



Politechnika Wrocławska

Testowanie oprogramowania

Marian Jureczko



Agenda

- Testy jednostkowe
 - JUnit
 - Mock Objects, EasyMock
- Programowanie przez testy (TDD)
- Testy akceptacyjne

Literatura

- Kent Beck: Test-Driven Development by example.
- Kent Beck, Cynthia Andres: Extreme programming.
- Andrew Hunt: Pragmatic unit testing in Java with JUnit
- Lasse Koskela: Test Driven: Practical TDD and Acceptance TDD for Java Developers.
- Lech Madeyski: Test-first programming Experimentation and Meta-Analysis.
- Rick Mugridge: Fit for developing software: framework for integrated tests.
- Glenford J. Myers: Art of software testing.
- Joseph Bergin: *XP Testing a GUI with FIT, Fitnesse, and Abbot*
- Filippo Ricca: *Automatic Acceptance Testing with FIT/FitNesse*



Testy jednostkowe

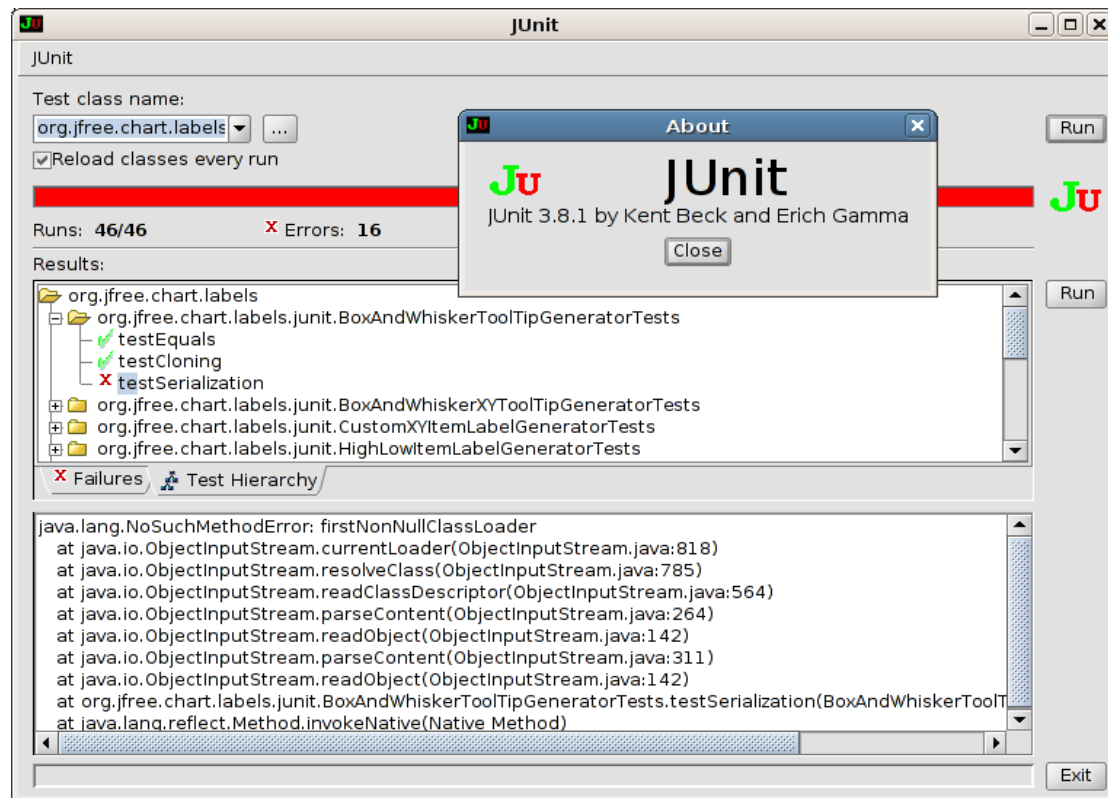
<http://pl.wikipedia.org>

Test jednostkowy (ang. *unit test*, test modułowy) to w programowaniu **metoda testowania** tworzonego oprogramowania poprzez wykonywanie testów weryfikujących poprawność działania **pojedynczych elementów** (jednostek) programu - np. metod lub obiektów w programowaniu obiektowym lub procedur w programowaniu proceduralnym. Testowany fragment programu poddawany jest testowi, który wykonuje go i porównuje wynik (np. zwrócone wartości, stan obiektu, wyrzucone wyjątki) z oczekiwanymi wynikami - **tak pozytywnymi, jak i negatywnymi** (niepowodzenie działania kodu w określonych sytuacjach również może podlegać testowaniu).



