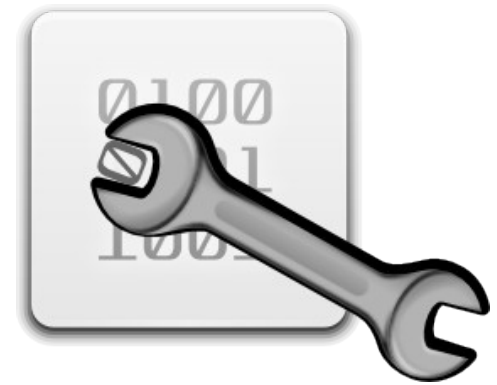


Podstawy inżynierii oprogramowania



Przegląd technik i narzędzi

Aleksander Lamża
ZKSB · Instytut Informatyki
Uniwersytet Śląski w Katowicach

aleksander.lamza@us.edu.pl

- TDD
- BDD
- CI



TDD

Test-Driven Development

czyli

wytwarzanie sterowane testami

Wywodzi się z metodyki **XP** (eXtreme Programming).



Główne założenia

Sporządzanie testów przed napisaniem kodu.

Napisanie jakiegokolwiek kawałka kodu musi być wymuszone istnieniem testu kończącego się błędem.

Cykl TDD

Etap czerwony: testy kończą się niepowodzeniem.

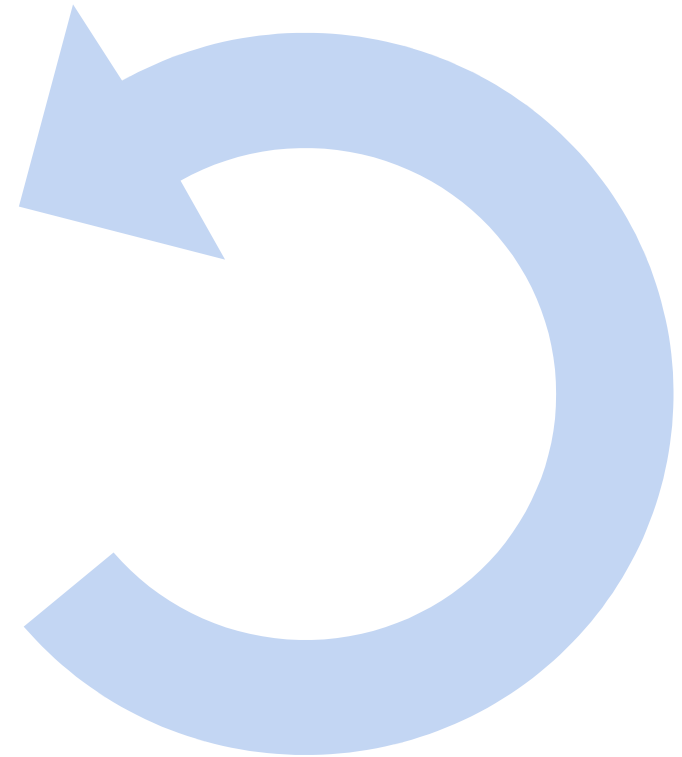
Najpierw napisz test sprawdzający funkcję, którą chcesz dodać. Test zakończy się niepowodzeniem, ponieważ nie ma kodu.

Etap zielony: kod przechodzi testy.

Dodaj najprostszy kod umożliwiający udane zakończenie testu.

Etap refaktoryzacji

Porządkowanie kodu: usuwanie powtórzeń, nieeleganckich rozwiązań, zbędnego kodu itp.





BDD

Behavior-Driven Development

czyli

wytwarzanie sterowane zachowaniami

Ta technika powstała na bazie TDD.



Podstawowe założenia i zasady

Testy mają być pisane przez programistów (jak w TDD),
ale przy **aktywnym współudziale klienta**. ← Testy akceptacyjne!

Scenariusze buduje się według zasady **G W T**

G jak **Given**

Przedstawienie sytuacji przed wykonaniem
(jak się zaczyna historia?)

W jak **When**

Opis tego, co wykonujemy

T jak **Then**

Opis uzyskiwanego efektu

Można też stosować zapis zgodny z RSpec: **describe – it**

Przykład testu we frameworku easyb

easyb.org



```
scenario "null is pushed onto empty stack", {  
  given "an empty stack", {  
    stack = new Stack()  
  }  
  
  when "null is pushed", {  
    pushnull = {  
      stack.push(null)  
    }  
  }  
  
  then "an exception should be thrown", {  
    ensureThrows(RuntimeException) {  
      pushnull()  
    }  
  }  
  
  and "then the stack should still be empty", {  
    stack.empty.shouldBe true  
  }  
}
```

Opowieści (ang. stories) zapisane
zgodnie z notacją GWT



Easyb – zapis specyfikacji w notacji RSpec

```
before "initialize the queue for each spec", {  
    queue = new Queue()  
}  
  
it "should dequeue gives item just enqueued", {  
    queue.enqueue(2)  
    queue.dequeue().shouldBe(2)  
}  
  
it "should throw an exception when null is enqueued", {  
    ensureThrows(RuntimeException) {  
        queue.enqueue(null)  
    }  
}
```

Inny przykład – framework Jasmine (JavaScript)

<http://pivotal.github.com/jasmine/>



```
describe("Koszyk", function() {
  var cart;
  var product1;
  var product2;

  beforeEach(function() {
    cart = new Cart();
    product1 = new Product("chleb", 3.50);
  });

  it("powinien przyjąć nowy produkt", function() {
    cart.addProduct(product1);
    expect(cart.productsCount()).toEqual(1);
  });
  ...
  describe("kiedy jest pusty", function() {
    it("powinien rzucić wyjątek 'shopping cart is empty' przy próbie przejścia do kasy", function() {
      cart.removeAll();

      expect(function() {
        cart.checkout();
      }).toThrow("shopping cart is empty");
    });
  });
});
```



CI

Continuous Integration

czyli

ciągła integracja

To również wywodzi się z metodyki XP



Podstawowym celem CI jest

zapewnienie możliwości wydania stabilnej wersji oprogramowania w dowolnym momencie i zminimalizowanie ryzyka wystąpienia błędów.

