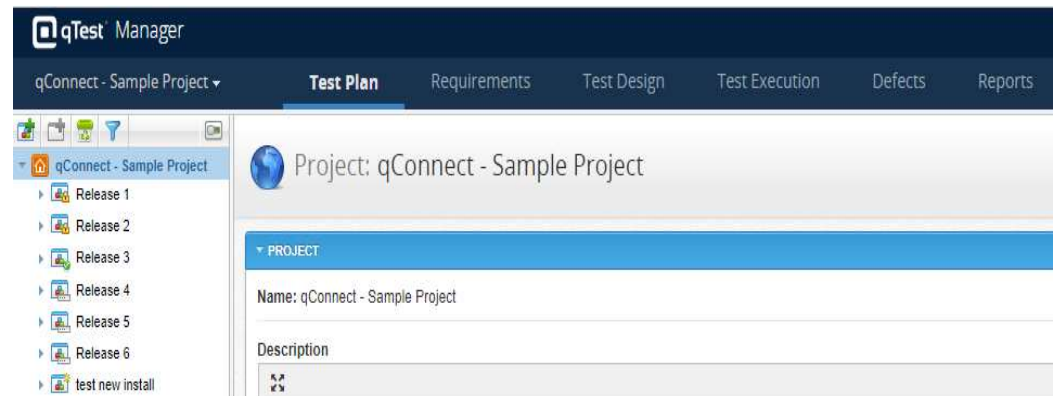
48

HP ALM (Application Lifecycle Management) and qTest tools are the most popular test management tools in the IT industry. **Conceptually, they have very similar functionality.**

HP ALM Home Page



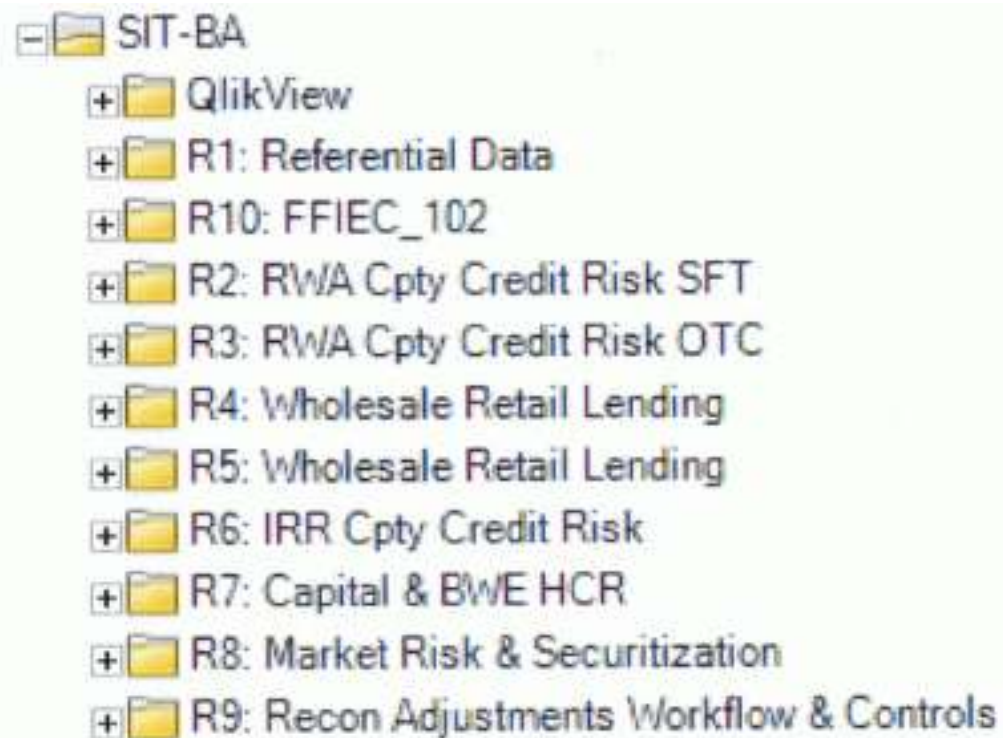
qTest Home Page



HP ALM: Test Repository Structure

49

Test Plan module: Test repository can be structured by test types, e.g., SIT and UAT, and then follow the structure of requirements.



