

Systems Analysis and Design: A Comprehensive Hands-On Introduction - 4 Days

Course 322 Overview

- You Will Learn How To**
- Analyse user requirements and design robust, change-tolerant software using UML
 - Select the right software architecture for your evolving business needs
 - Design a robust core of stored information for new or existing legacy requirements
 - Control complex behaviour for effective decision making and user interaction
 - Adopt a development process that ensures robust database and Web-enabled systems
 - Achieve optimum quality systems through UML techniques and supporting CASE tools

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 expectations. Controlling and managing such software depends on three critical elements: standards, architecture and process. This comprehensive introduction provides you with the knowledge and skills to effectively utilise these elements, delivering robust, future-proof software, particularly within database and Web-enabled environments.

Who Should Attend Systems analysts, business analysts and anyone involved in managing, specifying or designing products for business intelligence, database, Web-enabled, knowledge management, or user interaction software.

Hands-On Training Exercises and an evolving case study provide experience building "future-proof" software designs and include:

- Deciding the best migration strategy for legacy systems
- Establishing behavioural scope with UML use case diagrams
- Refining information structure for database design
- Describing control behaviour with a UML state chart
- Detailing control flow with UML activity diagrams
- Expanding a UML class diagram to show structure of the user interface
- Incorporating best practice into the software specification.

Systems Analysis and Design: A Comprehensive Hands-On Introduction - 4 Days

Course 322 Outline

Introduction and Overview

- Drawing diagrams to help us ask the right questions
- Dissecting UML 2 features
- An enterprise architecture: Information, Behaviour, Presentation
- Designing new or refining existing Web-enabled systems
- Exploring the Unified Process and the V-Model

Creating the Information Structure

Analysing information requirements

- Translating the business needs
- Structuring data with simplified UML class diagrams
- Establishing multiple or optional links
- Generalising and simplifying
- Connecting to legacy data systems

Achieving the best practice in data design

- Reducing redundancy with normalisation
- Developing the ontology
- Translating a data model to a relational database
- Managing data in a multitier Web-enabled environment
- Assessing design trade-offs

Formulating a physical data model

- Customising application and user views
- Partitioning data using packages
- Guaranteeing consistency and completeness
- The pros and cons of indexing with B-Trees
- Leveraging SQL Query Optimizers

Developing the Behaviour Model

Analysing behaviour requirements

- Scoping business behaviour with UML use case diagrams
- Realising a use case with a UML activity diagram
- Checking completeness and consistency
- Trading data complexity for control complexity

Determining best practice for application design

- Recognising UML stereotypes: process, boundary and entity
- Monitoring behaviour with UML communication diagrams

- Defining control using UML state charts
- Classifying stereotype responsibilities in SOAs
- Allocating behaviour in a Web-enabled environment

Forming the application architecture

- Managing application complexity
- Coupling and cohesion
- Creating congruent designs
- Matching process and data structure
- Measuring cyclomatic complexity

Presenting Component Objects to Users

Object-modelling techniques for analysis

- Assessing the benefits of an OO approach
- Mapping out structure at the user interface with detailed UML class diagrams
- Achieving consistency between UML class and communication diagrams
- Benefiting from inheritance as a consequence of generalisation
- Delegation arising from aggregation

Benefiting from best practice in component design

- Extending use case diagrams for user interface design
- Generalising actors and use cases
- Detailing mandatory reusable functionality with <<include>>
- Describing optional functionality using <<extend>>
- Improving the design of user interfaces: prototyping and polymorphism

Finalising the detailed use case

- Reusing knowledge with design patterns
- Model Driven Architectures
- Knowledge management

Achieving Optimum-Quality Results

Profiling the organisation

- Choosing appropriate personnel
- Matching the development approach to the organisational culture
- Nurturing the analyst's skill set
- Positioning UML within the overall software development process

Selecting the right software

- Database solutions: Sybase, SQL Server, Oracle, MySQL
- Exchanging data with XML
- Benefiting from Web technologies