

Training is withdrawn

Training program:

Refactoring legacy code to Domain Driven Design

Info:

Name:	Refactoring legacy code to Domain Driven Design
Code:	DDD-refaktoring
Category:	Domain Driven Design and Event Storming
Target audience:	analysts developers architects
Duration:	3 days
Format:	40% lecture / 60% workshop

During the training, participants will learn the techniques that allow for refactoring of existing systems in order to decrease the workload related to their maintenance.

A parallel goal of refactoring is the reverse engineering of a model that has not been maintained and has become diffuse.

An additional goal will be to learn the behaviour and structure in order to fix bugs, introduce new functionalities and refactor the design.

During the practical workshops, participants will refactor existing code and write regression tests. The participants will refactor the existing system, with the aim of introducing new functionalities.

The training should be preceded by trainings in the areas: [Domain Driven Design - complex business model](#) and [DDD Implementation Technician](#).

Training program

1. Reading the code

- 1.1. Collection and interpretation of metrics
- 1.2. Discovering the critical points of the system
- 1.3. Code review and identification of "smells"

2. The fundamentals of refactoring

- 2.1. Basic refactoring techniques
- 2.2. Support from the IDE
- 2.3. Identifying seams and dividing dependencies
- 2.4. Advanced refactoring in multiple steps
- 2.5. Creating a refactoring plan
- 2.6. Refactoring the design
- 2.7. Refactoring to a pattern

3. Use of modelling techniques in the refactoring process

- 3.1. Event Storming
- 3.2. Changing the approach in order to reduce the number of rules to be considered
 - 3.2.1. An approach "from the domain" instead of "from the process"
- 3.3. Linguistic techniques
 - 3.3.1. Extraction of the Domain Story
 - 3.3.2. Full sentence techniques instead of noun collection
 - 3.3.3. Exploring the domain using sentences in the form subject.verb(object, modifier)
 - 3.3.4. Gibberish Game - removing ambiguity and discovering new domain concepts
 - 3.3.5. Word-Meaning(Context)-Rules
 - 3.3.6. Reversing the order so as to discover hidden domain concepts

3.4. Visualisation techniques

3.4.1. Grouping the operation around invariants

3.4.2. The visual metaphors of real Aggregates

3.4.3. Levels of the model

3.4.4. Separation of the model according to susceptibility to change and instability

4. Logic stratification as the main refactoring strategy

4.1. Extraction of the Use Case/User Story in the application layer

4.1.1. Designing the system API

4.1.2. Encapsulation of WHAT the system should do into application services

4.2. Extraction of the business model from the building blocks of the domain layer

4.2.1. Encapsulation of HOW and WHY the system behaves in this way in the Building Blocks of DDD

4.2.2. The beginning of work - extraction of Value Objects

4.2.2.1. Work on the domain vocabulary

4.2.2.2. It is easier to manage a value than an entity

4.2.3. Extraction of Aggregates

4.2.3.1. Grouping attributes according to coherent change in the Use Case

4.2.3.2. Grouping attributes according to the protection of invariants

4.2.4. Domain Services - extraction of business sub-procedures

4.2.5. Policies - encapsulation of changeability beyond the stable interface

4.2.6. Factories - encapsulation of object creation in one place

4.2.7. Four levels of the model according to susceptibility to change

4.2.7.1. Capability

4.2.7.2. Operations

4.2.7.3. Policy

4.2.7.4. Decision Support

4.2.8. The search for stability

4.2.8.1. The open-close principle in practice

4.2.8.2. The stable model - Aggregates and Domain Services

4.2.8.3. Closures of the model - policies

5. Regression testing

5.1. Ways to test the system

5.2. Types of tests and examples of their use

5.3. Automation of the testing process

5.4. Choosing the testing strategy in the project

5.5. Writing automatic tests in a project that does not have them