BO·TT·EGA
IT minds

Training program:

Design and architectonical patterns and effective Object Oriented Design techniques for system designers

Info:

Name:	**Design and architectonical patterns and effective Object Oriented Design techniques for system designers**
Code:	**craft-patterns-Patterns Sys**
Category:	Design patterns
Target audience:	architects
Duration:	3-4 days
Format:	50% lecture / 50% workshop

The training presents selected Design Patterns in a practical and non-textbook approach set in the context of designing libraries, frameworks, platforms and systems. During the training, examples of practical use are presented, taken from real systems of classes: ERP, visual tools, distributed systems, servers.

During the training, participants will obtain integrated knowledge about achievements of modern software engineering, allowing them to create advanced systems. During the practical workshops, we combine design and architectural patterns to create flexible and open for extension solutions, which are characterized by a high level of testability. Discussed issues are at the root of modern frameworks and technologies – which increases the level of their understanding and allows for conscious use. We present techniques of combining patterns into higher-order structures.

The training is intended for designers and architects, who want to expand their competences in terms of professional techniques of software engineering that increase the project and code quality.

It's all about the content.

- Focusing on the context of designing applications and systems
- Selecting only useful patterns and techniques
- Real-life examples

BO·TT·EGA IT minds

Training program

- **1. System architecture patterns**
  - 1.1. Layers
    - 1.1.1. What is layer for?
    - 1.1.2. Relaxed Layers
  - 1.2. Micro Kernel
  - 1.3. Command and Command Handler
  - 1.4. Ports and Adapters
  - 1.5. Restful
    - 1.5.1. 4 maturity levels
    - 1.5.2. REST as an application protocol, and not a transport one
  - 1.6. Event Driven
    - 1.6.1. Time dependencies between events
    - 1.6.2. Sagas/Process managers – event orchestration
- **2. Application architecture patterns**
  - 2.1. Division into application logic and domain logic
  - 2.2. Application logic
    - 2.2.1. Use Case/User Story modeling
    - 2.2.2. Stateful or stateless
    - 2.2.3. Using the Aspect Oriented Programming
  - 2.3. Domain Logic
    - 2.3.1. Domain Driven Design techniques - Building Blocks
      - 2.3.1.1. Aggregates
      - 2.3.1.2. Entities

BO·TT·EGA IT minds

- 4.2. Decorator – logic reusability
  - 4.2.1. Forming a complex, incremental business logic
  - 4.2.2. Wrapper – variant of a pattern useful in modeling oriented on meaning
  - 4.2.3. Combining a Decorator with a Strategy in order to build variants of incremental algorithms
- 4.3. Strategy – encapsulation of business logic
  - 4.3.1. Selecting a variant of algorithm without an interference in the business core
  - 4.3.2. Integration with dependency injection mechanisms
  - 4.3.3. Defining a specific business strategy in the Inversion of Control container (XML or factory methods)
- 4.4. Chain of Responsibility – two variants of the pattern
  - 4.4.1. Selection of a business logic for current conditions
  - 4.4.2. Combining a Chain with a Strategy in order to build variants of condition algorithms
- 4.5. Abstract Factory – creating domain artifacts
  - 4.5.1. Consistent way of creating families of business objects dependent on the system implementation configuration
  - 4.5.2. Producing Strategies
- 4.6. Builder – reduction of the complexity of structure creation
  - 4.6.1. Unified export of domain objects
  - 4.6.2. Hiding the complexity of query building
- 4.7. Template Method – process templates
  - 4.7.1. Business logic unification technique
  - 4.7.2. Template Strategies
  - 4.7.3. Examples of an anti-pattern
- 4.8. Singleton – a dangerous pattern on examples
  - 4.8.1. Details of implementing Singletons created with a delay, resistant to a concurrent access

- 5.7. Iterator – a standard pattern for a collection
- 5.8. Interpreter – a convenient pattern for creating own DSL
- 5.9. Observer – systems oriented on events, MVC Component
- 5.10. Extension Object – generalization of the Role Object for systems open to extension

6. Testability – impact of using OOD good practices and Patterns on the code testability

- 6.1. Issues of architecture susceptibility to tests: problems and traps
- 6.2. Techniques of unit testing: dummy, fake, stub, mock

7. Architecture documenting in the 4C approach - Context, Containers, Components, Classes

- 7.1. Context
  - 7.1.1. Context, in which the system will be working
  - 7.1.2. Main actors
  - 7.1.3. Processes on the level of organization, in which the system is involved
  - 7.1.4. The role of other systems
- 7.2. Containers
  - 7.2.1. Technical containers, in which the system will be arranged
  - 7.2.2. Integration patterns
  - 7.2.3. Communication protocols
- 7.3. Components
  - 7.3.1. Components and Services in terms of SOA
  - 7.3.2. Canonical data exchanged between the Services
  - 7.3.3. API design
- 7.4. The single source of truth rule
- 7.5. Classes
  - 7.5.1. Introduction to techniques and patterns of designing on the level of classes