Proposal for CRAFTSMANSHIP AT CODE LEVEL

Contents

[VENDOR NAME 2](#_Toc33161830)

[PROGRAM NAME 2](#_Toc33161831)

[Program Summary 2](#_Toc33161832)

[Duration 2](#_Toc33161833)

[Cost 2](#_Toc33161834)

[Target Audience 2](#_Toc33161835)

[Terminal Objectives – what will the learner be able to ‘do’ at the end of the program? 2](#_Toc33161836)

[Day wise content outline 3](#_Toc33161837)

[Pre-course assessment 3](#_Toc33161838)

[Day 1: SOLID: Design Revisited 3](#_Toc33161839)

[Day 2: Technical Debt, Code Smells and Refactoring 3](#_Toc33161840)

[Day 3: Refactoring 4](#_Toc33161841)

[Day 4: TDD workshop 4](#_Toc33161842)

[Day 5: Putting it together 5](#_Toc33161843)

[References 6](#_Toc33161844)

# PROGRAM NAME

Craftsmanship at Code level

# Program Summary

This program will instil within each learner the values of a software craftsman. We believe that good programmers are “made” only if they work towards it. It emphasizes the fact that writing great code needs discipline with a sense of “cleanliness” and a structured approach towards “excellence”.

This program allows participants to explore practices, preventing defects and improve quality and reliability.

# Duration

10 Days ( 4 Hours/ Day)

# Target Audience

Programmers, Developers, Software Engineers, Team Leads

# Terminal Objectives – what will the learner be able to ‘do’ at the end of the program?

* Understand how “more” code leads to “complexity” and appreciate the importance of clean coding practices
* Distinguish between clean and dirty code
* Understand the importance of efficient, readable and maintainable code
* Understand how dirty code can generate more technical debt
* Appreciate the adoption of Test Driven Development, Continuous Integration and other software best practices to improve code quality
* Realize the cost savings for organizations through clean coding practices
* Internalize the passion required to practice clean code
* Software lifecycle management: Handling dependencies, constant version updates efficiently

# Day wise content outline

## Pre-course assessment

* Summative assessment to check familiarity with hands-on programming   
  (hands-on familiarity is a pre-requisite for the course)
* Formative assessment to identify focus-areas of participants

## Day 1: SOLID: Design Revisited

* Identify the User Stories (Cases) – This is where it starts
* Design starts from Abstraction - What is it?
* Separation of concerns through SOLID
  + Single responsibility: One level of abstraction, one reason for change  
    Naming Conventions – They Matter
  + Interface Segregation: Distinguish the How’s from the What; Expose what’s needed and hide implementation
  + Liskov Substitution: Subsets, subtypes and subclasses – what not to do
  + Dependency Injection: Separate the When from the What
  + Open-close: You can add without modification
* Design Disciplines
* Document your interface with tests and documentation

## Day 2: Technical Debt, Code Smells and Refactoring

* Technical Debt: What it is; The financial analogy; Is Your Code In Debt?
* Effects of Debt: Decreasing Customer Responsiveness
* Design Smells: Why care about them? What causes them?
* Types of Design Smells: Abstraction, Encapsulation, Modularization, Hierarchy
* How to Address Design Smells
* Exercises: Identify and Address:
  + Blotters
  + Disharmony: Identity, Collaboration, Classification
  + Couplers and Change preventers
  + Dispensable code: Dead code, Data class, comments, Duplication
  + Question complexity and speculative Generality
* Common smells in Test Code

## Day 3: Refactoring

**Refactoring-Code Smells**

* Why refactor
* Refactoring Flow: Tests to safeguard functionality and performance
* What, Where and When of Refactoring
* When not to refactor
* Exercises:
  + Method Extraction
  + Composing Methods
  + Moving Features Between Objects
  + Organizing Data
  + Simplifying Conditional Expressions
  + Making Method Calls Simpler
  + Dealing with Generalization

**Measurements and Visualization**

* Metrics and Thresholds
* Visualizing Metrics and Design
* Catalog of Metrics
* Tooling overview: Tools are aids, they don’t replace techniques!

**Repaying Technical Debt in Practice**

* Managing Technical Debt
* Big Picture Cost of Technical Debt
* The Tools
* The Process
* The People

**Evaluating the Design**

* Detection Strategies
* The Class Blueprint
* Conclusions and Outlook

## Day 4: TDD workshop

* Dissecting behavior
* Exercise on a Case Study: Experience TDD
  + Write a single failing test, implement to pass it
  + Isolate uncertainty in the environment
  + Is my test significant? Is it sufficient?
  + What is the residual risk?
* Evaluate metrics on your code
* Flavors of TDD
  + Specify before starting
  + Test and code together
  + Test, code and integrate: Fitting in the CICD
* Progressive reduction of risks with the test pyramid

## Day 5: Putting it together

* Confessions: Recognize behavior to change
  + Past review-records where things have been missed
  + Shortcuts that affected both quality and project schedules
  + Missed opportunities: What could have brought in efficiency and effectiveness?
* Committing to change: Identify an improvement you will do in your next work-item  
  (Individual exercise, done and reviewed in class)

## References

**Code Smells \*Martin Fowler**

**Blotters**

* Long Method
* Large Class
* Primitive Obsession
* Long Parameter List
* Data Clumps

**Object-Orientation Abusers**

* Switch Statements
* Temporary Field
* Refused Bequest
* Alternative Classes with Different Interfaces Types
* Identity Disharmonies
  + Rules of Identity Harmony
  + Overview of Identity Disharmonies
  + God Class
  + Feature Envy
  + Data Class
  + Brain Method
  + Brain Class
  + Significant Duplication
  + Recovering from Identity Disharmonies
* Collaboration Disharmonies
  + Collaboration Harmony Rule
  + Overview of Collaboration Disharmonies
  + Intensive Coupling
  + Dispersed Coupling
  + Shotgun Surgery
  + Recovering from Collaboration Disharmonies
* Classification Disharmonies
  + Classification Harmony Rules
  + Overview of Classification Disharmonies
  + Refused Parent Bequest
  + Tradition Breaker
  + Recovering from Classification Disharmonies

**Couplers**

* Feature Envy
* Inappropriate Intimacy
* Message Chains
* Middle Man
* Incomplete Library Class

**Change Preventers**

* Divergent Change
* Shotgun Surgery
* Parallel Inheritance Hierarchies

**Dispensable**

* Comments
* Duplicate Code
* Lazy Class
* Data Class
* Dead Code
* Speculative Generality

**Code smells (van Deursen, et al.)**

* Mystery Guest
* Resource Optimism
* Test Run War
* General Fixture
* Eager Test
* Lazy Test
* Assertion Roulette
* Indirect Testing
* Equality Test Code Duplication