

Modern Object-Oriented Software Design: Hands-On - 4 Days

Course 1801 Overview

- You Will Learn How To**
- Deliver software on time and within budget using iterative and Agile strategies
 - Capture accurate requirements using user stories and use case refinement
 - Strategically apply UML modeling to add value to the design process
 - Design highly reusable component-based object-oriented architectures
 - Produce flexible and adaptable software systems using iterative and incremental design
 - Guarantee robust implementations with test-driven development, refactoring and design patterns

Course Benefits In today's rapidly moving business environment, competitive advantage is achieved through the speedy delivery of responsive software that can adapt to evolving technology and changing user requirements. Applying UML modeling and Agile strategies is an industry-proven approach for developing such software. In this course you learn how to analyze, design and implement software using highly efficient, iterative and incremental methods.

Who Should Attend Software programmers and designers, team leads, project managers and requirements analysts. Basic familiarity with object-oriented concepts is assumed.

Hands-On Training Hands-on exercises provide experience using iterative and incremental UML and Agile methods. Exercises and demonstrations include:

- Expanding user stories into use cases
- Designing use case behaviour using UML sequence and activity diagrams
- Modeling complex behaviour with state charts
- Constructing a static architecture using class and component diagrams
- Producing and improving code using TDD
- Extracting and identifying design patterns in code

Modern Object-Oriented Software Design: Hands-On - 4 Days

Course 1801 Outline

Introduction

- Matching the method to the scale of the project
- Achieving agility through iterative development
- Modeling designs efficiently with UML
- Converting designs to software using test-driven development

Adapting the Method to the Project Appraising traditional approaches

- Critiquing waterfall and V-model life cycles
- Responding to change iteratively and incrementally

Exploring the iterative and Agile alternatives

- Identifying the risks of completely Agile approaches
- Reducing risk with UML-based design

Gathering Accurate Requirements Preparing for iterative and incremental development

- Identifying and involving stakeholders
- Capturing user stories and filling the backlog
- Refining requirements by expanding user stories into use cases

Planning an iterative cycle

- Estimating design and development work for user stories
- Soliciting priorities from stakeholders
- Handling incomplete and dependent user stories

Designing User Stories Efficiently with UML

Applying an appropriate amount of modeling

- Avoiding over- or under-modeling
- Modeling static structure: class and component diagrams
- Representing use case behaviour with activity diagrams

Designing the dynamic architecture

- Modeling use cases in three tiers
- Realizing use case behaviour with sequence diagrams
- Controlling alternative flows with UML state charts

- Mapping use case behaviour to model view controller (MVC) architecture

Representing the static architecture

- Preparing an entity model using classes and associations
- Confirming the data structure against the behavioural model

Engineering the Software

Documenting the detailed design with UML

- Constructing the implemented class diagrams
- Describing code behaviour with sequence and state diagrams
- Incorporating CASE tool models in iteration deliverables
- Specifying and designing method algorithms
- Improving robustness by modeling constraints

Establishing test-driven best practices

- Writing executable user story and use case tests
- Selecting the right unit tests: Equivalence partitions and Boundary values
- Automating the test process with unit testing and mocking frameworks
- Isolating components with Mock Objects

Refactoring for software excellence

- Improving reusability through the open/closed principle
- Reducing coupling and increasing cohesion through single responsibility
- Extracting interfaces and handling dependency inversion
- Segregating interfaces to maximize adaptability

Increasing design granularity by using patterns

- Decoupling behaviour with the strategy pattern
- Isolating the three tiers with MVC and observer patterns
- Centralizing object creation with factories

Integrating subsystems to create a functioning system

- Isolating tests by mocking downstream interfaces
- Automating use-case tests

Supporting the Iterative Process

Completing the iteration

- Validating completed user stories and use cases
- Delivering models and code into version control
- Tuning the process for subsequent iterations

Acquiring the right tools

- Comparing automated testing tools
- Supporting changes and bugs with tracking tools
- Replicating version control for requirements, models and code

Implementing UML and iterative best practices

- Identifying practices that can be utilized in the workplace
- Assessing which practices fit best in your organization