JUnit

- Twórcy: K.Beck, E.Gamma
- <http://junit.org>
- Wsparcie dla wielu języków:
 - SUnit (Smalltalk)
 - NUnit (C#)
 - PyUnit (Python)
 - CPPUnit (C++)
 - fUnit (Fortran)
 - JSUnit (JavaScript)





Najprostsze testy

Andrew Hunt: Pragmatic unit testing in Java with JUnit

```
public class Largest{
    public static int largest( int[] list ){
        int i, max=Integer.MAX_VALUE;
        for( i=0; i<list.length-1; i++ ){
            if( list[i]>max ) {
                max = list[i];
            }
        }
        return max;
    }
}
```



Najprostsze testy

```
import junit.framework.*;

public class TestLargest extends TestCase{
    public TestLargest(String name) {
        super(name) ;
    }
    public void testOrder() {
        assertEquals(9,Largest.largest(new int[]
            {8,9,7}));
    }
}
```



Najprostsze testy

There was 1 failure:

```
1) testOrder(TestLargest) junit.framework.AssertionFailed Error: expected<9> but was:<2147483647> at TestLargest.testOrder(TestLargest.java:7)
```




Najprostsze testy

```
public class Largest{  
    public static int largest( int[] list ){  
        int i, max=Integer.MAX_VALUE; //max=0  
        for( i=0; i<list.length-1; i++ ){  
            if( list[i]>max ) {  
                max = list[i];  
            }  
        }  
        return max;  
    }  
}
```



Najprostsze testy

```
public void testOrder() {  
    assertEquals(9, Largest.largest(new int[]  
        {9, 8, 7}));  
    assertEquals(9, Largest.largest(new int[]  
        {8, 9, 7}));  
    assertEquals(9, Largest.largest(new int[]  
        {7, 8, 9}));  
}
```



Najprostsze testy

There was 1 failure:

```
1) testOrder(TestLargest) junit.framework.AssertionFailed Error: expected<9> but was:<8> at TestLargest.testOrder(TestLargest.java:9)
```



Najprostsze testy

```
public class Largest{  
    public static int largest( int[] list ){  
        int i, max=0;  
        for( i=0; i<list.length-1; i++ ){  
            //i<list.length  
            if( list[i]>max ) {  
                max = list[i];  
            }  
        }  
        return max;  
    }  
}
```



Najprostsze testy

```
public void testOrder() {  
    assertEquals(9, Largest.largest(new int[]  
        {9,8,9,7}));  
    assertEquals(1, Largest.largest(new int[]  
        {1}));  
    assertEquals(-7, Largest.largest(new int[]  
        {-7,-8,-9}));  
}
```



Najprostsze testy

There was 1 failure:

```
1) testOrder(TestLargest) junit.framework.AssertionFailed Error: expected<-7> but was:<0> at TestLargest.testOrder(TestLargest.java:12)
```



Najprostsze testy

```
public class Largest{
    public static int largest( int[] list ){
        int i, max=0; //max=Integer.MIN_VALUE
        for( i=0; i<list.length; i++ ){
            if( list[i]>max ) {
                max = list[i];
            }
        }
        return max;
    }
}
```

Wykonywanie testów (konsola)

Kompilacja programu i testów:

```
javac Largest.java TestLargest.java
```

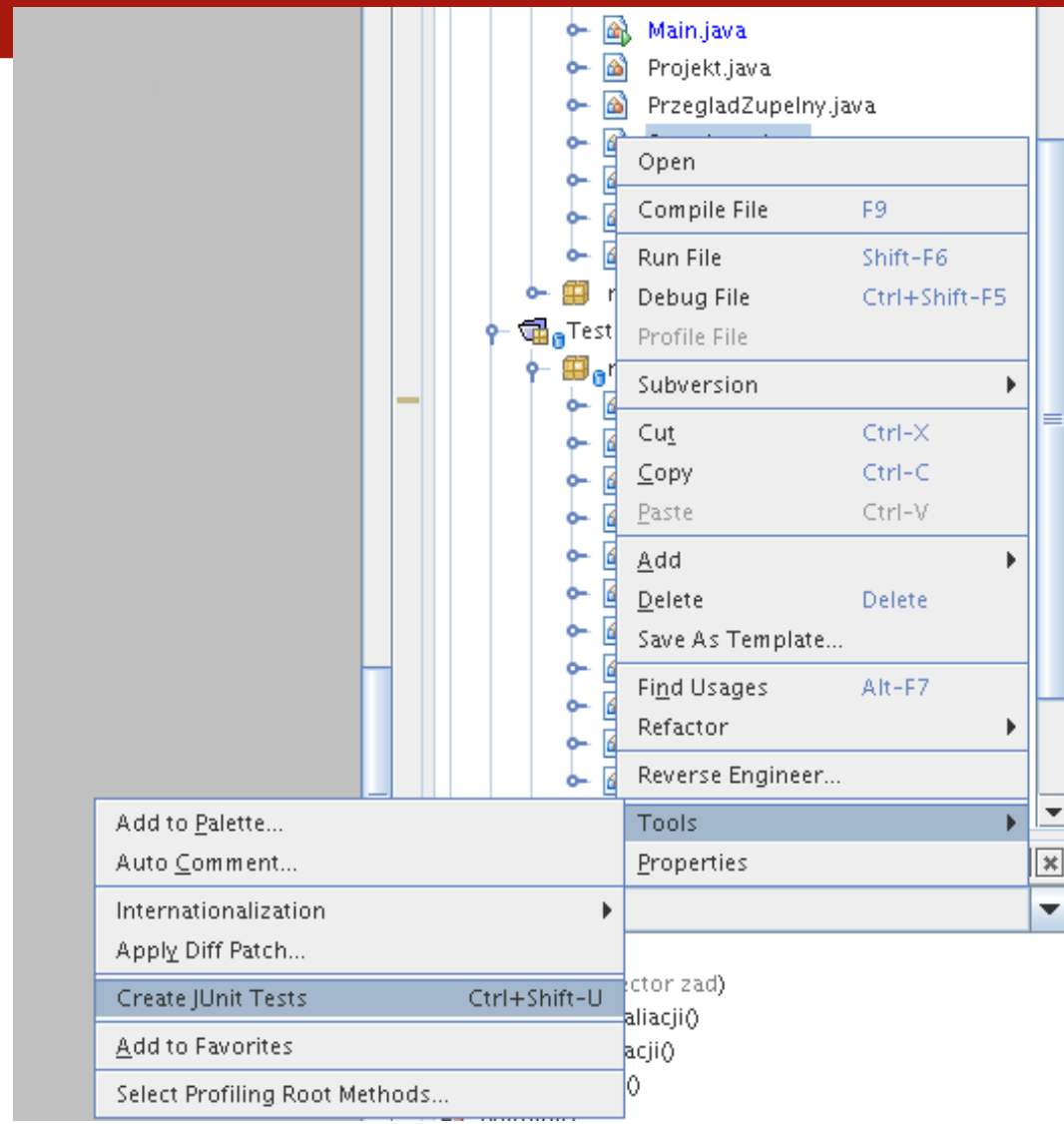
Uruchomienie testów:

```
java junit.textui.TestRunner TestLargest
```



