Metodyki zwinne wytwarzania oprogramowania

Wykład 4

Marcin Młotkowski

26 października 2016

Plan wykładu

Najsłynniejsze i najdroższe błędy programistyczne

Ariane 5 lot 501

Użycie oprogramowania z Ariane 4 (**370 mln USD**).

Najsłynniejsze i najdroższe błędy programistyczne

Ariane 5 lot 501

Użycie oprogramowania z Ariane 4 (**370 mln USD**).

Apollo 11

Błąd radaru zbliżeniowego wygenerował serię przerwań komputera nawigacyjnego.

Najsłynniejsze i najdroższe błędy programistyczne

Ariane 5 lot 501

Użycie oprogramowania z Ariane 4 (**370 mln USD**).

Apollo 11

Błąd radaru zbliżeniowego wygenerował serię przerwań komputera nawigacyjnego.

Błąd w aparacie Therac-25 do radioterapii

Wyścigi wątków/procesów spowodowały 200-krotne przekroczenie dawki podczas napromieniania.

Cele testowania

Weryfikacja

Sprawdzenie, czy oprogramowanie jest zgodne ze specyfikacją.

Cele testowania

Weryfikacja

Sprawdzenie, czy oprogramowanie jest zgodne ze specyfikacją.

Walidacja

Sprawdzenie, czy oprogramowanie jest zgodne z oczekiwaniami użytkownika.

Typy testów

Testy białej (przezroczystej skrzynki), testy strukturalne

Testy, które mają na celu przejście wszystkich możliwych ścieżek w programie.

Typy testów

Testy białej (przezroczystej skrzynki), testy strukturalne

Testy, które mają na celu przejście wszystkich możliwych ścieżek w programie.

Testy czarnej skrzynki (testy funkcjonalne)

Testy przygotowane na podstawie specyfikacji.

Hierarchia testów

- testy jednostkowe
- testy integracyjne
- testy akceptacyjne

Przypadek testowy (test case)

Zbiór testów oceniających, czy aplikacja spełnia warunki określone w *przypadku użycia*.

Test jednostkowy (unit test)

Test testujący pojedynczy element programu (procedurę, klasę, moduł).

Test jednostkowy (unit test)

Test testujący pojedynczy element programu (procedurę, klasę, moduł).

Sposób testowania

- sprawdzenie, czy dla przykładowych danych są zwracane właściwe wyniki;
- sprawdzenie reakcji na błędne dane (np: zwrócenie wyjątku);
- kontrola stanu aplikacji po wykonaniu operacji (np: operacja wstawienia do bazy danych).

Schemat działania testu jednostkowego

Schemat działania pojedynczego przypadku testowego:

1. utworzenie stanu podstawowego;
2. wykonanie testowanych operacji;
3. weryfikacja wyników;
4. przywrócenie stanu początkowego.

Uwagi

- Testy jednostkowe (i ich zestawy) to mogą być programy (skrypty etc) lub scenariusze do ręcznego testowania;
- test jednostkowy może składać się z kilku drobniejszych testów;
- testy powinne być "zewnętrzne" w stosunku do testowanej jednostki;
- zestawy testów umożliwiają prowadzenie *testowanie regresyjne*, tj. czy drobne modyfikacje nie wprowadziły błędów.

Środowiska xUnit

- są to środowiska dedykowane dla poszczególnych języków (Java, Python, C#)
- są to środowiska oparte o obiekty.

Podstawowe elementy środowiska xUnit

Przypadek testowy

Jest to klasa, pochodna klasy `*Unit`

Testy

Pojedyncze testy są implementowane jako metody.

Test fixtures

Co to jest

Ustalony stan początkowy testu.

Test fixtures

Co to jest

Ustalony stan początkowy testu.

Przykład

Pusta (ale istniejąca) baza danych.

Test fixtures

Co to jest

Ustalony stan początkowy testu.

Przykład

Pusta (ale istniejąca) baza danych.

Realizacja

Metody:

- setup()
- teardown()

Kontrola poprawności

Asercje

Metody, które sprawdzają czy zwracane wyniki są zgodne z oczekiwaniami.

Kontrola poprawności

Asercje

Metody, które sprawdzają czy zwracane wyniki są zgodne z oczekiwaniami.

Asercje robią wiele innych rzeczy: reagują na nieprzewidziane wyjątki, mierzą czas, etc.

Uzupełnienie

Kolekcje testów (test suites)

Kolekcje testów, które można uruchamiać dla całego istniejącego oprogramowania.

