

## **Writing Testable Software Requirements and Use Cases**

### **Course Summary**

#### **Description**

Poor software requirements are a major cause of costly rework, defects, delays, and dissatisfaction. Of all the factors that can impact requirements, lack of clarity is the most apparent. In turn, Testability—the ability to demonstrate that a requirement has or has not been met—is perhaps the single most effective indicator of clarity. If one cannot define how to test that a requirement has been met, then it's unlikely the developer will be able to tell how to meet the requirement correctly; and regardless there's no way to confirm the requirement was met. This interactive workshop consists largely of hands-on practice exercises writing clear and testable product/system/software/functional requirements/specifications and Use Cases.

#### **Objectives**

At the end of this course, students will be able to:

- Identify the three major sources of poor requirements that cause defects, rework, and cost/time overruns.
- Use common formats, including Use Cases, for more clearly documenting software requirements.
- Understand characteristics of "good" software requirements and ways to evaluate their clarity and testability.
- Write, review, and revise requirements so they are clearer and more testable.

#### **Topics**

- Essential Requirements Concepts
- Exercises: Writing Clear Wording
- Non-Textual Specification
- Writing Use Cases

#### **Audience**

This course has been designed for analysts, systems and business managers, project leaders, programmer analysts, quality/testing professionals, auditors, and others responsible for assuring requirements are defined adequately.

#### **Prerequisites**

There are no prerequisites required for this course.

#### **Duration**

Two days

## Writing Testable Software Requirements and Use Cases

### Course Outline

#### I. Essential Requirements Concepts

- A. Definitions of requirements
- B. Three causes of poor requirements
- C. Business vs. system requirements
- D. Requirements negotiation and transitions
- E. Problem Pyramid™ disciplined discovery
- F. "Inventing" the product/system solution
- G. Characteristics of "good" requirements
- H. IEEE Std. 830-1998 SRS structure, topics
- I. Business rules and pigeonholes
- J. Functional vs. non-functional requirements
- K. Common formats for specific requirements
- L. Common problems writing requirements

#### II. Exercises: Writing Clear Wording

- A. Magic words
- B. Complete
- C. Correct
- D. Necessary
- E. Consistent
- F. Unambiguous
- G. Testable
- H. Feasible
- I. Modifiable
- J. Ranked
- K. Traceable
- L. Objectively clarifying quality factors
- M. Reviewing, editing, and revising

#### III. Non-Textual Specification

- A. Decision trees and decision tables
- B. Cause-and-effect graphs
- C. Entity-Relationship diagrams
- D. Data models and process models
- E. Object models
- F. State transition diagrams
- G. Prototypes
- H. Test-driven agile XP code as requirements
- I. Proactive Testing™ test plans and designs
- J. How clear is clear

#### IV. Writing Use Cases

- A. Use cases defined
- B. UML use case diagrams
- C. Overall use case structure
- D. Identifying actors
- E. Defining business events
- F. Temporal events
- G. State events
- H. Includes and extends
- I. One-column use cases
- J. Happy path and alternative paths
- K. Two-column use cases
- L. Scenario relationship to test cases
- M. What use cases overlook
- N. Flowgraphing use case paths
- O. Functionality Matrix