Design Patterns in Embedded Systems

Level: intermediate / advanced

Length: 20 hours

Course objective: exercise the design principles & patterns applied in embedded programming, how they are expressed in C / C++

What You Will Learn

- Learn the design principles, how they are applied in embedded programming
- Show the place and the role of design patterns in software development
- Study several design patterns and learn to apply them
- The optional part is dedicated to design patterns which appear in concurrent context, specific to multi-threading programming.
- Gain a valuable experience by covering diverse practical problems.

Who can attend: programmers who want to take advantage in every day work of the experience and knowledge expressed by the design principles & patterns.

Prerequisites: knowledge of C or C++ programming language at least at medium level, basic notions of object oriented programming, basics of UML, practical experience in embedded programming including knowledge of hardware.

Required facilities: VGA projector, white board, computers, development environment for writing code in C or C++.

Related courses: Unified Modeling Language; Applying of OOP, UML and Design Patterns, programming in C / C++.

Note: the course will be personalized based on the attendees' technical profile (namely which programming languages they work with, their work experience).

Description: the embedded programming is a programming in a restricted environment (memory, processor, etc.), usually close to the hardware. Besides general patterns embedded programming has specific problems which have specific solutions. By using C / C++ we shall apply several patterns specific to the embedded programming for solving several practical problems.

The course is highly interactive, it is an excellent opportunity to exercise programming in this environment.

Contents

1. Introduction: context, environment definition, specific problems

2. Design principles (SOLID): single responsibility, open / close, Liskov, interface segregation, dependency inversion – how are they applied, particular support in C/ C++

3. Design patterns for accessing the hardware: basic concepts, hardware proxy, hardware adapter, mediator, observer, polling, debouncing, interrupt

4. Concurrency patterns and resource management: basic concepts, patterns for tasks scheduling & coordination

5. Patterns for finite state machines: basic concepts, state, state tables

6. Patterns for safety and reliability: complement of one, CRC, smart data, channel, protected single channel, dual channel