Wykonywanie testów (NetBeans)

- Tworzenie nowego testu





Wykonywanie testów (NetBeans)

- Parametry nowego testu



Create Tests [X]

Class to Test: org.openoffice.comp.ModelKreator.OOoWrapper

Class Name:

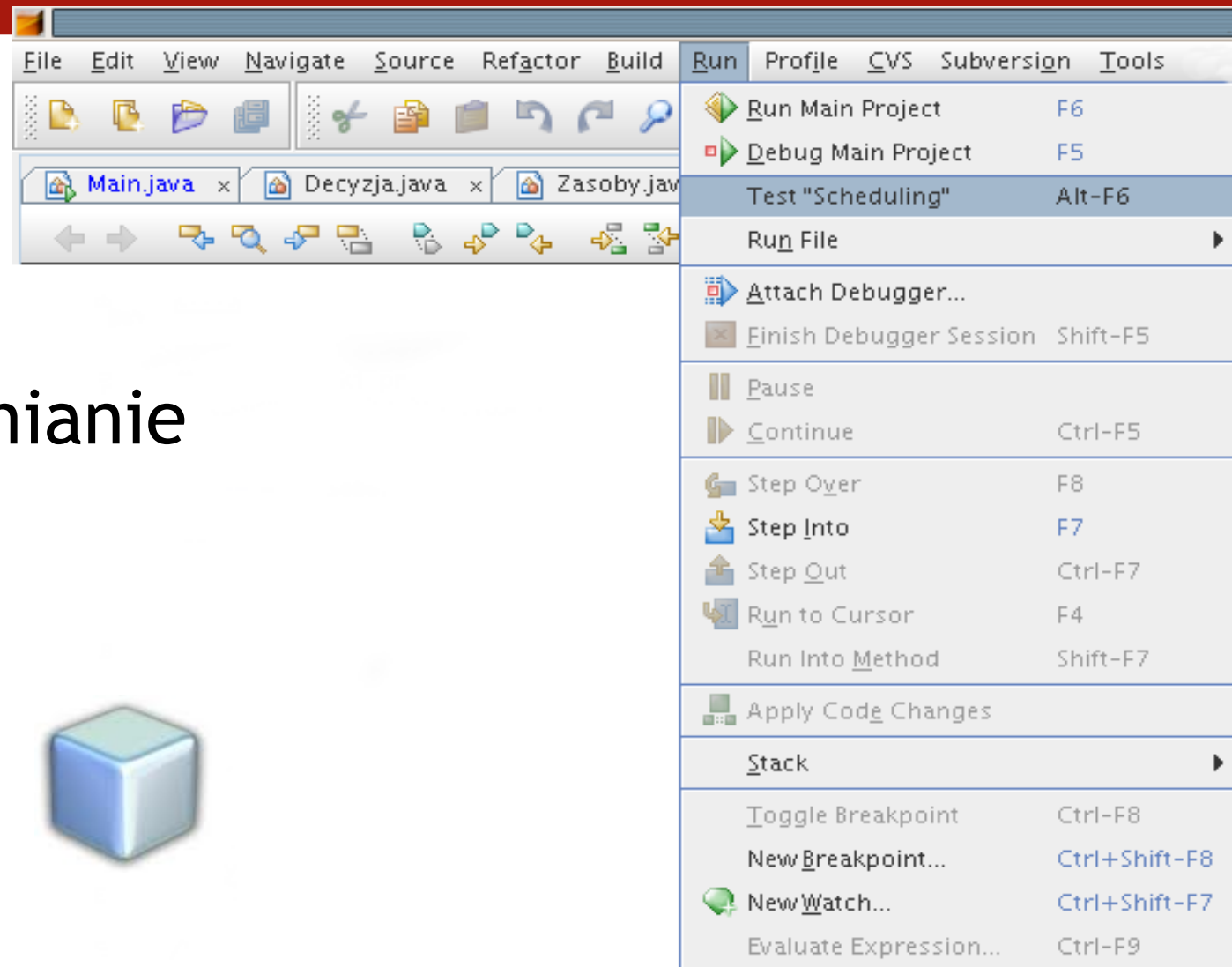
Location: ▼

Code Generation

Method Access Levels	Optional Code
<input checked="" type="checkbox"/> <u>P</u> ublic	<input checked="" type="checkbox"/> set <u>U</u> p
<input checked="" type="checkbox"/> <u>P</u> rotected	<input checked="" type="checkbox"/> tear <u>D</u> own
<input checked="" type="checkbox"/> <u>P</u> ackage Private	<input type="checkbox"/> Default <u>M</u> ethod Bodies
	Optional Comments
	<input checked="" type="checkbox"/> <u>J</u> avadoc Comments

[OK] [Cancel] [Help]

Wykonywanie testów (NetBeans)



- Uruchamianie testów





Struktura testów jednostkowych

- Dziedziczenie po klasie TestCase

- Adnotacja @Test

@Test

```
public void method() {...}
```

- Konwencja nazewnictwa

```
public void testMethod() {...}
```

Asercje JUnit (Klasa Assert)

- `assertEquals ([msg] , expected , actual)`
- `assertEquals ([msg] , expected , actual , tolerance)`
- `assertArrayEquals ([msg] , expected[] , actual[])`
- `assertNull ([msg] , object)`
- `assertNotNull ([msg] , object)`
- `assertSame ([msg] , expected , actual)`
- `assertNotSame ([msg] , expected , actual)`
- `assertTrue ([msg] , condition)`
- `assertFalse ([msg] , condition)`