Jak to osiągnąć?

- Każdy dodawany kod musi być uzupełniony testami (jednostkowymi).
- Kod musi być trzymany w repozytorium.
- Trzeba zautomatyzować wszystko, co się da (testowanie, kompilowanie, budowanie itd.).

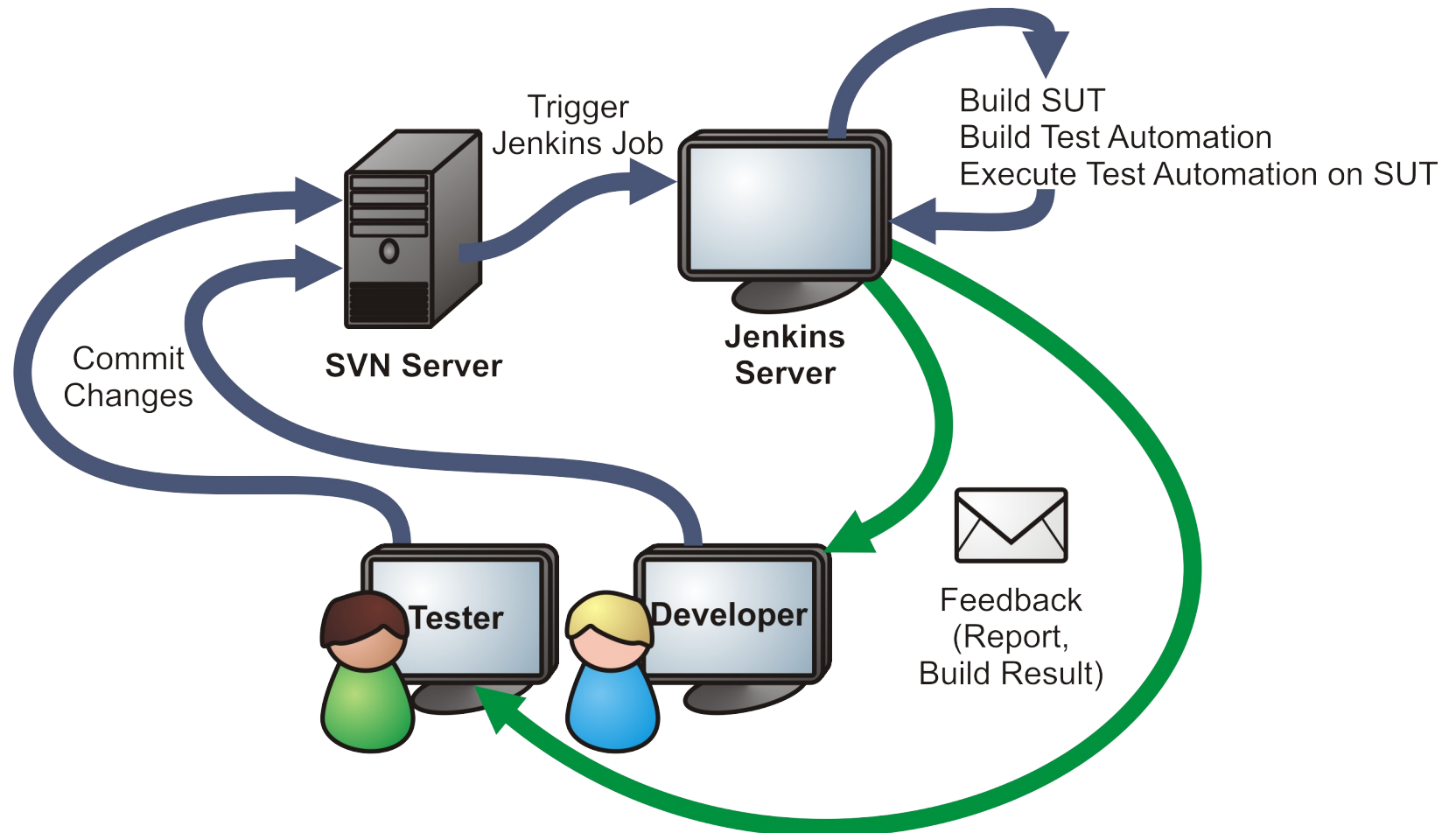
Przykładowe narzędzia CI

- CruiseControl
- Microsoft Team Foundation Server
- JetBrains TeamCity
- IBM Rational Team Concert
- Hudson/Jenkins
- ...

Jak to działa?



Jenkins



Żeby móc się nazwać dobrym programistą,
sama umiejętność programowania nie wystarcza!

**Trzeba jeszcze znać (i stosować!)
techniki oraz narzędzia
wspomagające pracę.**