Practical Software Testing: Techniques for User Acceptance and System Integration

There are many training courses offering a solid grounding in the theory and practice of software testing but software testers attending these courses often struggle to apply what they have learnt when they return to the workplace. This gap between theory and practice is known as the “testing gap”.

This two-day course is designed to bridge the “testing gap”. It is aimed at software testers who need a practical approach to User Acceptance Testing (UAT) and System Integration Testing (SIT) that can be applied to software projects in the real world.

The course is presented by an experienced software tester who shares a wealth of practical tips and recent project experience with participants during the course.

Course duration
2 days

Course Agenda
- The problem with software testing
- A framework for better understanding software testing
- Planning User Acceptance Testing (UAT) and System Integration Testing (SIT)
- Feature testing
- End-to-end testing
- Date-based testing
- Exploratory testing
- Managing the UAT and SIT Effort

Who should attend?
- Test Managers, Test Engineers, Testers, Quality Assurance Staff
- User Representatives, Project managers, Program Managers
- Software Engineers, Developers, Requirements Engineers, Requirements Analysts, Human Factors Specialists
- Process Engineers, Software Engineering Process Group (SEPG) Staff, Methodologists, Process Improvement Staff

Course Topics

Problems With Software Testing

Overview
There are a number of problems that prevent a clear understanding of software testing. These problems include an often-confusing terminology, a number of popular myths
and incorrect beliefs and a gap between what is considered to be best practice and the “actual practice” adopted by many organisations.

**Learning Outcomes**

- Explain how vaguely defined terms, inconsistencies and ambiguities in the terminology of software testing leads to confusion (K2)
- Explain how a number of popular myths and incorrect beliefs about software testing leads to poor planning and execution of test activities (K2)
- Recall that the “testing gap” refers to the large gap between software testing best practice and the actual practice adopted by most organisations (K1)
- Recall the key differences between product and project life cycles and how they affect software testing (K1)
- Recall Garvin’s five views of quality and how they can be summarised in the quality triangle (K1)
- Recall the five objectives of software testing (K1)

**Concepts**

- Confusing terminology
- Software testing popular myths and incorrect beliefs
- The testing “gap”
- Product vs. project life cycles
- Views of software quality
  - Measure of “excellence”
  - Fit for intended purpose
  - Conform to specification
  - Absence of defects and other quality goals
  - Provides value
- Summarising views of quality in the quality triangle
- Aligning software testing objectives with views of quality
  - Validating that software is fit for its intended purpose
  - Verifying that software conforms to its specification
  - Identifying defects
  - Measuring product attributes
  - Building confidence

**Exercises and Practice**

- A short quiz based on the concepts learnt in this section

**A Framework for Better Understanding Software Testing**

**Overview**

Many software testing approaches are based on a rather simplistic view that separates testing into either “black-box” or “white-box” testing performed during a number of sequential “testing phases” (unit, integration, system, acceptance) at the end of a project.

In reality, a good understanding of software testing requires a multi-dimensional framework that more precisely defines the “what”, “how”, “why” and “who” of testing.
Learning Outcomes

- Recall the different levels of testing, emphasising the “recursive” nature of test levels (K1)
- Recall the different categories of test item and explain why it is vital to identify test items in addition to test levels (K1)
- Describe the difference between a “test item” and a “test basis” (K2)
- Recall the test case design techniques discussed during the course (K1)
- Explain the difference between a “test to pass” and “test to fail” approach to test case design (K2)
- Describe the contents and role of a Test Specification (K2)
- Recall the different people that may execute test cases (K1)

Concepts

- The need for a multidimensional view of software testing
- Levels of software testing
  - Unit and component testing level
  - Integration testing level
    - Understanding multiple levels of integration
    - Vertical vs. horizontal integration
  - System testing level
- The “recursive” nature of test levels
- Identifying test items and their relationship to test levels
- Selecting a test basis
  - Requirements as a test basis
  - Program code as a test basis
  - Variations on requirements and code
  - Models as a test basis
  - Experience as a test basis
- Designing test cases
  - What is a test case?
  - Test to pass or “positive” test cases
  - Test to fail or “negative” test cases
  - Overview of test case design techniques
- Who executes the tests?
- Automating test execution

Exercises and Practice

- Apply the software testing framework to describing the UAT and SIT strategies that are currently in place at the participant’s organisation