- `assertThat ([msg] , actual , matcher)`
- `fail ([msg])`



Konfiguracja testu

```
public class TestDB extends TestCase{  
    private Connection dbConn;  
    @Before  
    protected void setUp() {  
        dbCon = new Connection(...);  
        dbConn.connect();  
    }  
    @After  
    protected void tearDown() {  
        dbConn.disconnect();  
    }  
    public void test1() {...}  
}
```



Konfiguracja testu

```
public class TestDB extends TestCase{  
    private Connection dbConn;  
    @BeforeClass  
    protected void setUpClass() {  
        dbCon = new Connection(...);  
        dbConn.connect();  
    }  
    @AfterClass  
    protected void tearDownClass() {  
        dbConn.disconnect();  
    }  
    public void test1() {...}  
}
```



Wyjątki

```
public void testException() {  
    try{  
        sortMyList(null);  
        fail("Metoda powinna wygenerować  
            wyjątek");  
    }catch (RuntimeException e) {  
        assertTrue(true);  
    }  
}
```

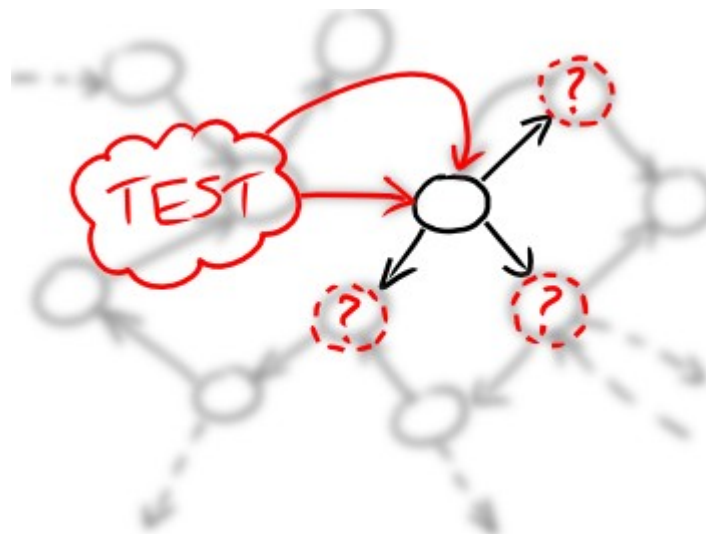



Efektywność

```
public void testSym()  
{  
    long start,end;  
    Symulator s=new Symulator();  
    start=Calendar.getInstance().getTimeInMillis();  
    s.sym();  
    end=Calendar.getInstance().getTimeInMillis();  
    assertTrue( end-start < 1000 );  
}
```

Obiekty imitacji (Mock Objects)

Obiekty imitacji zastępują rzeczywisty obiekt na czas uruchamiania i testowania kodu.



