**Programming Paradigms Syllabus**

**Objective:**

The objective of this course is to learn major programming models such as the procedural and object-oriented models and other models that complements above said models such as : the functional, declarative/logic, Event Driven , Aspect Oriented and Concurrent models. Each of these models is based on a body of theory and each can be implemented in various programming languages. Although this course introduces the theoretical basis of each model, the emphasis is on the practice of programming.

A paradigm is a pattern of thinking that is frequently difficult to change. For example, the procedural and object-oriented models are based on the concept of state, as exemplified by the ubiquitous assignment statement. It may be difficult to imagine programming without an assignment statement, but the pure functional and declarative/logic models have no assignment statement. In both of these models, you will have to make a paradigm shift to thinking without assignment statements as you contemplate various programming solutions. Similarly, the procedural model is concerned with a single thread of control. It will take a paradigm shift to think about several processors executing several threads of control concurrently to solve a single problem.

In this course, we will immerse ourselves completely in each of these paradigms. That is, rather than discuss the ideas abstractly and concurrently we will attempt to experience them sequentially through programming. The course is thus divided into different parts, one for each of the programming models in which we will program with each of the three programming languages. The goal is not to make you an expert programmer in any of the three languages. Each language has far too many advanced features and nuances.

**Motivation**

A good background in different programming paradigm will give the ability to think outside the box of one language style or another. Without this background, Engineers would not be as valuable to any decidedly practically-minded company.

Once you have understood the general concepts of programming paradigms, it becomes easier to learn new programming language

**Learning outcomes**

Explain the nature of each paradigms Explain the relationships between functional, imperative, object oriented and other paradigms Explain the use of Each Paradigm in Practice Use clear communication of programming solutions, and of their derivation Distinguish between different programming paradigms Choose an adequate programming paradigm in solving specific software engineering problems Apply at least one language from imperative, object-oriented and declarative paradigm Classify programming languages according to the paradigms they belong to Recognize the concepts of same kind from different programming languages and paradigms

**Duration: 2 Days**

**Course Content**

1. Programming paradigms. The main programming paradigms are described. The students must be able to identify the main abstraction provided by each paradigm, and its suitability for the nature of a programming problem.
2. The object-oriented paradigm. This unit covers advanced elements of the object oriented paradigm (e.g., generics, design by contract and type inference). This unit is also aimed at clarifying the different elements provided by most object-oriented languages (inheritance, polymorphism, dynamic binding, encapsulation, information hiding…)
3. The functional paradigm. Students must be able to design and implement applications in this paradigm, using the suitable elements provided by the programming language. At the same time, he/she must be able to compare the functional approach with the object-oriented one.
4. Concurrent and parallel programming. Students must be able to know and apply the basic techniques of concurrent and parallel programming
5. Meta-programming and dynamic typing. Dynamically typed programming languages have influenced the development of software
6. Event-Driven-Programming: Help students to build simple interactive applications using Event-driven programming, where EDP is a popular approach to functional concurrency in which events, also known as “futures,” “deferred values,” or “lightweight threads,” execute concurrently with each other and eventually produce a result. The abstraction of the event has been extremely successful in producing lightweight, extensible, and efficient concurrent programs.