

Object-Oriented Analysis and Design Using UML: Hands-On - 5 Days

Course 323 Overview

- You Will Learn How To**
- Capture user requirements in use cases and transform them into detailed designs
 - Exploit the rich object-oriented modelling provided by the Unified Modeling Language (UML)
 - Adapt to changing requirements with iterative techniques and component-based design
 - Design agile solutions optimised for modern object-oriented languages and platforms
 - Refactor design models by applying proven design patterns
 - Verify implemented designs with automated unit and system tests

Course Benefits Object-oriented (OO) analysis and design is the principal industry-proven method for developing reliable, modular, testable programs and systems. This course provides practical skills in the latest OO requirements gathering, analysis, design, and testing methods. Intensive hands-on exercises offer you a working knowledge that turns concepts into practice.

Who Should Attend Anyone involved in developing systems on modern object-oriented platforms. Project teams benefit greatly by sharing the same methodology with codevelopers or with supportive management. Familiarity with basic OO concepts is helpful, but not assumed.

Hands-On Training Hands-on exercises provide experience using industry-standard UML case tools. Exercises and demonstrations include:

- Capturing and refining use case requirements
- Producing class and communication diagrams as part of an analysis model
- Transforming analysis behavioural models into design sequence diagrams
- Investigating round-trip engineering of source code
- Refactoring UML designs by applying design patterns
- Sharing models between developers using a CASE tool with a repository

Object-Oriented Analysis and Design Using UML: Hands-On - 5 Days

Course 323 Outline

Introduction and Overview

Using UML notation

- Use case diagrams
- Object models
- Packages and subsystems
- Interaction diagrams

Review of object-oriented concepts

- Classes, objects and attributes
- Encapsulation and interfaces
- Associations and multiplicity
- Inheritance and aggregation
- Polymorphism and collections

Establishing a methodology

- Contrasting Unified Process and Agile methods
- Applying the use case-driven approach
- Exploiting iterative incremental development

Producing Requirements Models

Capturing system behaviour in use cases

- Finding primary and secondary use cases
- Refining use cases with Include and Extend dependencies
- Modelling user interface requirements
- Validating user interfaces against use cases

Creating the domain object model

- Mapping ontological data structures onto a UML data model
- Building a class description repository
- Finding analysis classes
- Conquering complexity with packages and subsystems

Establishing the Object Model

Refining classes and associations

- Migrating analysis models to design classes
- Categorising classes: entity, boundary and control
- Modelling associations and collections
- Preserving referential integrity

Achieving reusability

- Isolating reusable base classes
- Reuse through delegation
- Improving reuse with design patterns

Generating the Behavioural Model

Use case realisation

- Extracting design sequence diagrams from use-case models
- Refining sequence diagrams
- Sharing models in a version-controlled repository

Implementing sequential behaviour

- Preparing UML state chart diagrams
- Nestable state machines and concurrency
- Capturing state machines from sequence diagrams
- Modifying the object model to facilitate states

Analysing object behaviour

- Modelling methods with activity diagrams
- Swimlanes, concurrency and synchronisation
- Restructuring using polymorphism and delegation
- Improving robustness using constraints, dependencies and the Object Constraint Language (OCL)

Object-Oriented Design

Design at the object level

- Designing and evaluating methods
- Synchronising dependent attributes
- Normalising classes with dependent data

System design

- Partitioning systems for deployment
- Persisting objects to databases
- Mapping designs to concurrent systems

Service-oriented architecture

- Distributing applications with Web services
- Applying component technology
- Deploying applications using components

Design Patterns

Purposes of design patterns

- Improving architecture, analysis models
- Achieving reuse, robustness and flexibility

Applying design patterns

- Achieving user interface independence
- Patterns for persistence
- Improving designs by refactoring

- Creational, behavioural and structural patterns

Testing Object-Oriented Designs

- Unit testing classes against their specifications
- Instituting automated object-oriented regression testing
- Validating implemented behavioural requirements
- Writing test scenarios from use case descriptions