BO·TT·EGA
IT minds

Program szkolenia:

Wzorce projektowe w C++

Informacje:

Nazwa:	**Wzorce projektowe w C++**
Kod:	**CCPP-craft-C++ Patterns**
Kategoria:	Craftsmanship dla programistów C i C ++
Grupa docelowa:	developerzy
Czas trwania:	3 dni
Forma:	40% wykłady / 60% warsztaty

Szkolenie prezentuje wybrane Wzorce Projektowe w praktycznym i niepodręcznikowym ujęciu osadzonym w kontekście projektowania bibliotek, frameworków, platform i systemów. Podczas szkolenia prezentowane są przykłady praktycznego zastosowania zaczerpnięte z rzeczywistych systemów serwerowych i rozproszonych.

Podczas szkolenia uczestnicy nabędą zintegrowaną wiedzę na temat zdobyczy nowoczesnej inżynierii oprogramowania pozwalającą im na tworzenie zaawansowanych systemów. Podczas warsztatów praktycznych łączymy wzorce projektowe i architektoniczne aby stworzyć giętkie i otwarte na rozbudowę rozwiązania cechujące się wysokim poziomem testowalności. Omawiane zagadnienia leżą u podstaw nowoczesnych frameworków i technologii – co zwiększa poziom ich zrozumienia i pozwala na świadome korzystanie. Przedstawiamy techniki łączenie wzorców w struktury wyższego rzędu.

Szkolenie przeznaczone dla programistów C++ i architektów pragnących poszerzyć swe kompetencje w zakresie profesjonalnych technik inżynierii oprogramowania zwiększających jakość kodu i projektu.

Zalety szkolenia:

- Skupienie na kontekście projektowania aplikacji i systemów
- Wybór jedynie użytecznych wzorców oraz technik
- Realne przykłady

BO·TT·EGA
IT minds

Szczegółowy program:

1. Object Oriented Design

- 1.1. Analiza Paradygmatu Object oriented i jego poprawna interpretacja
- 1.2. Ukierunkowanie myślenia w stylu OO
- 1.3. GRASP - General Responsibility Assignment Software Patterns
- 1.4. SOLID - Single Responsibility Principle (SRP), the Open/Closed Principle (OCP), the Liskov Substitution Principle (LSP), the Dependency Inversion Principle (DIP), and the Interface Segregation Principle (ISP)
- 1.5. Najlepsze praktyki i pułapki
  - 1.5.1. Domain Driven Design - wzmianka pierwsza
  - 1.5.2. Test Driven Development - wzmianka pierwsza
  - 1.5.3. POSA - Pattern Oriented Software Architecture

2. Paradygmat Inversion of Control – sprawdzona koncepcja budowy frameworków i systemów

- 2.1. Dependency Injection – podstawa współczesnych frameworków - dokładne omówienie z zadaniami
  - 2.1.1. Wsparcie dla testability
  - 2.1.2. Praktyczne techniki wykorzystania w celu osiągnięcia giętkości designu
  - 2.1.3. Wstrzykiwanie poprzez kontenery
  - 2.1.4. Wstrzykiwanie w Fabrykach
- 2.2. Systemy sterowane zdarzeniami – omówienie koncepcji i problemów
  - 2.2.1. Architektura pluginowa
  - 2.2.2. Separacja modułów
  - 2.2.3. Zwiększanie responsywności systemu
  - 2.2.4. Skalowanie