<http://sourceforge.net/projects/mockobjects/>

<http://www.mockobjects.com/>

Obiekty imitacji (Mock Objects)

Zastosowania:

- Nie chcemy aby obiekt rz. Brał udział w teście
- Obiekt rz. zachowuje się niedeterministycznie
- Obiekt rz. jest trudny do skonfigurowania
- Trudno jest wywołać interesujące nas zachowanie obiektu (np. błąd sieci)
- Obiekt rz. działa powoli
- Obiekt rz. ma interfejs użytkownika
- Obiekt rz. jeszcze nie istnieje



Obiekty imitacji (mockobjects) - testowanie serwletu

```
public void doGet(HttpServletRequest req,  
    HttpServletResponse res)  
{  
    String s = req.getParameter("cal");  
    res.setContentType("text/html");  
    PrintWriter out = res.getWriter();  
    double cal = Double.parseDouble(s);  
    double joule = 4.1868*cal;  
    out.println( String.valueOf(joule) );  
}
```



Obiekty imitacji (mockobjects) - testowanie serwetu

```
import junit.framework.*;
import com.mockobjects.servlet.*;

public class TestServlet extends TestCase {
    public void test1(){
        Cal2JServlet s = new Cal2JServlet();
        MockHttpServletRequest req=new MockHttpServletRequest();
        MockHttpServletResponse res=new MockHttpServletResponse();
        req.setupAddParameter("cal", "1");
        res.setExpectedContentType("text/html");
        s.doGet( req, res );
        double j = Double.parseDouble(res.
            GetOutputgetOutputStreamContents());
        assertEquals( 4.1868, j, 0.01 );
    }
}
```

Obiekty imitacji (Easy Mock)



- Tryb nagrywania
 - Wołamy wymagane metody
 - Konfigurujemy zwracane wartości
- Tryb odtwarzania
 - Można wołać „nagrane” wcześniej metody
 - Otrzymuje się skonfigurowane wcześniej wartości

<http://easymock.org/>

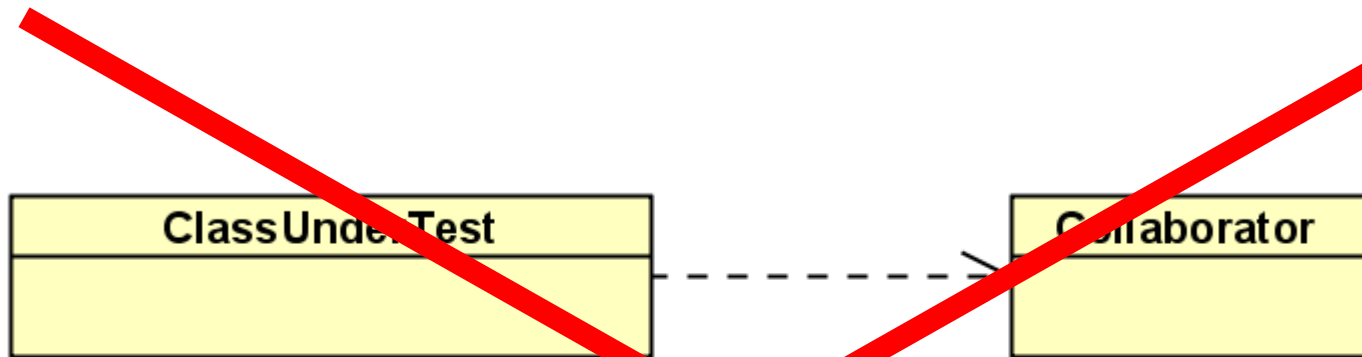


Obiekty imitacji (Easy Mock)

```
public class ClassUnderTest{  
    public void addDoc(String title, ICollaborator c){  
        c.docAdded(title);  
    }  
}
```

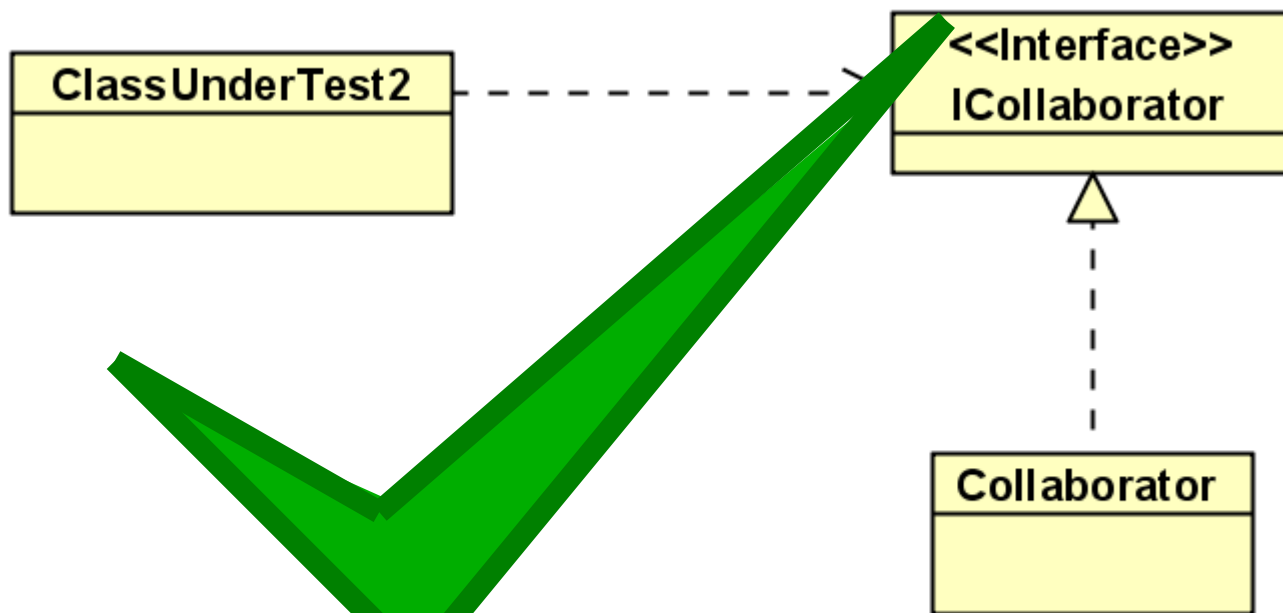
```
public void testAddDocument() {  
    ICollaborator mock = createMock(ICollaborator.class);  
    mock.docAdded("Document");  
    replay(mock);  
    classUnderTest.addDoc("Document");  
    verify(mock);  
}
```