Planning User Acceptance Testing (UAT) and System Integration Testing (SIT)

Overview

With the addition of a risk-based view of testing, the multi-dimensional software testing framework becomes an ideal tool for planning UAT and SIT strategies.
Learning Outcomes

- Apply the multidimensional testing framework to planning and customising UAT and SIT strategies (K3)
- Explain how risk can be incorporated into a UAT and SIT strategies (K2)
- Develop a UAT test plan based on a customised test strategy (K3)

Concepts

- Applying the software testing framework
  - The standard User Acceptance Testing (UAT) strategy
  - The standard System Integration Testing (SIT) strategy
  - Customising UAT and SIT strategies
- Risk-based testing
  - Product risks
  - Project risks
- Developing UAT and SIT Test Plans
  - Identifying test activities
  - Estimating test effort
  - Assigning resources
  - Developing a test schedule
  - Test environment and tools

Exercises and Practice

- Identify the key elements of a customised UAT Test Plan based on the details of a case study

Feature Testing

Overview

In real projects, requirements are often scattered throughout specifications, change requests and other documents, making the theoretical approach of “verifying” the coverage of requirements during SIT difficult to apply in practice.

A more workable approach is to invest some effort in building a simple model of the features that will be tested and use this to drive the testing effort rather than the specifications.

Learning Outcomes

- Recall a definition for software features (K1)
- Explain that the goals of feature testing and when it is performed (K2)
- Describe the relationship between feature testing and unit testing and its relationship to vertical integration (K2)
- Develop a model that summarises software features using an approach that is appropriate for feature testing (K3)
- Describe some of the test case design techniques that are suitable for designing feature test cases (K2)
- Recall the ways in which tools and automation can be used during feature testing (K1)
- Recall the main issues to take into account when planning and managing feature testing (K1)
Concepts

- What are features?
- Features vs. components
- What is feature testing?
  - Testing features in isolation
  - Testing vertical integration
- When is feature testing performed?
  - The role of feature testing during SIT
  - The role of feature testing during UAT
- Modelling features
  - Why model features?
  - How to model features
- Developing a feature test specification
- Feature testing tools and test automation
- Planning and managing feature testing

Exercises and Practice

- Develop a consolidated model of software features based on the details of a case study

End-To-End Testing

Overview
Testing features in isolation won’t confirm that the software will support business scenarios involving many features; or that complex business rules have been correctly implemented across multiple features.

End-to-end testing uses individual features as building blocks that are assembled into business scenarios. The scenarios are generated from life cycle models of core business objects.

Learning Outcomes

- Explain that the goals of end-to-end testing is and when it is performed (K2)
- Describe the relationship between end-to-end testing and horizontal integration (K2)
- Explain the difference between internal and external events and how both can trigger a change of state within a system (K2)
- Identify core business objects and develop state models of their life cycles (K3)
- Generate a set of business scenarios based on a business object life cycle model (K3)
- Describe some of the test case design techniques that are suitable for designing end-to-end test cases (K2)
- Recall the ways in which tools and automation can be used during end-to-end testing (K1)
- Recall the main issues to take into account when planning and managing end-to-end testing (K1)
Concepts

- What is end-to-end testing?
- Business object life cycles and scenarios
- What is end-to-end testing?
  - Testing features in a specific sequence
  - Testing horizontal integration
- When is end-to-end testing performed?
  - The role of end-to-end testing during SIT
  - The role of end-to-end testing during UAT
- Modelling business object life cycles
  - Business object states
  - External and internal events that trigger changes of state
  - Grouping states
  - Conditions and actions
  - Modelling decisions
- Generating test scenarios
  - Basing test scenarios on the life cycle of a business object
  - Prioritising test scenarios based on risk
- Developing an end-to-end test specification
- End-to-end testing tools and test automation
- Planning and managing end-to-end testing

Exercises and Practice

- Develop a state model that describes the life cycle of some core business objects based on the details of a case study
- Use the life cycle model to generate a set of business scenarios suitable for end-to-end testing

Date-Based Testing

Overview

There are several obstacles that make it difficult to perform comprehensive date-based testing. However, with some careful planning, date-based testing can be incorporated into feature and end-to-end testing with the result that business rules involving dates are more thoroughly tested.

