BO·TT·EGA
IT minds

Training program:

Pragmatic Refactoring with Domain-Driven Design

Info:

Name:	**Pragmatic Refactoring with Domain-Driven Design**
Code:	**Craft-refactor**
Category:	Patterns and Craftsmanship
Target audience:	developers
Duration:	3 days
Format:	50% lecture / 50% workshop

Refactorings like “Extract Method”, “Replace If with Guardian” or “Extract Object” improve the readability, but don’t rescue projects, where the model is fundamentally broken. How to perform a refactor which looks like replacing an airplane’s engine? Of course, while flying. This workshop shows a handful of categorized typical problems gathered thanks to years of dealing with software on production, doing architectural audits and consulting in plenty of domains.

This is not yet another workshop where we go through a catalogue of typical code smells and ways of getting rid of those. During the coding exercises, we will analyze architectural problems which cannot be solved using standard refactoring techniques. We will take into account the fact that our software cannot afford a 3-month-long “rewriting” pause. Attendees will see a map that helps them navigate and solve architecture problems. Each problems is solved pragmatically, backed up with good engineering practices, Domain-Driven Design and rollback possibility.

We cover ways of communicating the technical debts to the business, soft skills that help us communicate the need of refactoring to people who we need to convince. We will talk about how to collect important business metrics which build a common understanding between IT and business stakeholders.

After the workshop, you will be ready to identify and repair architectural flaws and code problems using Domain-Driven Design, analyzing the history of your project, analyzing the business metrics, code stability, Test-Driven Development, object-oriented and functional programming, modularization and many more.

This workshop can last from 3 up to 5 days. The 4th and the 5th day can be dedicated to a fresh product, where we start from scratch, model and implement a green-field project, using the knowledge gathered within the first 3 days in the legacy context.

Alternatively, this can be applied as a series of consulting meetings, using your codebase.

BO·TT·EGA
IT minds

Training program

1. Common problems

- 1.1. What to do with a big and "unsightly" class?
- 1.2. What to do with a big but "aesthetic" class?
- 1.3. How to repair the data inconsistency problem?
- 1.4. My module/class has so many dependencies and so much business logic - how do I repair that?
- 1.5. The model of my module does not fit the reality - how to safely fix that?
- 1.6. I have so many inefficient reads from my database, the code looks ugly - what do I do?
- 1.7. There are no clear boundaries in my codebase, how to introduce some?

2. Proven Refactoring Path

- 2.1. Finding and describing the problem
  - 2.1.1. Architectural drivers
- 2.2. Marketing of your ideas
  - 2.2.1. Technical interlocutor
    - 2.2.1.1. Egoless programming
  - 2.2.2. Business/non-technical interlocutor
    - 2.2.2.1. The vector of new business possibilities
    - 2.2.2.2. The vector of new business possibilities for all new clients
    - 2.2.2.3. The vector of new business possibilities for one strategic client
    - 2.2.2.4. Safe rollback - we are not going to break anything
- 2.3. Recognizing the location of the problem
  - 2.3.1. Metrics gathering and interpretation
  - 2.3.2. Finding the critical points
  - 2.3.3. "Smells" identification

BO·TT·EGA
IT minds

BO·TT·EGA
IT minds

- 4.2. Current business function's flaws
- 4.3. Improvement of the quality attributes
- 4.4. Building trust
- 4.5. Business metrics as the key factor of the process of convincing

5. Deployment

- 5.1. Tests
  - 5.1.1. Finding "seams"
  - 5.1.2. Lowering the seams
  - 5.1.3. Characterization tests
  - 5.1.4. Tests and seams as a technique of finding modules boundaries and new business possibilities
- 5.2. Big Bang Release Problem
- 5.3. Step by Step Techniques
- 5.4. Parallel Models as the only sane way of safe strategic refactoring
  - 5.4.1. Safe rollback
  - 5.4.2. Logs
  - 5.4.3. Advanced metrics
  - 5.4.4. Safe-healing system
  - 5.4.5. Building trust with business from day 1

6. Use of Domain-Driven Design

- 6.1. Encapsulating the "WHAT" into application services
- 6.2. Encapsulating the "HOW AND WHY" into Domain-Driven Design building blocks
- 6.3. Ubiquitous Language
- 6.4. Value Objects is easier to deal with than Entity
- 6.5. Value Objects, Aggregates and Policies discovering
- 6.6. Policy - encapsulating the flexibility in a stable interface

BOTTEGA IT minds