Testowalna architektura (Design for Testability)



```
public class ClassUnderTest{
    void method(){
        Collaborator c = new Collaborator();
        c.doSomething();
    }
}
```


Testowalna architektura (Design for Testability)



```
public class ClassUnderTest{
    void method(ICollaborator c){
        ...
    }
}
```



Cechy poprawnych testów jednostkowych

- **Automatyzacja** - uruchamianie testów musi być łatwe.
- **Kompletność** - należy testować wszystko co może zawieść.
- **Powtarzalność** - wielokrotne wykonanie testu daje te same wyniki.
- **Niezależność** - od środowiska i innych testów.
- **Profesjonalizm** - kod testujący jest tak samo ważny jak kod dostarczany klientowi.

Co testować?

- Czy wyniki są poprawne (klasy ekwiwalencji)?
- Czy warunki brzegowe zostały prawidłowo określone?
- Czy można sprawdzić relacje zachodzące w odwrotnym kierunku?
- Czy można sprawdzić wyniki w alternatywny sposób?
- Czy można wymusić błędy?
- Czy efektywność jest zadowalająca?

Log4j

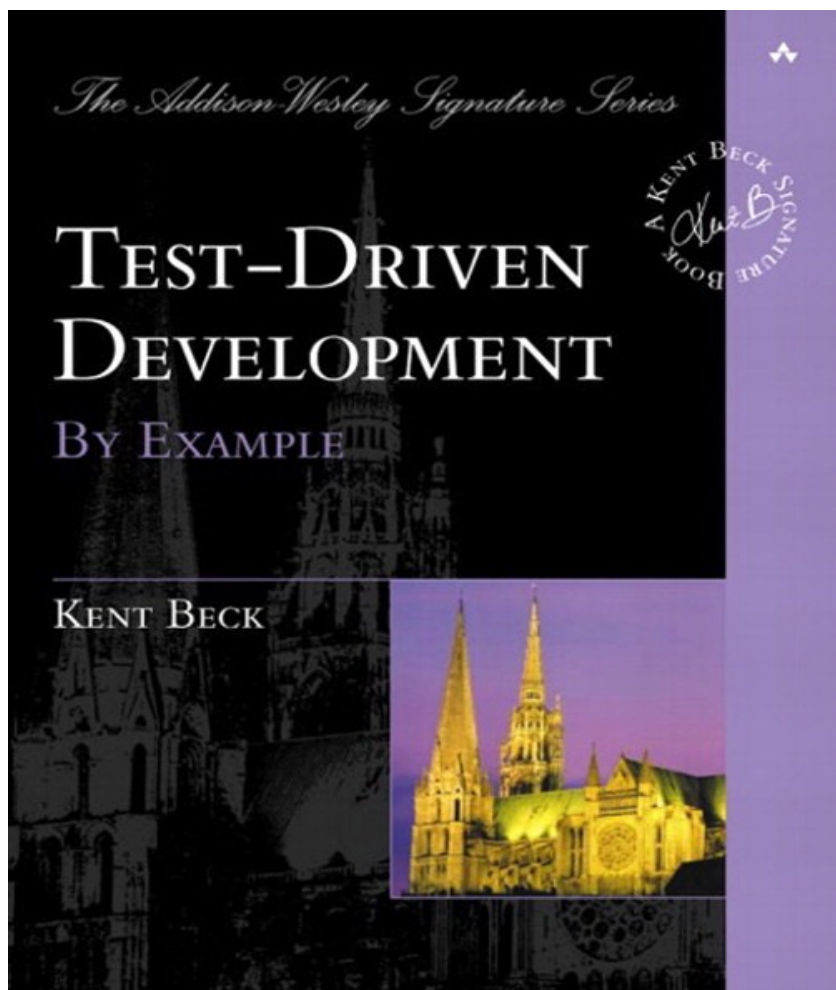
- Zalety
 - Łatwa i szybka implementacja.
 - Możliwość testowania w dowolnym miejscu kodu.
 - Przyspiesza (zastępuje) debugowanie.
- Wady
 - Kod logowania wymieszany z kodem dostarczany klientowi.