Przykład: NUnit

Zadanie

Implementacja stosu.

Implementacja

```
public class Stos<T>
{
   public Stos() { ... }

   public bool isEmpty() {  ... }

   public void push(T val) { ... }

   public T pop() { ... }
}
```

```
[TestFixture()]
public class TestowanieStosu
{

    [Test()]
    public void TestEmpty()
    {
        Stos<int> s = new Zwinne.Stos<int>();
        Assert.AreEqual(s.isEmpty(), true);
    }
    [Test()]
    public void TestPush()
    {
        Stos<int> s = new Zwinne.Stos<int>();
        s.push(1024);
        Assert.AreEqual(s.pop(), 1024);
    }
```

Plan wykładu

Ogólna definicja

Test akceptacyjny

Test przygotowany przez klienta/użytkownika.

Ogólna definicja

Test akceptacyjny

Test przygotowany przez klienta/użytkownika.

Test jest wyrażany w języku klienta.

Testy akceptacyjne w praktyce

Co może być testem akceptacyjnym

To mogą być przykłady podane przez klienta, na przykład w postaci scenariusza określającego stan początkowy, akcję i stan końcowy.

Testy akceptacyjne w praktyce

Co może być testem akceptacyjnym

To mogą być przykłady podane przez klienta, na przykład w postaci scenariusza określającego stan początkowy, akcję i stan końcowy.

Prosta akcja

Jeśli użytkownik systemu poda swoje dane i zaakceptuje regulamin, to mu się tworzy profil i widzi on inne profile.

Testy akceptacyjne w praktyce

Co może być testem akceptacyjnym

To mogą być przykłady podane przez klienta, na przykład w postaci scenariusza określającego stan początkowy, akcję i stan końcowy.

Prosta akcja

Jeśli użytkownik systemu poda swoje dane i zaakceptuje regulamin, to mu się tworzy profil i widzi on inne profile.

Inny przykład: System Zapisy

Jeśli użytkownik systemu kliknie na przedmiot, i liczba osób zapisanych na przedmiot jest mniejsza niż limit, to osoba zostaje zapisana na przedmiot.

Plan wykładu

TDD: Test–Driven Development

Jedna z koncepcji stosowana w programowaniu ekstremalnym (”najpierw testy”).

TDD: Test–Driven Development

Jedna z koncepcji stosowana w programowaniu ekstremalnym (”najpierw testy”).

