Sela.

CleanCPP

Clean Code Workshop

college@sela.co.il 03-6176666

Clean Code Workshop

CleanCPP - Version: 1

2 days Course

Description:

Most of the code we write will accompany us for years and everything we write for the first time, is a draft. If that's the case, why not make it better? Make the code clean for our future selves to fix bugs and add features without slowing down, or fearing touching it? Clean code is code that is readable, maintainable, and flexible. There are effective, proven ways to achieve it. In this 2-day workshop, we're going to discuss and practice these methods.

Intended audience:

Software developers and team leaders. The course is targeted for programmers who would like to learn new skills for writing software. Attending this workshop will introduce new techniques and ideas on how to write and review quality code.

Prerequisites:

Objectives:

Write Simple, Expressive Code.
Identify and Fix Code Smells.
Refactor Legacy Code into Clean Code.
Use Visual StudioRrefactoring Features.
Understand and Apply SOLID (good code design) Principles .
Organize Code in a MaintainableWay.
Review Code Effectively.

Understand the Principles of Test Driven Development, and their Relationship with Clean Code.

Topics:

Clean code and Legacy Code

- What is clean code, what are the attributes, and how can we move
- from legacy, untestable and unmaintainable code to clean it

Simple Design and Coding by Intention

- The 4 rules of simple design are guidelines on how to write readable code,
- that also comes up with cleaner design. We'll discuss and have exercises on
- how to write code this way.

Naming

- Naming of modules, functions and classes are important for better code
- organization and maintenance. Exercises include refactoring code
- according to readable names. We'll also discuss how to review naming

Use of Comments

- Comments are mistreated, and often left there to die, creating confusing
- about what the code does. We'll discuss when and where to use
- comments, and how to use the code and tests to document itself

Code Styles

- Styles and conventions are usually created by the team level, and some
- coding guidelines selected at the organization level. We'll talk about
- defensive programming, coding with tests, and how they affect styles
- and guidelines. We'll also discuss different styles and when to apply them.
- Also what not to look for in code reviews.

SOLID Design Principles

- The SOLID principles of good design were named by Bob Martin, and they
- are the basis of not only readable and well-designed code, but also make it
- testable. In this part we'll go over the different rules, what they mean in code,
- and do exercises, refactoring and writing code using these rules.

Refactoring Principles With and WithoutTests

- Refactoring is a basic skill in coding. When assisted by tests refactoring is
- safer, without them it may be a necessity still. We'll go over refactoring
- patterns, and how they apply to known design patterns. In addition, we'll do
- refactoring exercises, and learn about tools (like ApprovalTests) to assist
- with refactoring.

CodeSmell Identification and Ffixing

- When you look at the code, and something bothers you about the design,
- we call it a code smell. Code smells allow us to identify design problems.
- We'll go over a list of known code smells, and do exercises about
- identification and fixing those smells.

Principles of Effective CodeReview

- What to look for, when to do it, relevance to clean code, good practices
- and tools - What to look for when doing code review? Defining the objective
- of code review is the beginning, as organizations waste time doing identifying
- unimportant coding problems and fixing them causing risks. We'll discuss
- online and offline code reviews, pair programming, and talk about how to
- come up with guidelines that fit effective work. We'll also practice this,
- focusing on the process, identifying problems and reporting them.

TDD

- Test driven development doesn't only add tests to the code, it also helps
- design it. Combined with the other clean code principles, it gets code clean
- as you write it