Log4j

```
import org.apache.log4j.*  
  
...  
BasicConfigurator.configure();  
Logger log=Logger.getLogger („name”);  
log.setLevel( Level.WARN );  
log.debug („...”);  
log.info („...”);  
log.warn („...”);  
log.error („...”);  
log.fatal („...”);
```

Programowanie przez testy (Test Driven Development)



Programowanie przez testy to sposób programowania (nie testowania), w którym wpierw piszemy testy, a dopiero później właściwy program



Programowanie przez testy

Red - Green - Refactor

Tworzenie nowego
nieprzechodzącego
testu dla nowej
funkcjonalności

Red

Pisanie
kodu produkcyjnego
- dokładnie tyle, ile
potrzeba by
zadziałał test

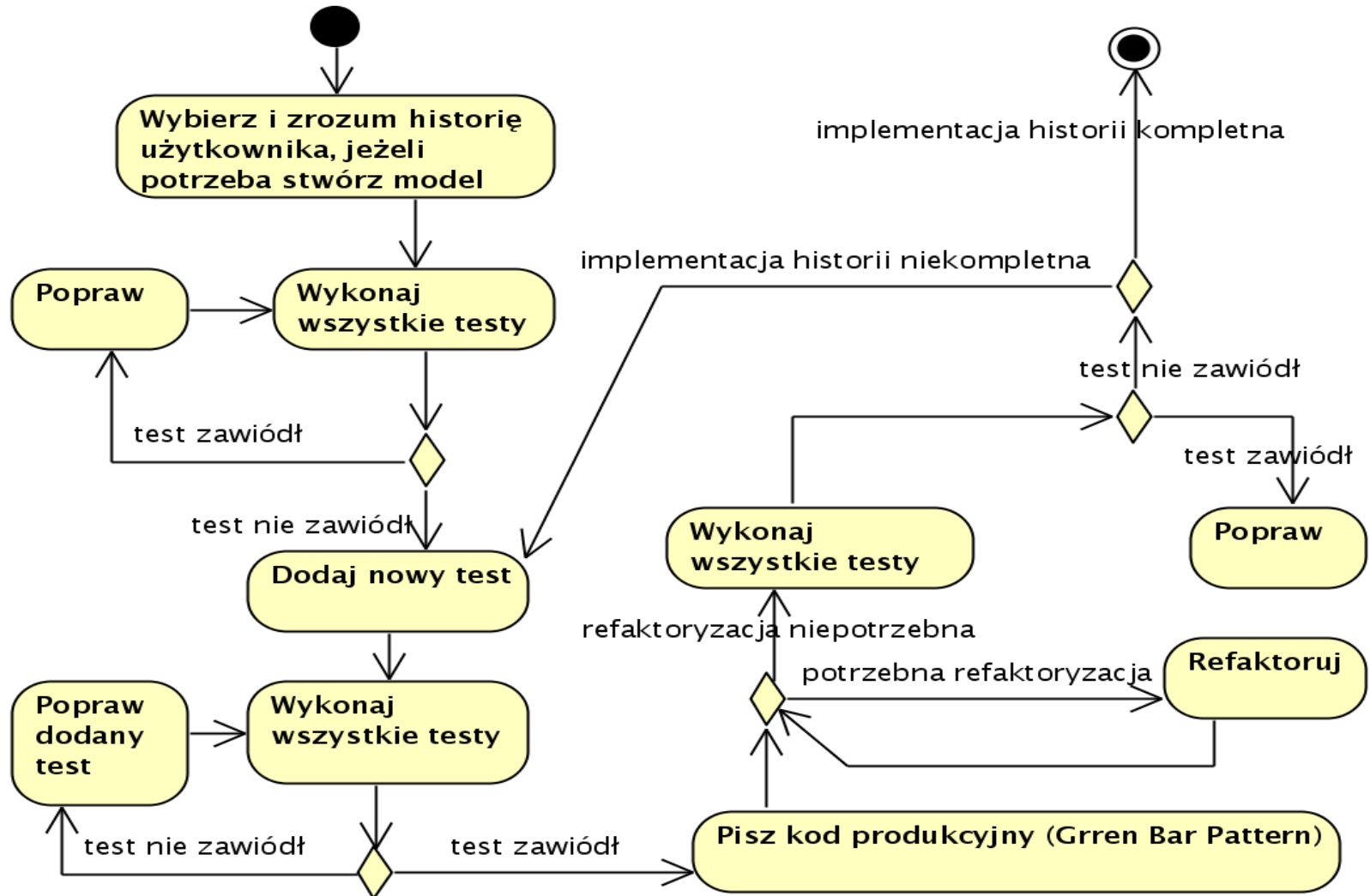
Refactor

Green

Refaktoryzacja, zrefaktoryzowany kod
musi przechodzić przez wszystkie testy.
Brak nowej funkcjonalności.