HP ALM: Test Case Specification Screen

50

Each test case specification in ALM includes test execution steps (Design Steps tab) and traceability to related requirements (Req Coverage tab).

Test Case
Name =>

R1 - BUSINESS LINE & PROFIT CENTER - BUSINESS_LINE(PMA) Table Contents

Execution
Steps =>

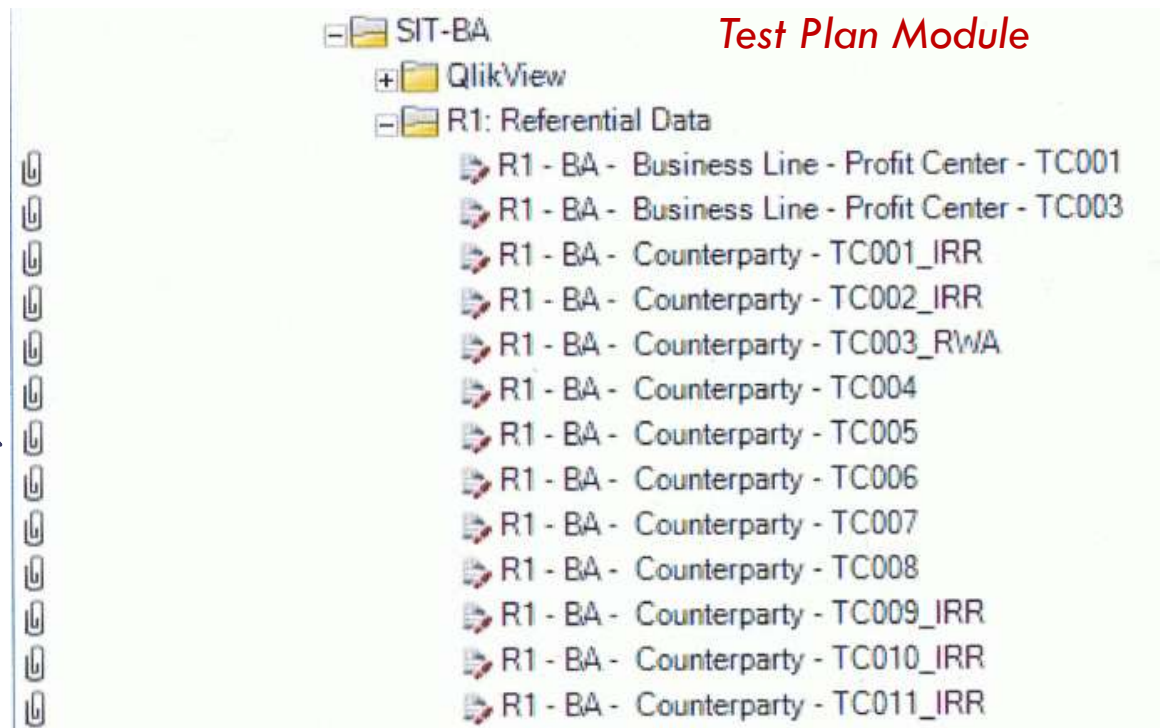
Details	Design Steps	Parameters	Test Configurations	Attachments	Req Coverage	Linked Defects
	Step Name	Description	Expected Result			
	Step 1	Verify that the table BUSINESS_LINE(PMA) is loaded	The data should be loaded. User should be able to verify the contents.			
	Step 2	Validate the Required fields, Non nullable, PK, FK fields of the table BUSINESS_LINE(PMA)	All the required fields (see the description for fields list) should have data			
	Step 3	Validate the default values & authorized values of required fields in the table BUSINESS_LINE(PMA) as applicable	The values should bound to those mentioned in mapping doc.			
	Step 4	Validate the records of the table BUSINESS_LINE(Paradigm)	Records in Moody's should exactly match with those mentioned in FSD. See description for more information			

Managing Large Test Cases in HP ALM

51

- For complex tests, test specifications can be created in Excel documents that are then attached to test cases in ALM.
- In this case, test case items in ALM are used primarily for traceability and status tracking purposes.

