Course

Best Programming Practice and Design Pattern

Theoretical and practical course, 3 Credits

BEST PROGRAMMING PRACTICE & DESIGN PATTERN COURSE CONTENT

1. Effective Programming
2. Applying Test-Driven Development
3. Leveraging Design Patterns
4. Tuning for Maximum Performance
5. Architecting for Separation of Concerns
6. Mini-Project

Books

Agile Software Development, Principles, Patterns, and Practices	Robert C. Martin	©2003

1. Effective Programming in Java

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- Clarifying the goals of best practices
- Identifying the key characteristics of high-quality software
- Organizing classes, packages and subsystems into layers
- Designing to the principles of SOLID

2. Applying Test-Driven Development

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- **Exploiting a testing framework**
  - Composing and maintaining JUnit tests
  - Taking advantage of advanced JUnit features
  - Testing in the presence of exceptions
- **Monitoring software health using logging libraries**
  - Configuring logging with log4j and SLF4J
  - Minimizing the impact of logging on performance
- **Creating matchers and mock objects**
  - Writing custom Hamcrest matchers
  - Testing with fake objects and mocks

3. Leveraging Design Patterns

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- **Employing common design patterns** (Observer, Iterator, Template method, Strategy, State, Singleton, Data Accessor Object, Data Transfer Object, Composite, ServiceLocator, Proxy, Factory
- **Refactoring legacy code**
  - Identifying reasons to change software
  - Clarifying the mechanics of change
  - Writing tests for legacy classes and methods

4. Tuning for Maximum Performance

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- Measuring and improving performance
- Exploiting garbage collectors
- Taking full advantage of threads
- Bulletproofing a threaded application
- Exploring alternatives to synchronization

5. Architecting for Separation of Concerns

UPON COMPLETION OF THE MODULE THE STUDENT SHOULD BE ABLE TO:

- **Allocating responsibilities to components**
  - Translating method calls with the adaptor pattern
  - Adding behavior with a proxy
- **Centralizing the creation of objects**
  - Employing factories to achieve loose coupling
  - Implementing Inversion of Control (IoC)

Introduction to Best Programming Practice

- Programming is not only about learning and writing code, it is an **art**.
- It helps you, as a programmer to think differently and build a problem solving attitude within you.
- A simple program can be written in many ways if given to multiple developers. Thus, the need to best practices come into picture.
- It helps standardize products and help reduce future maintenance cost.

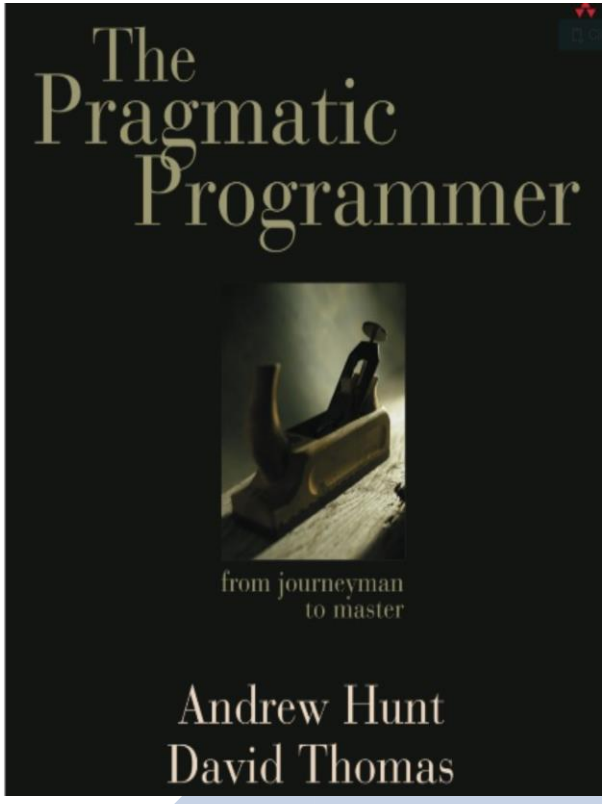
1. Introduction: Effective Programming in Java

- Is there an "ideal software development project," and, if so, what are steps you should take to achieve this ideal state? There are strict guidelines and different software development best practices methodologies such as scrum or extreme programming, but it's not always possible – or wise – to strictly follow these processes.
- It doesn’t imply that we don’t strive to accurately implement these methods; we just need to stay flexible. As part of this flexibility, we should think about overall structures that make a project successful and how consistency and coherence can improve your chances of achieving the "ideal."
- Because there isn't a "playbook" on the components of the ideal software development project, we pulled together some of our most tried and true best practices that make software development projects work better

1. Code Simplicity

- Strive to make your code simple.
- The idea is to reduce unnecessary complexity in software development

The code simplicity movement goes hand in hand with other software principles such as DRY (Don't Repeat Yourself), introduced in the book *The Pragmatic Programmer*, *and* YAGNI (You Aren't Gonna Need It), a mantra in agile development.



2. Testing

- Continuously test from end to end
- Test Driven Development (TDD) gives more confidence regarding code quality
- On the other hand, Behavior Driven Development (BDD) allows you to learn the features and requirements directly from the customer and that alignment translates into code that is closer to the users' needs.
- Full integration testing ensures that all components are working together as expected, and increases code coverage.

3. Code coherence

- Keep it consistent across your team
- When working with a team, it's important to have a consistent style guide for your codebase.
- There are many tools to enforce consistent style. 3 of them are:
  - JSCS (Java Script Code Style Checker)
  - SonarLint
  - Editorconfig

4. Code Reviews

- Don't be shy, allow someone to check your code.
- Code reviews help reduce bugs in the product –that's the bottom line- so give up that idea of perfection.
- "The Code Review Mindset" is a great article on the importance of code reviews

Everyone makes mistakes. An attitude which allows you to acknowledge imperfections is the first step to investing your trust in a code review.

5. Estimation

- Set your time and budget expectations realistically.
- It's hard to find a happy balance between being realistic and sandbagging when there are so many unknowns.

A realistic budget keeps your software project from feeling too much pressure. With agile methods, this enables the scope to flex more easily as the project progresses, but an estimate that is truly off can cause problems in quality, morale and output.

5. Estimation (cont'd)

- In reality, the "**ideal software development project**" may never exist!
- Each project has its own characteristics , flows, joys, sorrows, dreams, features, users, bugs, codebases, test cases, and many other components.

Rest assured, better estimation comes with experience, and there are many tools available to assist with software development project coding estimates.

1. Introduction - Effective Programming in Java

When in doubt, share your knowledge, encourage trust among your development team, stay positive, and remember you will ship.

Assignment

- Identifying the key characteristics of high-quality software
- Organizing classes, packages and subsystems into layers
- Designing to the principles of SOLID

Assignment 1

- **Exploiting a testing framework**
  - Composing and maintaining JUnit tests
  - Taking advantage of advanced JUnit features
  - Testing in the presence of exceptions
- **Monitoring software health using logging libraries**
  - Configuring logging with log4j and SLF4J
  - Minimizing the impact of logging on performance
- **Creating matchers and mock objects**
  - Writing custom Hamcrest matchers
  - Testing with fake objects and mocks