(BA122) **Software Engineer's Workshop (SEW)**

Duration: 4 days

CDUs (*Continuing Development Units*): 28

Description: A practical workshop covering the role of the Business-Systems Analyst – a position combining both business and technical duties. The trainee develops a complete project starting from creating the business case and the requirements documentation and continuing right through to the detailed design of the databases and software required to implement the requirements.

This course is especially designed for the professional whose role includes both business and systems analysis on legacy and OO systems. A nuts-to-bolts analysis and design course that guides trainees through a complete project from requirements gathering and analysis to detailed software design specification. Covers project cost-benefit analysis, function-point cost estimation, use cases, data and object modeling, relational database design, normalization, ERDs, DFDs, structured analysis, OOAD, sequence diagrams, structured analysis, HIPO program specification for legacy systems, structured testing and more.

Why Attend this Course?

Take this course to learn how all the analysis and design tools and techniques fit together over the course of a project. Whether you are a BA wishing to expand your role into the technical realm or a Systems Analyst taking on more business-related tasks, you will learn to handle the complete picture- from requirements-gathering right through to testing and design. And you'll learn exactly what happens to the requirements once they've been thrown over the fence to the developers. Best of all, you'll learn by doing – by developing a case study over the course of the workshop.

What Makes this Course Stand Apart?

Breadth: One of the only courses out there to cover the complete project life-cycle from project justification right through to detailed design. If you want to know how the analysis techniques tie into the design, this is the course for you.

Depth: Detailed step-by-step directions give you precise instructions in using the tools and techniques necessary to carry out the Business-Systems Analysis function.

New and legacy systems: Most professionals must be able to work in both legacy and non-legacy (OO) environments. This course is unique in providing detailed guidance in both areas – just what you need to survive in today's' heterogeneous IT shops.

Job Aids: Each trainee receives invaluable Job Aids for use back at the office, including a completed case study and comprehensive glossary for use back at the office.

Audience

- Business-Systems Analysts
- Business Analysts expanding their role into the technical arena
- Systems Analysts
- Software Engineers

Prerequisites

None

Class Format

The course is an extremely hands-on one, with most of the focus on developing a case study from start to finish. Workshops are performed manually. Upon request, modeling tools such as Visio and Rational Rose may be incorporated into the program.

Objectives

Upon completion of this course, you will be able to:

- **Fill the role of Business/ Systems Analyst on an IT project on both legacy and non-legacy systems**

Detailed Objectives:

- Understand the role of the business-systems analyst
- Know the differences between the structured approach and the OO approach and when to use which one.
- Learn to work within both a waterfall and iterative project management methodology
- Employ use cases in project planning
- Perform cost-benefit analysis for IT projects
- Use Data Flow Diagrams (DFDs) for business analysis and for software design
- Create detailed programming specifications for legacy systems
- Use the principles of coupling and cohesion to design easily-modified software systems
- Describe process logic using structured English, decision table and trees
- Understand the types of databases currently in use
- Design an RDBMS database
- Normalize a data model
- Create use-case documentation and link this to the structural OO model (business classes and relationships)
- Analyze workflow using activity diagrams
- Understand OO concepts and use them in analysis and design
- Design technical software classes (interfaces, controller classes)
- Convert modeling elements (such as associations) into detailed programming elements and specifications
- Describe use-case scenarios with sequence diagrams and use these to design class operations.
- Convert class diagram into ERDs for RDBMS design
- Design both requirements-based and technical tests using structured testing, white box tests, boundary-value analysis and systems tests.

Content

- Overview of Software Engineering
  - > What kind of technical background does a typical entry-level SE have?
  - > What are the soft skills required for a SE?
  - > Responsibilities of the business-systems analyst
- Alternative approaches to software engineering
  - > Structured Analysis and Design approach
  - > The programming counterpart to Structured Analysis and Design
  - > The Object-Oriented Analysis and Design (OOAD) approach?
  - > The programming counterpart to Object-Oriented Analysis and Design (OOAD)
  - > Which approach to use and when
- Software Engineering toolkit
  - > Entity Relationship Diagrams (ERD)
  - > Data Flow Diagrams (DFD)
  - > Structure Charts / Hierarchy Charts
  - > Use Case Diagrams
  - > Class Diagram
  - > Activity Diagram
  - > Sequence Diagrams
  - > Unified Modeling Language (UML)
  - > Pseudocode and Tight English
  - > Decision Tables
  - > Structured Walkthroughs
  - > Structured Testing
  - > CASE Tools
- The Business Requirements Document (BRD)
  - > Business Requirements Document Template
- Financial and Managerial Fundamentals Every SE Should Know
  - > What is a project management methodology
  - > Waterfall SDLC
    - – Steps of the waterfall
    - – How much time is spent on each step?
  - > What is an iterative methodology?
    - – When to use iterative
    - – The SE role within the iterative approach
    - – Steps of the iterative methodology
    - – How much time is spent on each step?
    - – Planning releases using use cases and iterative development
  - > Estimating project time and cost requirements
    - – Options for estimating
    - – Estimating by analogy
    - – Estimating by extrapolation
    - – Cost per unit analysis
      - ∘ Steps for performing cost per unit analysis
      - ∘ Table: Rules of thumb for cost/unit analysis
    - – Table: Conversion Factors for Use in Time Estimates
  - > Calculating payback period
  - > Present value of money
  - > Future Value
  - > ROI
- High-Level Work Flow Analysis and Design using DFDs
  - > Steps of structured Analysis and Design
  - > The Data Flow Diagram (DFD)

