Unified Modeling Language

Level: intermediate / advanced

Length: 21 – 35 hours, depending on the practical part

Course objective: learn to use UML in software development

What You Will Learn

- Learn the role of UML, how UML modeling is used in Object Oriented analysis (OOA) and Object Oriented Design (OOD)
- Introduces the diagram types in order to express a software construct or behavior, or to document a software system
- Learn to translate the UML diagrams to code

Who Can Attend: programmers who want to apply UML during software development

Prerequisites: because UML is usually used in object oriented programming context programmers should know at least at a medium level a particular object oriented programming language like C++, Java, C#, Python, etc.

Required Facilities: VGA projector, white board, workstation, development tools for writing programs in an object oriented language (Java, C++, C#, Python, etc.)

Related Courses: Design Patterns, Object Oriented Programming, object oriented programming languages (C++, Java, C#)

Description

The course offers a theoretical and practical approach of UML 2.x towards its usage by a programmer. The examples, case studies, hands on assignments offer a good understanding of UML diagrams. There are performed the following activities:

- Presentation of the main graphical elements, diagram types, their semantics and how they are used
- Understanding, „reading” the UML diagrams which were built by others
- Building of diagrams in order to express structures (static aspects) or behaviors (dynamic aspects)
- Implementation of UML diagrams („translation”) in one object oriented language (C++, Java, C#) in order to emphasize variants and particularities related to that language
- Using of UML for modeling during software development, at requirements specifications, object oriented analysis (OOA) & object oriented design (OOD)

Even UML is not bound to a particular software development process, it is important the understanding of modeling particularities of every phase or activity type belonging to the software development. To illustrate this, the Rational Unified Process (RUP) is shortly presented; there are emphasized modeling perspectives, views on a software system.

The course is not based on any particular modeling tool (for example IBM Rhapsody) or any editing tool of the UML diagrams.

Contents:

1. Introduction – the place of UML in software development, standard, current status; modeling – context, principles, views in modeling; software development process, tools, how is used

2. Class diagrams – express the domain vocabulary as a result of requirements analysis; practical project

3. Object diagrams – how to discover and / or verify the class diagram; exercise verifying the class diagram

4. Sequence diagrams – document the interaction between several entities; exercise this type of diagram for documenting a communication protocol

5. Communication diagrams – alternative to the sequence diagram, how to translate one to another

6. Activity diagrams – how to describe algorithms; exercise by describing how a component behaves based on how it is integrated in the whole system

7. Timing diagrams – describing how the states of several entities correlates in time; exercise this kind of diagram by describing how the states of several components which collaborate in a system correlates to each other

8. Use cases diagrams – how are they used to document what the system provides to the users, how they interact with the system in order to get the desired value; exercise this kind of diagram by specifying what a system provides, how it is used

9. Package diagrams – principles to group artefacts, what defines a good grouping

10. Component diagrams – how are they used to document the structure, the physical parts, how are they connected

11. Deployment diagrams – how are they used to document the system topology

12. Composite structures, collaborations, interaction overview diagrams

13. State machine diagrams – context, elements, how to describe & translate to code a state machine; project implemented by state machine from specs to code.