Excel
attachments



HP ALM: RTM and Test Coverage Reporting

52

List of
Requirements

List of Related
Test Cases

Req ID	Name	Number of linked tests
177	2.12 Detailed Data Mapping- Table ECONOMIC_SECTOR	2
561	2.11 Detailed data mapping - Table ON_OFF_BALANCE	1
887	Slide Page 25 - RL - Counterparty Risk (1/3)	1
1414	2.13 Detailed Data Mapping - CUST_IND_SEC Table Contents	1
1427	2.12 Detailed data mapping - Table CUST_RISK_TYPE	2
1642	2.12 Detailed Data Mapping - FUTURE Table Contents	1
586	2.11 Detailed data mapping - Table GL_REC_MAPPING_DIM	2
858	Slide Page 19 - RL - Risk Profile Deep Dive - Concentration Risk: Exposure v...	1
562	2.11 Detailed data mapping - Table LOANDEPO	1
1792	2.10 Detailed data mapping - Table BUSINESS_LINE_LINKS_TYPE	2
207	2.12 Detailed data mapping - Table CONTRACT_GUARANTEE	2
1438	2.12 Detailed Data Mapping - CUST_IRR_RTG_XREF Table Contents	1
299	2.12 Detailed data mapping - Table CREDIT_DERIVATIVE	2
193	2.12 Detailed data mapping - COMPANIES Table Contents	1
178	2.12 Detailed Data Mapping- Table ISSUER_CREDIT_RATING	4
859	Slide Page 27 - RL - Appendix A - Concentration Risk: Top 20 Corporate Cou...	1
587	2.11 Detailed data mapping - Table GL_REC_MAPPING	2
831	2.12 Detailed data mapping - Table SECURITY_POSITIONS	2
1793	2.10 Detailed data mapping - BUSINESS_LINE_SET Table Contents	1
1415	2.13 Detailed Data Mapping - CUST_IRR_CLASS_XREF Table Contents	1
179	2.12 Detailed Data Mapping- Table CUST_IRR_ENT_GRP_ATT	2
1643	2.12 Detailed Data Mapping - SECURITIZATION Table Contents	1
1439	2.12 Detailed Data Mapping - Table CUST_XREF_RDD	2

Linked Tests

☐ Show Full Path

Test ID	Test Name
46907	R1 - COUNTERPARTY - ISSUER_CREDIT_RATINGS (OECD)
47828	R1 - COUNTERPARTY - ISSUER_CREDIT_RATINGS (OECD) Check Errors
47829	R1 - COUNTERPARTY - ISSUER_CREDIT_RATINGS (OFSA) Check Errors
1937	R1 - COUNTERPARTY - ISSUER_CREDIT_RATINGS (OFSA)

Requirements
Module

HP ALM: Test Execution Log

53

- The test execution log is a very important deliverable.
- It is use for capturing the execution statuses of test cases and for management reporting.

Test Lab Module

The screenshot displays the HP ALM Test Execution Log interface. On the left, a tree view shows the hierarchy: RISK > SET > SIT-BA. Under SIT-BA, several test cases are listed, with 'SIT_Cycle1_R1_Companies and Dealbooks' selected. On the right, a table titled 'Test: Test Name' shows the execution status of these test cases. The table has two columns: 'Test: Test Name' and 'Status'. The status values are: Not Completed, Failed, Passed, Failed, Passed, Passed, Passed, Failed, Failed, Failed, Passed, Passed, Passed, Failed, Passed, and Failed.

