

Program szkolenia:

## **Test Driven Development w C++ od ogółu do szczegółu**

Informacje:

<b>Nazwa:</b>	<b>Test Driven Development w C++ od ogółu do szczegółu</b>
<b>Kod:</b>	<b>CCPP-craft-C++ TDD</b>
<b>Kategoria:</b>	Craftsmanship dla programistów C i C ++
<b>Odbiorcy:</b>	developerzy, architekci
<b>Czas trwania:</b>	3 dni
<b>Forma:</b>	40% wykłady / 60% warsztaty

---

Szkolenie przeznaczone jest dla programistów, którzy w codziennej pracy korzystają z języka C++.

## Szczegółowy program:

### 1. TDD – podstawowe założenia

- 1.1. Test(y) i automatyzacja
- 1.2. Implementacja bez szczególnego nacisku na jakość
- 1.3. Refaktoryzacja – konieczna część całego procesu

### 2. Dobry start

- 2.1. Makro-architektura podstawą sukcesu
  - 2.1.1. Paradygmat OOP i jego poprawne rozumienie
  - 2.1.2. Stosowanie sprawdzonych technik na rzecz testowalnego i rozszerzalnego kodu
- 2.2. POSA czyli architektura w oparciu o sprawdzone wzorce

### 3. Test(y)

- 3.1. Definicja punktu odciążenia (inflection point)
- 3.2. Projekt testu(ów)
- 3.3. Framework i jego możliwości

### 4. Implementacja

- 4.1. Dbalność o szczegóły a dbalność o architekturę
- 4.2. Implementacja zorientowana na działanie
- 4.3. Otestowanie zupełne – wprowadzenie testów dla tzw. „corner case'ów”

### 5. Refaktoryzacja

- 5.1. Czym jest refaktoryzacja?
- 5.2. Kiedy zacząć refaktoryzację?
- 5.3. Refaktoryzacja właściwa (do wzorców)
  - 5.3.1. Upraszczenie kodu
    - 5.3.1.1. Ekstrakcja metod

5.3.1.2. Strategia w zamian za naiwne warunki logiczne

5.3.1.3. Zamiana upiększeń w dekorator

5.3.1.4. Klasy stanu na rzecz wyrażeń zmian stanu

5.3.2. Tworzenie obiektów

5.3.2.1. Metody tworzące obiekty vs. naiwne konstruktory

5.3.2.2. Fabryka abstrakcyjna

5.3.2.3. Budowniczy dla tworzenia skomplikowanych struktur i kompozytów

5.3.2.4. Singleton – a może zło wcielone

5.3.3. Uogólnianie kodu

5.3.3.1. Metoda szablonowa – wprowadzenie dziedziczenia na rzecz polimorfizmu

5.3.3.2. Kompozyt – traktowanie zbiorów jak jednostki

5.3.3.3. Obserwator do obsługi notyfikacji

5.3.3.4. Adapter na rzecz unifikacji interfejsów

5.3.3.5. Interpreter – obsługa niejawnych języków

5.3.4. Prewencja

5.3.4.1. Akumulacja

5.3.4.2. Gromadzenie danych w parametrze zbierającym

5.3.4.3. Visitor do zbierania danych

## 6. Value objects – zamiana typów prostych klasami

## 7. Narzędzia

7.1. Cppcheck – statyczna analiza kodu

7.2. CPD – copy paste detector

7.3. CCCC – analiza złożoności logicznej

7.4. Valgrind – analiza sterty (memcheck), śledzenie wskaźników (ptrcheck), analiza wielowątkowa (hellgrind)

## 8. R.I.P. TDD – problemy

8.1. Over-engineering

8.2. Późna informacja zwrotna o trafności decyzji w fazie planowania architektury

8.3. Błędy logiczne

8.4. Trudności w dokumentowaniu

8.5. Przysłoczenie – trudności w wyobrażeniu testu

8.6. Trudności w ukryciu separacji API od szczegółów implementacyjnych

8.7. Trudności w komunikacji

8.8. Ciągły niepokój – czy wszystko zmierza w dobrym kierunku?

## 9. Zastosowanie TDD w pracy z kodem „legacy”

9.1. Bezpieczne wprowadzenie zmian w oparciu o TDD

9.2. Zbieżność metodyki postępowania

9.2.1. Inflection point

9.2.2. Test

9.2.3. Change

9.2.4. Check