Content con't…

- Why use DFDs?
- How do DFDs relate to other analysis tools?
- Physical and logical DFDs
- Procedure for analyzing and designing software using physical and logical DFDs
- Designing from the top down with leveled DFDs
- Defining scope on a Level 0 DFD
- Identifying interfaces between the system and the "outside world" on a DFD
- Designing subsystems and interfaces using a level 1 DFD
- Naming processes
- Tips for simplifying data flows
- Identifying data flows
- When are data stores added to the DFD?
- Drilling down using multiple-level DFDs

• Top-down analysis and design
  > Designing processes identified in the data flow diagrams
  > Structure charts
  > Specifying parameters and global variables
  > Reducing the 'ripple effect' of changes
  > Reducing coupling (data, control, pathological)
  > Optimizing cohesion
  > Categories of cohesion: Coincidental, logical, temporal, communicational, functional
  > Analyzing scope of effect and span of control
  > The scope of effect/span of control rule for evaluating design options
  > Improving design through the transform-centred model
  > Designing with structure charts
    - Indicating input and output parameters indicated on a structure chart
    - Designing switches; indicating switches on a structure chart
    - Indicating data items on a structure chart
    - Indicating loops on a structure chart
    - Indicating decisions on a structure chart

• Specifying Process Logic
  > What makes English ambiguous?
  > Logical constructs necessary to express logic: Sequential, iteration, case , selection
  > Documenting the constructs in structured English
  > Iteration Construct in Structured English?

• Analyzing complex business rules
  > Decision tree
  > Decision Table

• Creating design specifications using HIPO (Hierarchical, Input, Processing, Output) documentation
  > IPO chart

• Designing the Database
  > Overview of database design
  > Terms used in designing databases/data modeling
  > What is a database management system (DBMS)?
  > Sequential files
  > Why are sequential files still used today?
  > Indexed Sequential Files
  > The hierarchical DBMS
  > Network DBMS
  > RDBMS
  > OODBMS
  > The steps of database design (data modeling) for RDBMS:
  > Identifying Tables and Entities

Content con't…

- > Assigning attributes to tables
- > Defining keys
- > Designing cross-referencing between tables
- > Steps for Drawing ERDs
  - – Depicting entities and relationships
  - – Unary relationships
  - – Binary Relationships
  - – Cardinality
  - – Foreign keys
- > Normalizing the data model
  - – Why is it important to remove redundancies?
  - – The n Degrees of Normalization:
  - – Bringing the model to 1st, 2nd and 3rd normal form (3NF)
  - – When to de-normalize

• Analyzing Work Flow with OOAD (use cases, activity diagrams)
- > Steps of OOAD suing use cases
- > UML
- > Identifying use cases and actors
- > Do actors have an equivalent in structured analysis?
- > Defining high-level requirements with use-case diagrams
- > Discovering business classes
- > Using a use-case description template.
- > Specifying timing constraints on an activity diagram

• Modeling Classes (OOAD)
- > OO concepts
  - – Object
  - – Encapsulation
  - – Classes
  - – How does OO treat objects whose properties and behaviour are similar in some respects but different in others?
  - – OO terms relating to inheritance
  - – Multiple inheritance
  - – Aggregation
  - – Association
  - – Polymorphism
  - – How does polymorphism affect programming design?
  - – Concrete and abstract classes
  - – Objects send messages to other objects
- > Adding non-entity classes to the model
  - – The full set of class types
  - – How does the SE find interface classes?
  - – Controller class
  - – Utility classes

• Architectural layers
• How are Class Packages Physically Realized?
• Specifying inheritance
- > How is inheritance implemented in software?
- > Modeling inheritance in UML

• Designing aggregation
- > What are the programming implications of the "strength" of an aggregation
- > What does the SE have to specify about an aggregation aside from the fact that it exists?
- > Indicating aggregation and multiplicity in UML:

• Designing low-level OO Requirements (Sequence diagrams, database design)
• Specifying associations based on data relationships

Content con't…

- > What are data relationships?
- > How are they handled in OO?
- > How are associations implemented in programming?
- > Defining navigability
- > Association names and role names

• Specifying scenarios with sequence diagrams
- > Steps for drawing a sequence diagram
- > Deriving associations and class operations (methods) from sequence diagrams

• Deriving database requirements from the OO model
- > What's the relationship between the OO model and the database?
- > What types of database management systems work with OO?
- > Implementing Aggregation with RDBMS
- > Implementing Inheritance with RDBMS

• Software Testing
- > Principle of Structured Testing
- > When is testing done?
- > Where does testing documentation reside?
- > Tools of structured testing
- > Structure Walkthroughs: rules, roles, participants
- > Code inspections
- > White-box testing
  - – Coverage levels (statement, decision coverage, etc.)
- > Black-box testing
- > Deriving tests cases from decision tables
- > Use-case scenario testing

• Selecting test data using boundary-value analysis
• Unit testing
• What is unit testing?
- > Scheduling Techniques
- > Big Bang Testing
- > Top-down Testing
- > Bottom-up Testing

• Use case scenario testing
• System Testing
- > Volume testing
- > Stress testing
- > Usability testing
- > Security testing
- > Performance testing
- > Storage testing
- > Configuration testing
- > Compatibility testing
- > Reliability testing
- > Recovery testing
- > Acceptance testing
- > Beta testing
- > Parallel testing
- > Installation testing