Clean Code: Software Craftsman Advanced

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.

This course is a workshop oriented presentation and exploration of software craftsmanship within the object oriented paradigm. The course is built around a hands on development project which students are expected complete and have functional by the end of the course. Each module is designed to introduce a set of software craftsmanship principles which are then applied by the students in the further refinement of their project. Students work in groups following a simplified Agile development methodology in order to keep the focus of the material on what needs to be done from a software craftsmanship perspective and how that integrates with different processes types like waterfall, interactive and Agile without getting bogged down in process details. The emphasis of the course is on:

- Architectural design: of both code base and from application perspective with emphasis on these topics as work organization methods./
- 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.

Course Structure

The course is structured in a series of phases or iterations that follows what would normally be the sequence of development in a real life project. This is done for two reasons. The first is to create a learning environment that emulates a work environment as closely as possible. The second is to present material to the students at the point that it is most relevant to what they are doing and thus has a higher mastery and retention rate.
For each phase the objectives of the phase are identified and students start to implement the phase objectives. This is followed by a class discussion on what the issues were that were discovered, the a lecture section presents the relevant information addressing the issues the students raised as well as any they didn't. This is immediately followed by the hands on “re-work” session where the material is applied by the students in a mentored fashion.

Clean Code: Software Craftsman Advanced

Course Summary (cont'd)

Phase 1: Students are organized into teams, and have their first session with the client to scope their project. An iteration plan is developed and a basic proposal on code standards and the architecture of the code base us explored.
Phase 2: Students develop a functional architecture based on requirements. The architecture is used as an opportunity to explore code structure and design in accordance with good OO programming best design practices. .
Phase 3: Students get ready to write code by setting up their test driven development environment, develop their test cases and set up a code plan.
Phase 4: After the basic code is written and proven to be functional the students redesign their implementation using re-factoring to meet the non-functional requirements of the system.
Phase 5: After the code is tested, groups swap projects under the premise that each group will be maintaining and supporting another group's code. A final code review and analysis is performed.

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

Audience

This version of the course is for advanced students only. If differs in two primary ways from the introductory course. The first is that the programming project is significantly more difficult and challenging and requires substantially more effort and creativity from the students.
The second difference is the manner of presentation of the material. In this class the students will inductively explore what needs to be done as they work on the project which will then be confirmed in the lecture. For the introductory class the sequence is lecture-work, in the advanced class the sequence is work-lecture-rework

Prerequisites

This is a course for programmers. Students should be experienced in an object oriented language such as Java or C#. The course is four days in length. Students who cannot code in Java, C++, C# or a similar language competently will not be able to follow the course content.

Duration

Four days

Clean Code: Software Craftsman Advanced

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 words 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 Craftsman Advanced

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.