Politechnika Wrocławska

Wprowadzenie do Behavior-driven development

Jakub Kosiński
Email: ja@ghandal.net

Czym jest BDD?

- praktyka, powstała na podstawie TDD, wykorzystywana w zwinnych metodykach
- stworzona przez Dana Northa w 2003 roku

Czym jest BDD?

„Behaviour-driven Development polega na tworzeniu oprogramowania przez opisywanie jego zachowania, z perspektywy jego udziałowców.”

[D. North]

Na co kładzie nacisk BDD?

- potrzeba zrozumienia potrzeb klienta i poznania jego języka i punktu widzenia – sposobu, w jaki opisuje to, jak oprogramowanie ma się ***zachowywać***

Na co kładzie nacisk BDD?

- potrzeba zrozumienia potrzeb klienta i poznania jego języka i punktu widzenia – sposobu, w jaki opisuje to, jak oprogramowanie ma się ***zachowywać***
- wielu „udziałowców”- patrzymy na projekt z perspektywy wszystkich ludzi zaangażowanych i zainteresowanych projektem

3 zasady BDD

1. Enough is enough
2. Deliver stakeholder value
3. It’s all behaviour

Enough is enough

- analizujemy, planujemy i projektujemy tyle, ile naprawdę trzeba
- nie tracimy czasu na na wyspecyfikowanie od razu całego zakresu projektu – wymagania i tak będą się pewnie zmieniać
- robimy tylko tyle, ile trzeba ☺

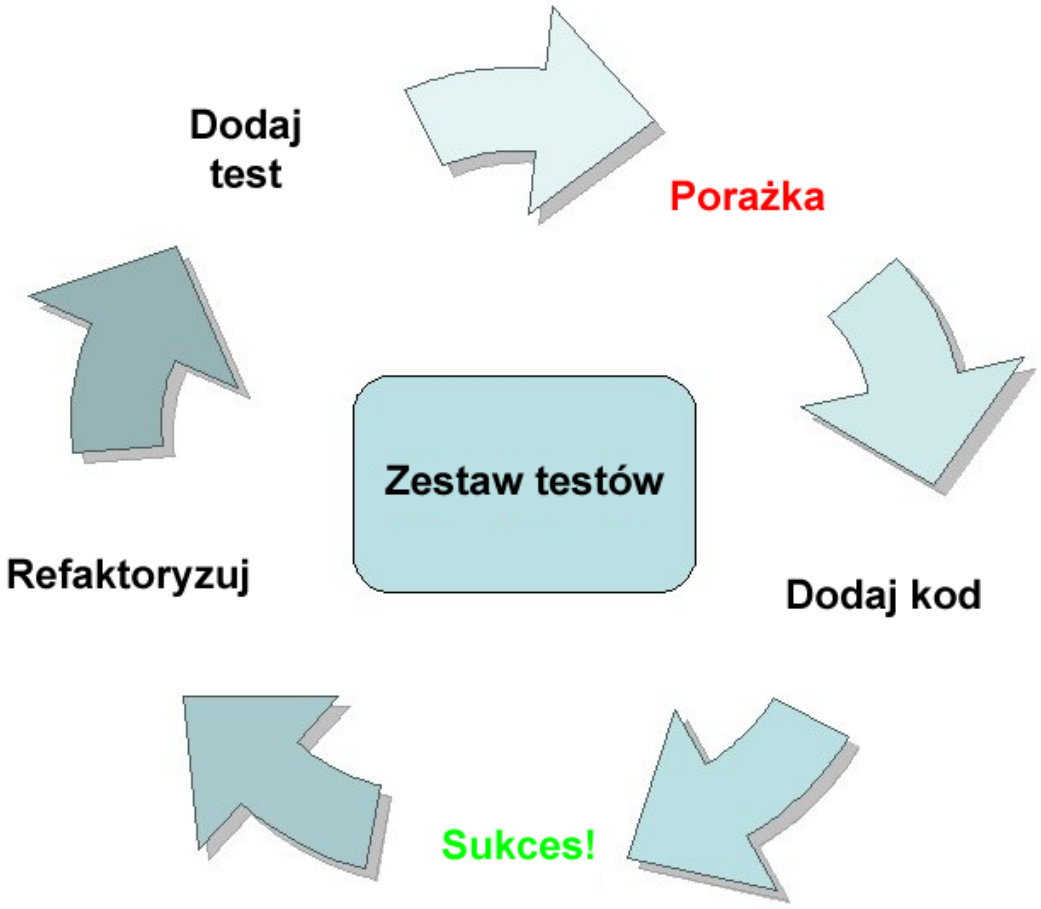
Deliver stakeholder value

- wszystko, co robimy ma nieść ze sobą realną wartość biznesową
- jeśli robimy coś, co tej wartości nie przynosi, warto zająć się czymś innym
- jeżeli funkcjonalność pojawia się w projekcie, to znaczy, że jest wartościowa

