

## Introduction to Modern Software Development - 4 Days

### *Course 933 Overview*

- You Will Learn How To**
- Analyze, design, program and test software projects
  - Adjust to change and manage risk using iterative and agile practices
  - Document requirements with user stories, use cases and use case diagrams
  - Draft mock-up user interfaces and create functional UI prototypes
  - 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 an agile team with varied skills. To deliver software on time and on budget, team members need to understand each stage of the development cycle, be able to monitor progress and adjust to change. In this course, you analyze, design, implement and test applications that meet user requirements through a simulated case study. You gain hands-on experience performing each role within an agile development team to engineer a successful program.

**Who Should Attend** Analysts, programmers, designers, architects, testers and other development team members who want an introduction to modern software engineering skills and techniques.

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

# Introduction to Modern Software Development - 4 Days

## Course 933 Outline

### Introduction

#### Agile software development

- Identifying software development roles and activities
- Comparing plan-driven vs. agile methodologies
- Waterfall
- V
- Spiral
- RUP
- XP
- Scrum
- Tracking and monitoring progress

#### Gathering software requirements

- Eliciting requirements from users
- Developing software iterations

### Analyzing User and System

#### Requirements

#### Creating use case diagrams and user stories

- Identifying actors and use cases
- Representing user-system interactions
- Describing system functionality from the user perspective

#### Estimating Requirements

- Accurately estimating user stories
- Translating relative estimates into time and money

#### Detailing use cases

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

### Designing User Interfaces (UI)

#### Refining the use case analysis based on user feedback

- Analyzing 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 behavior

- Controlling code execution with conditional logic
- Organizing code inside
- 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
- Modeling 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

### Modeling Classes and Objects

#### Constructing classes

- Defining fields and methods
- Encapsulating and accessing object data

#### Maximizing program flexibility with inheritance and polymorphism

- Creating and realizing base classes
- Defining virtual and abstract methods
- Overriding base class behavior

#### Reusing code at runtime

- Instantiating classes and executing object behavior
- 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