Test: Test Name	Status
R1 - BA - Companies & Deal books - TC001 - Companies	Not Completed
R1 - BA - Companies & Deal books - TC002 - Companies	Failed
R1 - BA - Companies & Deal books - TC003 - Companies	Passed
R1 - BA - Companies & Deal books - TC004 - Companies	Failed
R1 - BA - Companies & Deal books - TC006 - Companies_Links	Passed
R1 - BA - Companies & Deal books - TC007 - Companies_Links	Passed
R1 - BA - Companies & Deal books - TC008 - Companies_Links	Passed
R1 - BA - Companies & Deal books - TC010 - Consolidation_Perimeter	Failed
R1 - BA - Companies & Deal books - TC011 - Consolidation_Perimeter	Failed
R1 - BA - Companies & Deal books - TC012 - Consolidation_Perimeter	Failed
R1 - BA - Companies & Deal books - TC014 - Dealbooks	Passed
R1 - BA - Companies & Deal books - TC015 - Dealbooks	Passed
R1 - BA - Companies & Deal books - TC016 - Dealbooks	Passed
R1 - BA - Companies & Deal books - TC017 - Company_Perimeter	Failed
R1 - BA - Companies & Deal books - TC018 - Company_Perimeter	Passed
R1 - BA - Companies & Deal books - TC019 - Company_Perimeter	Failed

HP ALM: Defect Management

54

When reporting defects, we can link them to related test cases and attach supporting evidence of issues being reported.

Defects Module

	Defect ID	Status	Summary	Description	Defect Type	Sub Type	Detected in Release	Detected in Cycle
		Not Closed And...						
	3587	Assigned	R1 SIT BA Legal Entity - Managed View entities missin...	Following managed view entities are availa...	Issue	Data	SIT	R1
	3585	Assigned	R1-SIT-BA - Legal Entity Multiple Regent codes for one...	1. Regent codes 30475 (BNP PARIBAS HO...	Issue	Data	SIT	R1
	3584	Blocked	R2 - Mapping Logic update for Securities Lending and...	The data mapping for Securities Lending an...	Issue	Requirement	SIT	R2
	3577	Assigned	R3 - ET Commodities - 108 BNPPAR trades are missin...	Title: R3 - ET Commodities - 108 BNPPAR...	Issue	Data	SIT	R3
	3545	Assigned	R4 - CUST_IRR_REFMAP_DET ATT1_VALUE is...	ATT1_VALUE is Non Nullable in FSD. NUL...	Issue	Requirement	SIT	R4
	3541	Assigned	R4 - REFMAP_HDR Data Issues	R4 - REFMAP_HDR Data Issues Please fin...	Defect	Moody's	SIT	R4
	3526	Assigned	R8 - LCR Misclassifications causing wrong RV in Mood...	The following counterparties are being miscl...	Defect	Informatica	SIT	R8
	3525	Blocked	R7 - Capital - SIT BA - GL_GAAP table	Issue: GL_GAAP is populated with all popul...	Defect	Moody's	SIT	R7
	3520	Blocked	R7 CIB - SIT BA - Transformation logic for Average Tot...	Title: Incorrect transformation applied for Av...	Defect	Moody's	SIT	R7
	3510	Assigned	R8 - Interest only MBS receiving improper RV/	Interest only MBS are receiving risk weights...	Defect	Moody's	SIT	R8
	3485	Assigned	R8 - SECURITIZATION - Cleanup Effort - Attribute K_G...	Issue found as part of the Check Errors Cle...	Issue	Data	SIT	R8
	3484	Assigned	R8 - SECURITIZATION - Cleanup Effort - Attribute UN...	Issue found as part of the Check Errors Cle...	Issue	Requirement	SIT	R8

Key Points

55

- A conventional test process lifecycle includes five phases - Test Planning, Test Design, Test Preparation, Test Execution, and Test Reporting.
- There are two fundamental approaches to software testing – black-box and white-box testing:
 - black-box testing examines the functionality of an application without the knowledge of its internal structure or workings.
 - white-box testing tests internal structures or workings of an application, as opposed to its functionality.
- UML is a standard notation for specifying system functionality and behavior and it can be used to support model-based testing.
- Black-box testing helps a tester design tests that are both efficient and effective in finding defects. It includes conventional techniques like Equivalence Class Partitioning, Boundary Value Analysis, Decision Table Testing, and State-Transition Testing.

Key Points (continued)

56

- Exploratory testing is a style of software testing where test planning, test design, and test execution are mutually supportive activities that run in parallel throughout the project.
- Exploratory testing can effectively complement the scripted (requirements-based) testing, especially, when we know that requirements are not complete.
- On large software projects, testers commonly use test management tools to manage their test assets.
- HP ALM is the most-commonly used test management tool in the IT industry.