It’s all behaviour

- potrzeba mówienia „wszechobecnym językiem” (ang. *ubiquitous language*)
- wszyscy uczestnicy projektu powinni odwoływać i myśleć o systemie w ten sam sposób – opisując jego zachowanie tym samym słownictwem
- dzięki temu zmniejszana jest bariera komunikacyjna między „nietechnicznym” klientem, a „technicznymi” programistami
- wymagania definiujemy w formie historyjek użytkownika

BDD vs. TDD

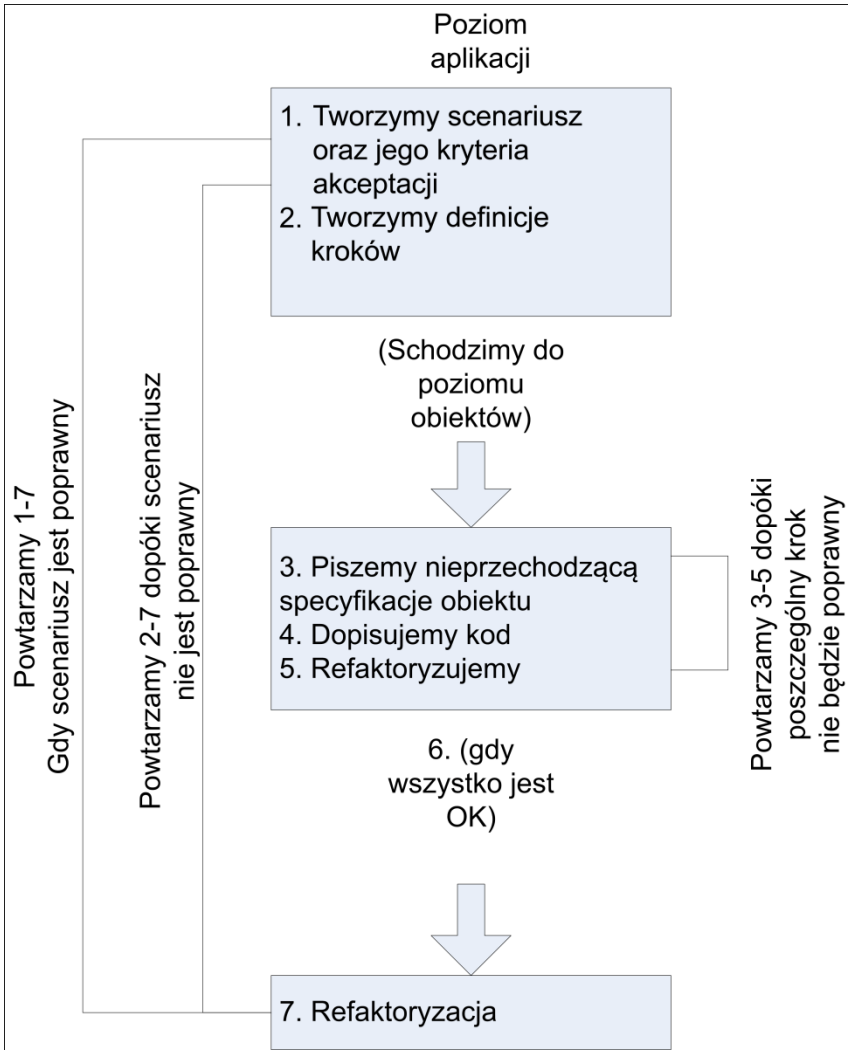

Dodaj test
Porażka
Zestaw testów
Dodaj kod
Sukces!
Refaktoryzuj

Cykl BDD

- Jeżeli skorzystamy z dwóch typów narzędzi:
  - „wysokopoziomowego” do specyfikowania na poziomie aplikacji
  - „niskopoziomowego” do specyfikowania na poziomie obiektów

  możemy nieco zmodyfikować nasz cykl tworzenia oprogramowania

Cykl BDD


Poziom aplikacji
1. Tworzymy scenariusz oraz jego kryteria akceptacji
2. Tworzymy definicje kroków
(Schodzimy do poziomu obiektów)
3. Piszemy nieprzechodzącą specyfikacje obiektu
4. Dopisujemy kod
5. Refaktoryzujemy
Powtarzamy 3-5 dopóki poszczególny krok nie będzie poprawny
6. (gdy wszystko jest OK)
7. Refaktoryzacja
Powtarzamy 2-7 dopóki scenariusz nie jest poprawny
Powtarzamy 1-7 Gdy scenariusz jest poprawny

