Clean Code: Software Craftsmanship

Course Summary

Description

Software development, like many other disciplines, is based on two distinct pillars:

- *A paradigm:* which is a common set of techniques, methods, principles, assumptions, best practices and other artifacts that define a body of knowledge about how to craft high quality code.
- *A production process:* which is a set of techniques and procedures that allow the production of code under the constrains of delivery on time, to spec and within budget.

Similar distinction can be made in other fields, such as cooking for example: a cooking paradigm would encompass the collection of techniques and skills that would be used by a chef to produce exquisite food, while the production process would be the set of techniques and skills used to run a kitchen in a restaurant to deliver those meals on time, on spec and to budget. The production processes of many disciplines share a common structure, whether it is developing code, running a restaurant, building a bridge or making movie: the sequence of requirements, analysis, design and construction whether it is expressed as a waterfall process or as agile process.

However, the current emphasis on process has neglected to a degree the importance of the craft or paradigm or programming which is expressed, for example in the Agile manifesto as the idea that success of software development is due in great part to the ability of programmer's skill to craft high quality code. One of the aspects of that craft of software development is the dimension of being able to write “clean code.”

Clean code can be described as code that is elegant: simple, easy to understand, support and modify and has is written to conform to the best practices of a programming paradigm: whether it is structured programming, object oriented programming, functional programming or other paradigm. Clean code is code that fits martin Fowler's comment “Anyone can write code a computer can understand, but a really good programmer can write code that other programmers can understand.”
This course is about writing clean code with an emphasis on the object oriented paradigm and languages like Java. C# and C++ although the concepts and practices are applicable to other languages like Python, .NET based languages and other related paradigms.

The emphasis of the course is on:

- Code design: organizing code components, modules and structures from a functional perspective. The basic principles of elegant and sound design (for example, use of interface and implementation).
- Code structure: planning and organization of code elements like classes, methods, packages and modules. Principles of good code structure, re-factoring and code smells.
- Code formatting: making code easy to read, easy to follow and modify or update.
- Effectiveness: working with frameworks, deigning code for reuse, developing best practices, the right amount of documentation.
- Best practices: using testing and test driven development, self-applied metrics, eliminating common sources of error, code organization and maintaining a healthy code base.

Topics

- Craftsmanship Defined
- Code Design
- Code Structure
- Code Formatting and Documentation
- Testing and Debugging
- Refactoring: Improving Structure
- The Code Base
- Using Frameworks and Tools
- The Programming Process
- Putting it all together

Clean Code: Software Craftsmanship

Course Summary (cont'd)

Audience

This course is designed for programmers.

Prerequisites

Students should be experienced in an object oriented language such as Java or C#.

Duration

Three days

ProTech
protechtraining.com

"Charting the Course ...

... to Your Success!"

Clean Code: Software Craftsmanship

Course Outline

I. Craftsmanship Defined
- A. Process versus paradigm
- B. Software development processes: waterfall, iterative and agile
- C. Craftsmanship: mastery of the paradigm
- D. Software paradigms: structured, OO, functional, logical etc
- E. Common aspects of all paradigms: the craft of developing code
- F. Dimensions of craftsmanship: design, structure, form, quality and effectiveness.
- G. Describing and defining well crafted code.
- H. Becoming a craftsman.

II. Code Design
- A. Good design is good design regardless of paradigm
- B. Fundamental characteristics of good design: modular, loosely coupled, etc.
- C. Using design to simplify code structure
- D. How programming languages are designed to support good code design.
- E. Best practices of design in OO program development.
- F. First Fundamental OO principle: Interface and implementation design
- G. Second Fundamental OO Principle: Recursive design
- H. The six principles of designing OO code.
- I. Design Patterns: reusing best practices.

III. Code Structure
- A. Classes, packages and methods: building clocks of code.
- B. Organizing code: the size of methods and classes.
- C. What makes methods and classes “good”
- D. Conceptual objects, using metaphors to make code understandable.
- E. Kinds of objects: data transfer objects, data structures, functional structures
- F. Using standardized algorithms and libraries
- G. Designing and using easily maintained structures
- H. Overview of best practices in structure: law of Demeter, Open close principle, etc.

IV. Code Formatting and Documentation
- A. Horizontal and vertical formatting, what works and what doesn't
- B. Measuring the “readability” of code.
- C. Naming practices and common mistakes
- D. Comments: when, why, how and mow much
- E. Developing code documentation for other developers.
- F. The specification: what makes a spec good enough to design code to: completeness, correctness, consistency , etc.

V. Testing and Debugging
- A. Planning for errors and exceptions.
- B. Basic test driven development: writing tests first
- C. How TDD improves the quality of the resulting code
- D. Automating testing: using Junit, etc.
- E. Avoiding creeping errors.

VI. Refactoring: Improving Structure
- A. Code smells: symptoms of poorly designed code.
- B. Refactoring: changing code structure without changing functionality
- C. Using TDD for controlled code changes
- D. The refactoring process.
- E. Using refactoring to make better code faster

VII. The Code Base
- A. Keeping your stuff organized
- B. Code repositories and version control
- C. Syncing up your work plan with your code assets organization
- D. Best practices in keeping your code base clean
- E. Creating your own code library and toolkit

VIII. Using Frameworks and Tools
- A. Common frameworks and code architectures: Spring etc..
- B. How much should your code rely on third party code?
- C. IDEs: Eclipse, Visual Studio etc: the good and bad of an IDE
- D. Avoiding common mistakes with IDEs
- E. The issue of making code IDE dependent

Clean Code: Software Craftsmanship

Course Outline (cont'd)

IX. The Programming Process

A. Knuth's Principle: First solve the problem then write the code.
B. Choosing a programming strategy, what gets coded first
C. Top down versus bottom up programming
D. Laying out your work plan
E. Estimating work and effort
F. Constant code improvement
G. Working with others: code reviews, formal and in formal
H. Language specific issues.

X. Putting it all together

A. Defining your goals in becoming a craftsman
B. Identifying your strengths and weaknesses
C. Developing an action plan for self-improvement
D. Emulating the best and avoiding common pitfalls.