BO·TT·EGA
IT minds

Training program:

Craftsmanship - professional tools for developers and architects

Info:

Name:	**Craftsmanship - professional tools for developers and architects**
Code:	**Craft-practices-przybornik**
Category:	Craftsmanship
Target audience:	developers
Duration:	3 days
Format:	50% lecture / 50% workshop

The training is a synthesis of crucial elements of classic and modern software engineering. It gives a general outlook on practical aspects of using discussed techniques in projects. Discussed issues are at the root of modern frameworks and technologies – which increases the level of their understanding and allows for a conscious use.

A professional in our approach:

- perfectly uses development techniques and architectural styles,
- can communicate with business using the DDD techniques,
- selects the right tool for the class of a problem provides a high quality code

The training is intended for programmers and designers who want to expand their competences in terms of professional techniques that increase the quality of code and design. Obtained knowledge translates in a practical way to the productivity, measured in a wider perspective of time.

It's all about the content.

- SOLID, GRASP and DDD techniques
- Setting techniques in application and system architecture
- All in the context of automatic testing
- Real-life examples

BOTTEGA IT minds

Training program

1. Object Oriented techniques

- 1.1. Orienting the thinking in the OO style
- 1.2. Code smell
- 1.3. Traps of inheritance
  - 1.3.1. Closing the code for extension
  - 1.3.2. Replacing the inheritance with composition – practical benefits of changing approach
    - 1.3.2.1. Inheritance is not suitable for role modeling
    - 1.3.2.2. Liskov Substitution Principle
    - 1.3.2.3. Party pattern
    - 1.3.2.4. Role Object and Extension Object patterns
- 1.4. GRASP - General Responsibility Assignment Software Patterns
- 1.5. Practical use of SOLID
  - 1.5.1. Class cohesion
    - 1.5.1.1. How to achieve it
    - 1.5.1.2. How to detect code smell
  - 1.5.2. Openness for extension
    - 1.5.2.1. Strategy pattern
    - 1.5.2.2. 3 types of logic: stable, closure, selection of closures
  - 1.5.3. When inheritance has no use
  - 1.5.4. Range of interfaces
  - 1.5.5. Proper level of abstraction
  - 1.5.6. Striving for a code with readability close to the prose
    - 1.5.6.1. The subject.predicate(object, attribute) technique

BOTTEGA IT minds

- 1.5.7. Invariant modeling technique
- 1.5.8. Technique for determining class boundary based on the Use Case analysis
- 1.6. Responsibility Driven Design

2. Clean Code

- 2.1. Detecting Code Smells
- 2.2. Selected implementation and design Patterns

3. Techniques for arranging the logic and application architecture patterns

- 3.1. Division into the application and domain logic
- 3.2. Application logic
  - 3.2.1. The Use Case/User Story modeling
  - 3.2.2. Stateful or stateless
- 3.3. Domain logic
  - 3.3.1. DDD techniques - Building Blocks
  - 3.3.2. Model levels
    - 3.3.2.1. Capacity
    - 3.3.2.2. Operations
    - 3.3.2.3. Policy – tuning the model
    - 3.3.2.4. Decision Support

4. System architecture patterns

- 4.1. DatAccess to data
  - 4.1.1. Repository
  - 4.1.2. ORM – range of applicability
    - 4.1.2.1. Traps of Lazy Loading – detection and fixing
    - 4.1.2.2. Optimistic locking, which will always work
- 4.2. Inversion of Control – tested concept of building frameworks and systems

BOTTEGA IT minds

- 4.2.1. Dependency Injection – the basis of modern frameworks
  - 4.2.1.1. Support for testability
  - 4.2.1.2. Practical techniques of use in order to achieve a flexibility of design
- 4.2.2. Event driven systems
  - 4.2.2.1. Plug-in architecture
  - 4.2.2.2. Separation of the modules
  - 4.2.2.3. Increasing the system responsiveness
  - 4.2.2.4. Scaling
- 4.2.3. Aspect Oriented Programming

4.3. Web application architectures

- 4.3.1. Classic n-layer architecture (variants)
- 4.3.2. Command-query Responsibility Segregation
- 4.3.3. Architecture supporting Domain Driven Design
- 4.3.4. Scalable systems

5. Testability – designing for TDD support

5.1. The Specify First approach

5.2. Behaviour Driven Development

5.3. Designing for tests with the use of OO techniques

5.4. Techniques for dependency reduction

5.5. Use of the Dependency Injection

5.6. Mapping a test pyramid on application layers

5.7. Application of Stub and Mock

5.8. Reducing the amount of testing accidents

- 5.8.1. Separation of logic
- 5.8.2. Functional objects