Historyjki użytkownika

Historyjka jest spisanym wymaganiem klienta

Historyjki użytkownika

- Każda historyjka składa się z kilku części:
  - tytułu
  - narracji
  - kryteriów akceptacji

Historyjka użytkownika - narracja

- w narracji powinniśmy zawrzeć:
  - opis funkcjonalności
  - korzyści, jakie płyną z danej funkcjonalności
  - osobę (rolę), która będzie czerpać te korzyści

Historyjka użytkownika - narracja

Wypłata pieniędzy z bankomatu

Jako klient

Chcę wypłacić pieniądze z bankomatu

Aby nie musieć stać w kolejce do kasy

Historyjki użytkownika - kryteria akceptacji

- kryteria akceptacji określają moment, w którym historyjka jest kompletna
- spisywane są w postaci scenariuszy, których pozytywne przejście gwarantuje osiągnięcie celu

Historyjki użytkownika - kryteria akceptacji

- zwykle scenariusze składają się z trzech bloków:
  - **`Given`** - określający kontekst
  - **`When`** - określający zdarzenie
  - **`Then`** - określający rezultat

Historyjki użytkownika - kryteria akceptacji

Scenariusz: Konto posiada odpowiednią ilość środków

Zakładając, że na koncie jest odpowiednia ilość środków

Oraz, że karta jest poprawna

Oraz, że w kasecie są pieniądze

Jeżeli klient zażąda wypłaty gotówki

Wtedy konto zostanie obciążone

Oraz pieniądze zostaną wypłacone

Oraz karta zostanie zwrócona klientowi

Historyjki użytkownika

- kluczem do sukcesu jest spowodowanie, by kryteria akceptacji poszczególnych historyjek były wykonywalne
- dzięki temu będziemy mogli je zautomatyzować

Czym są specyfikacja obiektów?

- „testy” jednostkowe, pisane według pewnych konwencji:
  - nazwy metod testowych są zdaniami
  - nazwy metod korzystają z „wszechobecnego języka” – opisują zachowanie i są zrozumiałe dla wszystkich udziałowców
  - specyfikujemy to, co klasa ***powinna robić***

Specyfikacja obiektu - przykład

```
public class CustomerLookupBehavior {
   @Test
   public void shouldFindCustomerById() { … }

   @Test(expected=DuplicatedCustomerException.class)
   public void shouldFailForDuplicateCustomers() {…}
}
```

Co dają specyfikacja obiektów?

- jeśli obiekt nie przeszedł testu, mamy podpowiedź, dlaczego tak się stało
- pomagają w projektowaniu klas
- stanowią dokumentację klas
- są specyfikacjami wykonywalnymi

Jak weryfikować coś, czego nie ma?

Co zrobić, gdy testujemy funkcjonalność odwołującą się do części aplikacji, która nie została jeszcze zaimplementowana?

Mocks & stubs

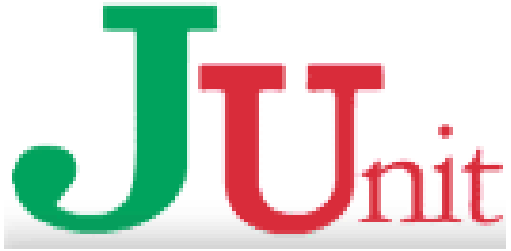
Stubs:

- Klasy, zawierające implementacje interfejsów

Mocks:

- Klasy, dla których możemy określić, jakie metody lub właściwości będą wykonywane, jakie wartości mają być przyjmowane i jakie zwracane

BDD w Java (JBehave + JUnit)



[Przykład dostępny w serwisie Github: http://github.com/ghandal/jbehave-example]

Zalety stosowania BDD

- Specyfikacje klas stają się testami i dokumentacją (specyfikacją wykonywalną)

Zalety stosowania BDD

- Specyfikacje klas stają się testami i dokumentacją (specyfikacją wykonywalną)
- Scenariusze stają się automatycznymi testami akceptacyjnymi, które stają się testami regresyjnymi oraz dokumentacją

Podsumowanie

- BDD jest techniką pisania oprogramowania, ale może być również traktowana jako pełna zwinna metodyka
- opiera się na trzech zasadach:
  - Enough is enough
  - Deliver stakeholder value
  - It’s all behaviour
- skupia się na polepszeniu komunikacji między poszczególnymi osobami zaangażowanymi w projekt

Literatura i zasoby

1. David Chelimsky, *The RSpec Book. Bahaviour Driven Development with RSpec, Cucumber and Friends*, The Pragmatic Programmers LLC., 2009.
2. Eric Evans, *Domain driven design*, 2002
3. Dan North, *Introducing BDD*, http://dannorth.net/introducing-bdd, 2006.
4. http://behaviour-driven.org

- JBehave – http://jbehave.org
- Przykłady - http://github.com/ghandal/jbehave-example