Projektowanie obiektowe oprogramowania Wykład 13 – Testowanie oprogramowania Wiktor Zychla 2012

1 TDD vs BDD

Test-Driven Development (TDD) – rozwijanie oprogramowania sterowane testami

System Under Test (SUT) – klasa użytkowa, która jest podmiotem testu jednostkowego.

Collaborators – klasy usług pomocniczych, z których korzysta klasa SUT, ale które nie są podmiotami testów. Myślimy o architekturze, w której usługi pomocnicze są wstrzykiwane do klas użytkowych.

Czy do testu jednostkowego dobrze jest używać rzeczywistych implementacji usług pomocniczych?

NIE!

Jeśli na przykład usługa dodatkowa wysyła maile czy drukuje dokumenty, to skutki uboczne testów jednostkowych mogą być niepożądane.

Rozwiązanie? Dubler.

Test Double (Dubler) – klasa implementująca usługę, zastępująca prawdziwą implementację podczas testowania

Interfejs „udawanej" usługi można zaimplementować na różne sposoby:

- **Dummy** - implementacja, która w ogóle nie jest wykorzystywana, a jej jedynym celem jest wypełnienie listy usług wstrzykiwanych do klasy SUT. Poszczególne metody implementacji typu Dummy mogą nawet wyrzucać wyjątki typu NotImplementedException, bo klasa SUT w ogóle tej usługi nie będzie wykorzystywać w danym teście.
- **Fake** – implementacja, która faktycznie działa i nawet robi to co powinna, ale sposób jej implementacji wyklucza jej produkcyjne wykorzystanie. Przykład to implementacja repozytorium, która utrwala obiekty w pamięci operacyjnej.
- **Stub** – implementacja, która niekoniecznie działa zgodnie ze specyfikacją funkcjonalną; poszczególne metody zwracają wyniki spreparowane pod kątem konkretnego testu/testów.
- **Mock** – *Mocking Object*, typ zastępczy, gotowa rama aplikacyjna dostarczająca implementacji usług pod kątem testowania BDD

Behavior Driven Development (BDD) – TDD, w którym testuje się *zachowanie* implementacji SUT i usług pomocniczych a nie ich stanu.

Uwaga! W praktyce rama typów zastępczych pozwala na testy zachowania i testy stanu.
Uwaga! Testowanie stanu możliwe jest również bez ramy typów zastępczych.

Przykład na żywo.

2 Design by Contract

Design by Contract – technika projektowania obiektowego, w której częścią interfejsu metod i klas są zobowiązania dotyczące **stanu** w określonych momentach obliczeń:

- **precondition** (warunek wejścia) – stan w chwili rozpoczęcia wykonywania się metody
- **postcondition** (warunek wyjścia) – stan w chwili zakończenia wykonywania się metody
- **invariant** (niezmiennik) – stan w określonym momencie wykonywania się metody

Warunki mają zwykle postać predykatów (formuł logicznych typu boole'owskiego) wyrażonych w języku logiki pierwszego rzędu.

OCL (Object Constraint Language) – uniwersalny formalizm zaprojektowany do wyrażania kontraktów DbC w językach obiektowych, stosunkowo mało rozpowszechniony.

Podstawowa technika weryfikacji kontraktów to weryfikacja dynamiczna. Należy umieć „przechwycić" moment wywołania metody i moment zakończenia wykonywania się metody i zweryfikować poprawność formuły logicznej (czyli czy po podstawieniu wartości za zmienne formuła ewaluuje się do **true**).

Z kolei samo przechwytywanie może mieć postać:

- **statyczną** – na etapie kompilacji do metod wstrzykiwane są dodatkowe wywołania funkcji z API technologii DbC, które służą weryfikacji kontraktów
- **dynamiczną** – przechwytywanie wywołania odbywa się w trakcie działania programu.

Alternatywą dla weryfikacji dynamicznej jest weryfikacja statyczna – co w ogólności oczywiście nie jest możliwe ponieważ problem statycznej weryfikacji kontraktów jest nierozstrzygalny. Oznacza to, że wydajny algorytm może mylić się na „niekorzyść", tzn. uznać za niepoprawny taki kod, który w rzeczywistości jest poprawny.

Metody formalne są już na tyle dobrze rozwinięte, że odpowiednie narzędzia są częścią technologii przemysłowych – przykład: Code Contracts w VS2010.

Przykład na żywo.

3 Testowanie interfejsu użytkownika – UI Automation

Za pomocą dedykowanych ram aplikacyjnych do automatyzacji interfejsu użytkownika, możliwe jest budowanie testów akceptacyjnych od strony interfejsu użytkownika aplikacji.

Przykład na żywo.

4 Literatura

Martin Fowler – „Mocks Aren't Stubs", http://martinfowler.com/articles/mocksArentStubs.html

Gerard Meszaros – "xUnit Test Patterns", http://xunitpatterns.com/