albo

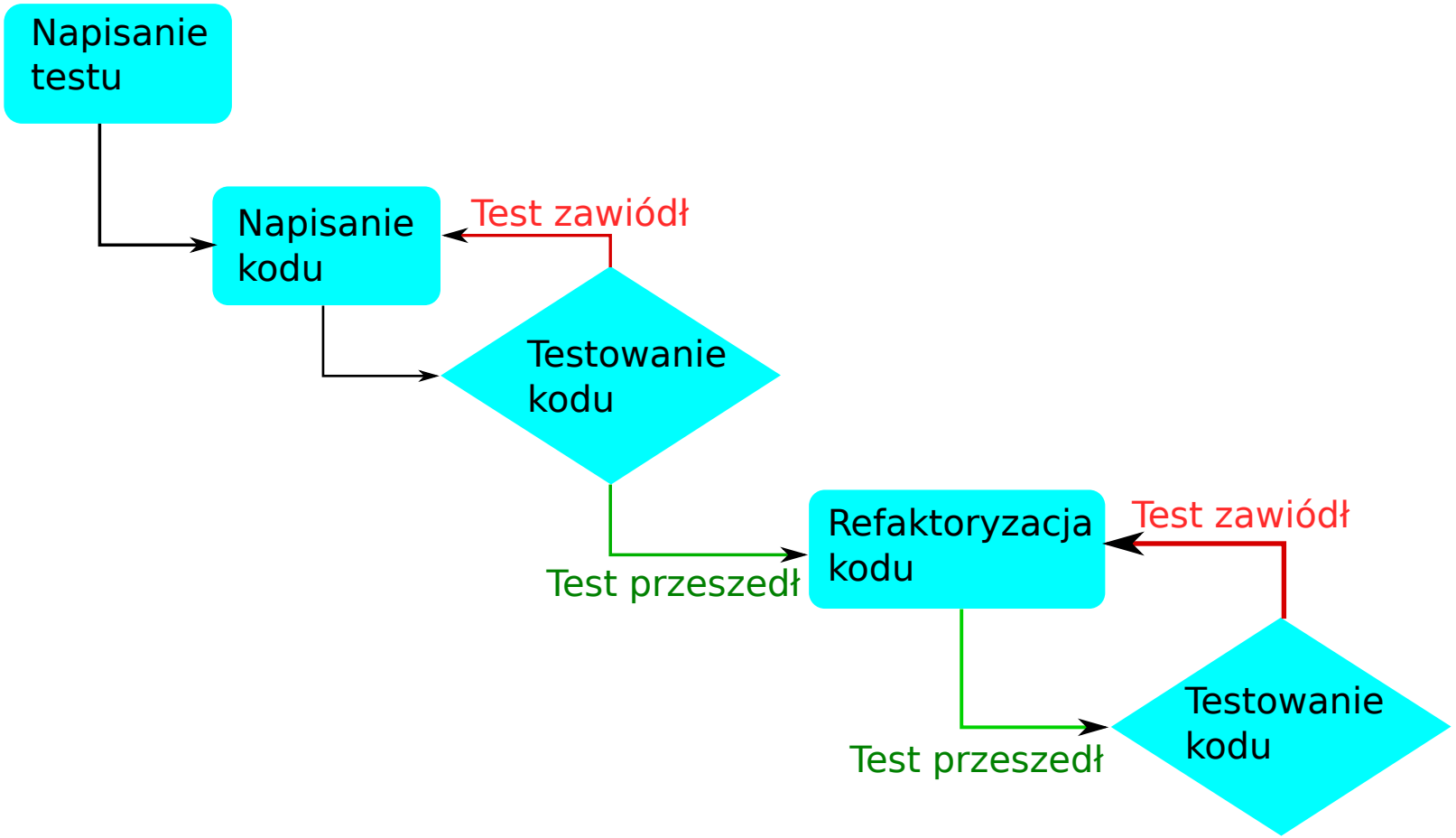
Samodzielna technika rozwoju oprogramowania.

Zasady TDD

Reguły:

- Najpiew pisany jest test jednostkowy, potem kod.
- Tworzone są tylko niezbędne testy.
- Kodu pisze się tylko tyle, ile jest niezbędne do przejścia testów.

Cykl pracy


Napisanie testu
Napisanie kodu
Test zawiódł
Testowanie kodu
Test przeszedł
Refaktoryzacja kodu
Test zawiódł
Test przeszedł
Testowanie kodu

Cykl pracy

Zakres prac w ramach cyklu

Pojedyncza funkcjonalność/funkcja

Czas pracy

Kilkanaście minut

Faza pierwsza

Dodanie testu

Utworzenie testu na podstawie opowieści użytkownika lub przypadku użycia.

Faza pierwsza

Dodanie testu

Utworzenie testu na podstawie opowieści użytkownika lub przypadku użycia.

Napisanie szkicu właściwej klasy/modułu tak, aby test się skompilował.

Faza pierwsza

Dodanie testu

Utworzenie testu na podstawie opowieści użytkownika lub przypadku użycia.

Napisanie szkicu właściwej klasy/modułu tak, aby test się skompilował.

Próba testu. Test powinien zawieść, co sugeruje, że test jest poprawny.

Faza druga

Testowanie testu

Uruchmienie wszystkich testów (również utworzonych wcześniej), i sprawdzenie czy nowy test zawiedzie.

Faza trzecia

Pisanie właściwego kodu

Kod nie musi być elegancki, ale powinien przechodzić testy. Kod jest poprawiany dopóki nie przejdzie swojego testu.

Faza czwarta

Uruchomienie wszystkich testów

Testowany jest cały program/moduł.

Faza piąta

Refaktoryzacja

Ulepszanie kodu i jego testowanie.

Przykład

Implementacja stosu

Wstępna implementacja stosu

```
[TestFixture()]
public class TestowanieStosu
{
    [Test()]
    public void TestCase()
    {
        Stos<int> s = new Stos<int>();
        Assert.AreEqual(s.isEmpty(), true);
        s.push(1024);
        Assert.AreEqual(s.pop(), 1024);
    }
}
```

Przykład

Szkielet implementacji stosu

Wstępna implementacja stosu

```
public class Stos<T> {
    public Stos() { }
    public bool isEmpty()
    {
        return false;
    }
    public void push(T val) { }
    public T pop()
    {
        return default(T);
    }
}
```

Red–green–refactor