

Software Engineering Best Practices: A Comprehensive Hands-On Introduction - 4 Days

Course 933 Overview

- You Will Learn How To**
- Analyse, design, program and test software projects
 - Elicit requirements and write user stories, use cases and use case diagrams
 - Draft mock-up user interfaces and create functional UI prototypes
 - Translate application requirements into working code
 - Simplify complex systems using modern object-oriented analysis and design techniques
 - Ensure software quality with both manual and automated testing techniques

Course Benefits Modern software development requires the collaborative effort of a diverse team with varied skills. To be effective, team members need to understand the activities performed at each stage in the development cycle. In this course, you analyse, design, implement and test applications that meet user requirements through a simulated case study. You gain hands-on experience performing each role within the development team using all the core concepts and skills necessary to engineer a successful program.

Who Should Attend Business analysts, QA testers, programmers, software designers, technical project managers and those who want an introduction to modern software development.

RealityPlus Through an evolving case study, you perform the typical roles and activities of software development team members. Team- and PC-based activities include:

- Eliciting requirements
- Writing user stories and use cases
- Sketching user interface mock-ups and creating UI prototypes
- Programming using a modern object-oriented language
- Modelling complex systems using UML class diagrams
- Implementing Model View Controller (MVC) design pattern
- Coding classes, inheritance and polymorphic behaviours
- Representing data relationships and entities
- Manipulating data with SQL
- Writing manual and automated tests

Software Engineering Best Practices: A Comprehensive Hands-On Introduction - 4 Days

Course 933 Outline

Introduction

Software development life cycle

- Identifying software development roles
- Matching roles to activities

Gathering software requirements

- Eliciting requirements from users
- Developing software iterations

Analysing User and System Requirements

Creating use case diagrams

- Identifying actors and use cases
- Representing user-system interactions

Capturing user stories

- Describing system functionality from the user perspective
- Recognising viable user stories

Detailing use cases

- Elaborating on complex system behaviours
- Scripting user and system conversations
- Documenting nonfunctional and system requirements

Designing User Interfaces (UI)

Refining the use case analysis based on user feedback

- Analysing the use case to determine system functional requirements
- Sketching a UI mock-up

Transferring your UI mock-up into a prototype

- Leveraging a prototyping tool
- Laying out screens and controls
- Setting form and control properties

Object-Oriented Programming

Handling and manipulating program data

- Declaring variables
- Defining data types
- Handling events and event-driven programming

Structuring application behaviour

- Controlling code execution with conditional logic
- Organising code inside functions
- Passing arguments to functions
- Calling and returning data with functions

Crafting an Object-Oriented Class Hierarchy

Refactoring code to improve design

- Applying the Single Responsibility Principle (SRP)
- Dividing functionality into classes
- Modelling applications with UML class diagrams

Simplifying UI code with the Model View Controller (MVC) pattern

- Separating UI and application logic
- Designing controller classes

Improving code maintainability with inheritance

- Removing code duplication
- Disentangling complex conditional logic

Modelling Classes and Objects

Constructing classes

- Defining fields and methods
- Encapsulating and accessing object data

Maximising program flexibility with inheritance and polymorphism

- Creating and realising base classes
- Defining virtual and abstract methods
- Overriding base class behaviour

Reusing code at runtime

- Instantiating classes and executing object behaviour
- Sending messages from objects using events
- Throwing and catching object exceptions

Saving Data to Storage

Defining data requirements

- Drawing UML data models
- Representing data relationships and multiplicities
- Programming entity classes

Creating and accessing relational databases

- Manipulating data with SQL insert, update and delete queries
- Retrieving data with SQL select queries
- Managing multiple records using collections

Testing and Deploying an Application

- Creating test plans
- Scripting user acceptance tests

- Automating unit tests
- Testing nonfunctional requirements
- Delivering a first iteration software project