BOTTEGA IT minds

Program szkolenia:

Programowanie współbieżne i rozproszone w języku C++: taktyki, wzorce, narzędzie wspomagające i testowanie.

Informacje:

Nazwa:	**Programowanie współbieżne i rozproszone w języku C++: taktyki, wzorce, narzędzie wspomagające i testowanie.**
Kod:	**ccpp-paralel C++**
Kategoria:	C i C++
Odbiorcy:	developerzy
Czas trwania:	3-4 dni
Forma:	35% wykłady / 65% warsztaty

Szkolenie prezentuje faktyczną możliwość (a nawet konieczność) stosowania Wzorców Projektowych podczas implementacji kooperujących aplikacji wieloprocesowych i wielowątkowych. Wszystkie wzorce przedstawiane są na przykładach wyciągniętych z rzeczywistych projektów.

W opozycji do akademickiego/książkowego podejścia, uczestnicy szkolenia zdobywaną stopniowo wiedzę jednocześnie wykorzystują i testują przy okazji implementacji zadań warsztatowych, które odzwierciedlają wyzwania napotykane w rzeczywistości, tworząc przy tym skalowalny kod, otwarty na zmiany i podatny na testowanie.

Szkolenie przeznaczone jest dla programistów C++ tworzących aplikacje serwerowe, chcących poszerzyć swoje kompetencje w zakresie profesjonalnych technik programistycznych zwiększających jakość kodu i projektu. Zdobyta wiedza przekłada się w praktyczny sposób na produktywność mierzoną w szerszej perspektywie czasu.

Zalety szkolenia:

- Skupienie uwagi na "fizyce" problemu programowania współbieżnego i rozproszonego
- Wybór jedynie użytecznych wzorców i technik
- Rzeczywiste przykłady

BOTTEGA IT minds

Szczegółowy program:

1. OOP – Object Oriented Programming – przypomnienie podstawowych zasad programowania obiektowego w świetle rzeczywistych problemów.

1.1. Paradygmat programowania obiektowego

1.1.1. Analiza paradygmatu programowania obiektowego i jego poprawna interpretacja

1.1.2. GRASP – General Responsibility Assignment Software Patterns (Principles).

1.1.3. 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).

1.1.4. Defense In Depth – programowanie defensywne

1.2. POSA – Pattern Oriented Software Architecture – architektura oparta na wzorcach

1.2.1. Wzorzec – czym jest?

1.2.2. Pattern language – architektura rozwiązania (rozwiązań) oparta na wzorcach

1.2.3. Framework – zbiór reużywalnych, zaimplementowanych wzorców i funkcjonalności

2. Programowanie Współbieżne – podstawowe pojęcia w starciu z rzeczywistością (wymaganiami funkcjonalnymi i niefunkcjonalnymi).

2.1. Aplikacja, program, proces, wątek – poprawne rozumienie podstawowej nomenklatury

2.2. Synchronizacja

2.2.1. Zrozumienie pojęcia sekcji krytycznej

2.2.2. Mechanizmy blokad i oczekiwania

2.2.3. Pułapki i typowe błędy

2.3. Idiomy i wzorce wspomagające programowanie równoległe

2.3.1. Wątek, grupa wątków, pula wątków

2.3.2. Kolejki – dobór algorytmu do problemu, a nie odwrotnie!

2.3.3. Wraper-fasada vs Proxy – różnice oraz aspekt przenośności

2.3.4. Template method vs RAII (Resource Acquisition Is Initialization) – różnice oraz możliwość połącznenia

2.3.5. Monitor – upraszczanie zależności

2.4. Prawa równoległości

2.4.1. Amdahl vs Gustawson

2.4.2. Konfrontacja prawideł matematycznych a szara rzeczywistość

2.5. Testowanie

2.5.1. Testowanie jednostkowe

2.5.2. Testowanie modułowe

2.6. Third-party w ujęciu rozmaitych pułapek

2.6.1. Zrozumienie kontekstu języka C++ - język niskopoziomowy

2.6.2. POSIX vs reszta świata oraz aspekt C++11

2.6.3. Przegląd bibliotek

2.6.4. Narzędzia wspomagające

3. Komunikacja międzyprocesowa – wymiana danych pomiędzy aplikacjami zdalnymi i lokalnymi w kontekście rzeczywistych problemów.

3.1. Potoki i gniazda – różnice i praktyczne zastosowanie

3.2. Demultipleksacja – obsługa komunikacji z wielu źródeł

3.2.1. Select, poll, epoll

3.2.2. Wykorzystanie podejścia „Event Oriented”

3.2.3. Analiza aspekt współbieżności – dobór wzorca i możliwość podmiany

3.3. Wzorce

3.3.1. Wrapper-fasada – obiektowe modele dla kodu „error prone” i legacy API na rzecz reużywalnych komponentów

3.3.2. Proxy – implementacja konkretnych funkcjonalności z użyciem przygotowanych fasad i wrapper’ów

3.3.3. Silniki obsługi zdarzeń

3.3.3.1. Observer, reactor, proactor, half sync/half asyc, thread pool – niby to samo, a jednak inaczej

BO·TT·EGA IT minds

- 3.3.3.2. Event broker
- 3.4. Testowanie
  - 3.4.1. Unit vs Module –które części kodu i jak je testować
  - 3.4.2. Stress tests – „dmuchanie na zimne”
  - 3.4.3. Testowanie funkcjonalne
- 3.5. Third-party i gotowe rozwiązania
  - 3.5.1. Narzędzia wspomagające
    - 3.5.1.1. Valgrind – poprawność użycia sterty (memcheck), przepływ wskaźników (ptrcheck), poprawność wielowątkowości (hellgrind)
    - 3.5.1.2. Cppcheck – analiza statyczna kodu
    - 3.5.1.3. CPD – wykrywanie zduplikowanego kodu i kontekst kodu legacy
  - 3.5.2. Elementy ułatwiające pisanie aplikacji rozproszonych
    - 3.5.2.1. RPC – Remote Procedure Call – standardy wywołań funkcji zdalnych takie jak: CORBA, WebServices
    - 3.5.2.2. REST – Representational State Transfer – bardziej filozofia niż formalny standard
  - 3.5.3. Biblioteki i frameworki (zakres szczegółowości do uzgodnienia)
    - 3.5.3.1. Boost::ASIO
    - 3.5.3.2. POCO
    - 3.5.3.3. ACE i TAO