Learning Outcomes

- Explain that the goals of date-based testing is and when it is commonly performed (K2)
- Describe the relationship between date-based testing, feature testing and end-to-end testing (K2)
- Recall the different categories of date-based events (K1)
- Explain the importance of using a fixed reference dates for date-based testing (K2)
- Identify date-based events and incorporate them into life cycle models (K3)
- Recall the different categories of system clock and their relationship to date-based testing (K1)
- Describe the different techniques that can be used to change a system clock during testing (K2)
• Explain the role of date-based testing during feature and end-to-end testing (K2)
• Describe some of the test case design techniques that are suitable for designing date-based test cases (K2)
• Develop a calendar based of date-based events (K3)
• Explain how tools and automation can be used during date-based testing (K2)
• Recall the main issues to take into account when planning and managing date-based testing (K1)
• Plan a schedule of date changes (K3)

Concepts
• What is date-based testing?
• When is date-based testing performed?
  o Date-based testing and feature testing
  o Date-based testing and end-to-end testing
• Date-based events
  o Relative and absolute dates
  o Anniversaries and time periods
  o Adding date-based events to the life cycle models
• System clocks
  o Real time clocks
  o Proxy clocks
• Date-Based testing combined with feature and end-to-end testing
• Developing a date-based test specification
• Date-based testing tools and test automation
• Planning and managing date-based testing

Exercises and Practice
• Identify date-based events based on the details of a case study and incorporate them into a life cycle model
• Develop a test calendar for the date-based events and a schedule of date changes together with a suitable strategy for changing the system clock during the testing effort

Exploratory Testing

Overview
Formal test specifications are good for validating that the software is fit for its intended purpose and verifying that it conforms to its specification. However, they are not good at harnessing the experience and creativity of testers.

In contrast, exploratory testing provides an opportunity for testers to experiment and probe by supplementing the test in the formal specification with their own. It is also the best way for testers to develop a sense of confidence in the software under test.

Learning Outcomes
• Explain that the goals of exploratory testing and when it is commonly performed (K2)
• Describe the relationship between exploratory testing, feature testing, end-to-end testing and date-based testing (K2)
• Explain the importance of relevant skills to exploratory testing and recall the skills required (K2)
• Explain how exploratory testing emphasises the use of checklists rather than detailed test specifications (K2)
• Explain the ways in which tools and automation can be used during exploratory testing (K2)
• Recall the main issues to take into account when planning and managing exploratory testing (K1)

Concepts
• What is exploratory testing?
• When is exploratory testing performed?
  o Exploratory testing and feature testing
  o Exploratory testing and end-to-end testing
  o Exploratory testing and date-based testing
• Skills required for exploratory testing
• Test specifications vs. checklists
• Exploratory testing tools and test automation
• Planning and managing exploratory testing

Exercises and Practice
• View some videos describing the principles of exploratory testing
• Discuss strategies for developing and sharing checklists

Managing the UAT and SIT Effort

Overview
All too often, the software quality message is buried in a mass of charts and reports making it difficult for managers to understand and act on what has been discovered during testing. In contrast, a few careful chosen key performance measures often present a better picture.

Learning Outcomes
• Explain the importance of triaging bugs before they are entered into the bug tracking system (K2)
• Explain the desirability of an individual or small group managing the bug tracking system (K2)
• Create a “burn down” chart that tracks test progress (K3)
• Plot the rate of finding errors to track product quality (K3)
• Plot the gap between finding and fixing bugs to track product readiness (K3)
• Explain the need for everyone involved with the product to freely express his or her level of confidence in the product (K2)

Concepts
• Managing issues
  o The importance of bug “triage”
  o Bug tracking
Requirements and change management

- Tracking progress
  - Using “burn down” charts to track test execution progress
  - Using the rate of finding bugs to track product quality
  - Using the gap between finding and fixing bugs to track product readiness
- Finding a voice for product confidence!

Exercises and Practice

- Draw some conclusions about the measured software quality by viewing a number of charts that describe progress of an imaginary test effort
- Discuss strategies for encouraging all involved to share their level of confidence in the software under test

Classification of Knowledge Levels

- **Level 1: Remember (K1)** - the student will recognise, remember and recall a term or concept.
- **Level 2: Understand (K2)** - the student can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify, categorize and give examples for the testing concept.
- **Level 3: Apply (K3)** - the student can select the correct application of a concept or technique and apply it to a given context.