3. Wzorce projektowe - praktyczne, nieksiążkowe przykłady oparte o rzeczywiste problemy

BOTTEGA IT minds

- 3.1. Command – hermetyzacja usług, autoryzacja dostępu
- 3.2. Decorator – reużywalnośc logiki
  - 3.2.1. Składanie złożonej logiki biznesowej o przyrostowym charakterze
  - 3.2.2. Wrapper – odmiana wzorca użyteczna w modelowaniu zorientowanym na znaczenie
    - 3.2.2.1. Opakowanie typów podstawowych wygodnymi obiektami
    - 3.2.2.2. Alternatywa dla Utils
    - 3.2.2.3. Archetyp Money
  - 3.2.3. Połączenie Dekoratora ze Strategią w celu zbudowania odmian algorytmów przyrostowych
- 3.3. Strategy – hermetyzacja logiki biznesowej
  - 3.3.1. Wybór odmiany algorytmu bez ingerencji w core biznesowy
  - 3.3.2. Integracja z mechanizmami wstrzykiwania zależności
  - 3.3.3. Definiowane konkretnej strategii biznesowej w kontenerze Incersion of Control (XML lub metody fabrykujące)
- 3.4. Chain of Responsibity – dwie odmiany wzorca
  - 3.4.1. Dobór logiki biznesowej do aktualnych warunków
  - 3.4.2. Połączenie Łańcucha ze Strategią w celu zbudowania odmian algorytmów warunkowych
- 3.5. Abstract Factory – tworzenie artefaktów domenowych
  - 3.5.1. Spójny sposób na tworzenie rodzin obiektów biznesowych zależnych od konfiguracji wdrożeniowej systemu
  - 3.5.2. Produkowanie Strategii
- 3.6. Builder – redukcja złożoności tworzenia struktur
  - 3.6.1. Zunifikowane eksportowanie obiektów domenowych
  - 3.6.2. Ukrywanie złożoności budowania zapytań
- 3.7. Template Method – szablony procesów.
  - 3.7.1. Technika uwspólniania logiki biznesowej.

BO·TT·EGA
IT minds

3.7.2. Szablonowe Strategie

3.7.3. Przykłady antywzorca.

3.8. Singleton – niebezpieczny wzorzec w przykładach.

3.8.1. Szczegóły implementacji Singeltonów tworzonych z opóźnieniem, odpornych na współbieżny dostęp.

3.9. State – hermetyzacja procesu biznesowego.

3.9.1. Implementacja maszyny stanów reprezentującej złożony cykl życia obiektu biznesowego.

3.9.2. Maszyna Stanów jako Wrapper dodający nowe funkcjonalności.

3.9.3. Pułapki zbytniego uogólniania Maszyn Stanów

3.10. Observer – redukcja zależności

3.11. Event Broker - uogólnienie do poziomu Architektury Systemu

3.12. Wysokowydajne systemy Event Driven i asynchroniczne przetwarzanie.

3.13. Saga - orkiestracja wielu zdarzeń w czasie

3.14. Specification – hermetyzacja reguł biznesowych

3.14.1. Redukcja złożoności systemów zawierających złożoną logikę decyzyjną

3.14.2. Przypadek gdy istnieje wiele możliwych kryteriów logicznych

3.14.3. Jednak w danym kontekście (wdrożenie, klient) używanym tylko podzbioru reguł

3.15. Role Object

3.15.1. Modelowanie ról w systemie.

3.15.2. Alternatywa dla dziedziczenia.

3.15.3. Odmiana wzorca Bridge.

3.16. Facade – redukcja złożonej struktury pod wygodnym API

4. Użyteczne wzorce o zastosowaniu technicznym

4.1. Bridge – rozdzielenie interfejsu koncepcyjnego od jego implementacji

4.2. Flyweight, Prototype – wzorce optymalizacji

BO·TT·EGA
IT minds

- 4.3. Memento – komunikacja pomiędzy kontekstami
- 4.4. Proxy – podstawowy wzorzec frameworków
- 4.5. Composite – powtarzalne struktury
- 4.6. Visitor – dynamiczne rozszerzanie API Klasy (emulacja Double Dispatch)
- 4.7. Iterator – standardowy wzorzec dla kolekcji
- 4.8. Interpreter – wygodny wzorzec dla tworzenia własnych DSL
- 4.9. Observer - systemy zorientowane na zdarzenia, Składowa MVC
- 4.10. Extension Object - uogólnienie Role Object dla systemów otwartych na rozbudowę

5. Testability – wpływ użycia dobrych praktyk OOD i Wzorców na testowalność kodu

- 5.1. Zagadnienia podatności architektury na testy: problemy i pułapki
- 5.2. Techniki testowania jednostkowego: dummy, fake, stub, mock
- 5.3. Narzędzia testowania jednostkowego i integracyjnego
  - 5.3.1. GTest
  - 5.3.2. GMock