Red-Green-Refactor na przykładzie programowania ekstremalnego (XP)

Lech Madeyski: Test-first programming Experimentation and Meta-Analysis.





Wzorce tworzenia testu (Red Bar Patterns)

- Test początkowy (Starter Test)
- Test wyjaśniający (Explanation Test)
- Test poznawczy (Learning Test)
- Kolejny test (Another Test)
- Test regresyjny (Regression Test)

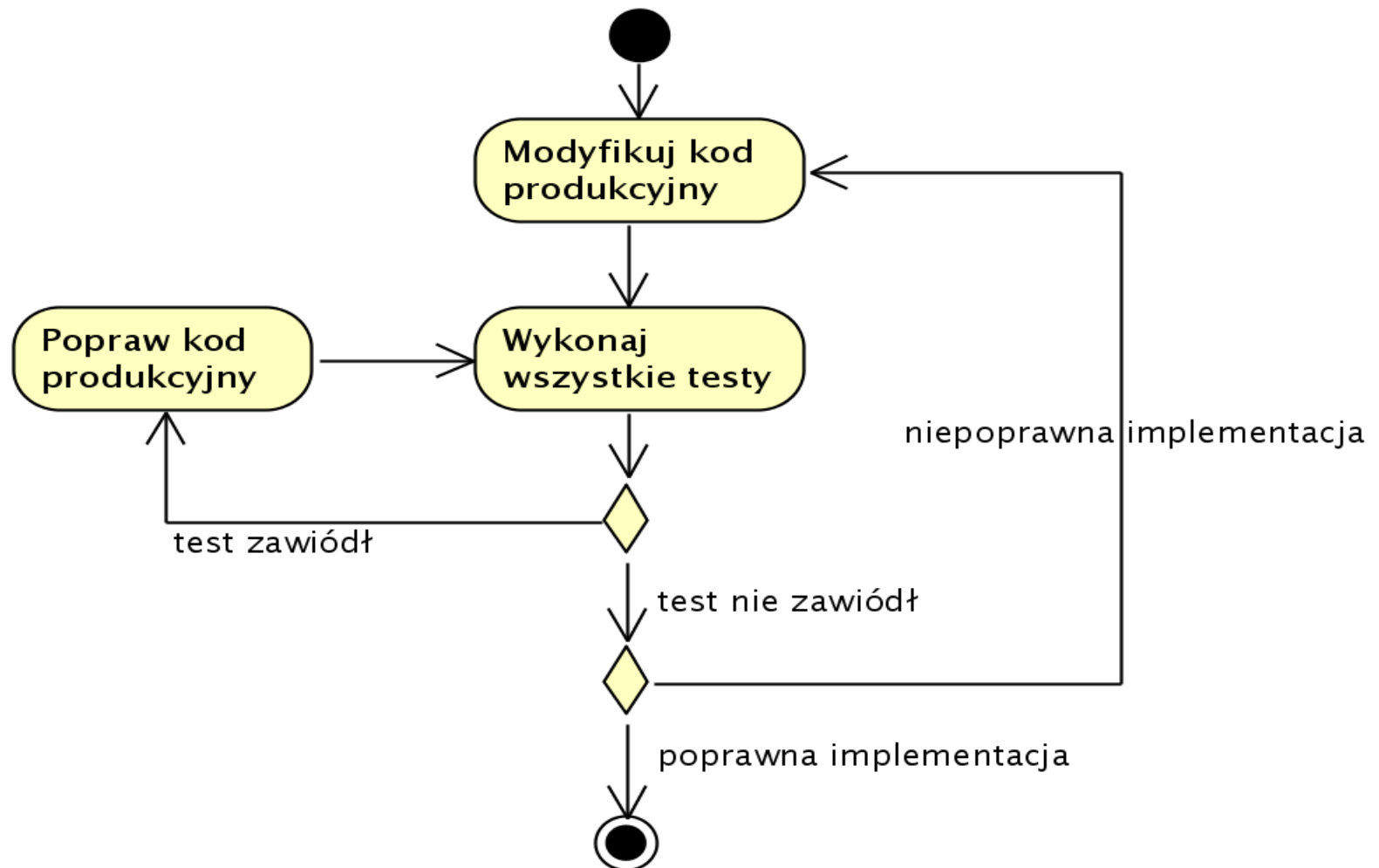


Wzorce tworzenia kodu (Green Bar Patterns)

- Fałszywa implementacja (Fake It 'Til You Make It)
- Triangulacja (Triangulate)
- Oczywista implementacja (Obvious Implementation)

Fałszywa implementacja (Fake It)

Lech Madeyski: Test-first programming Experimentation and Meta-Analysis.





Fałszywa implementacja (Fake It)

Kent Beck: Test-Driven Development by example

```
assertEquals(new MyDate(„28.2.02”,  
    new MyDate(„1.3.02”).yesterday());
```

```
public MyDate yesterday() {  
    return new MyDate(„28.2.02”);  
}
```



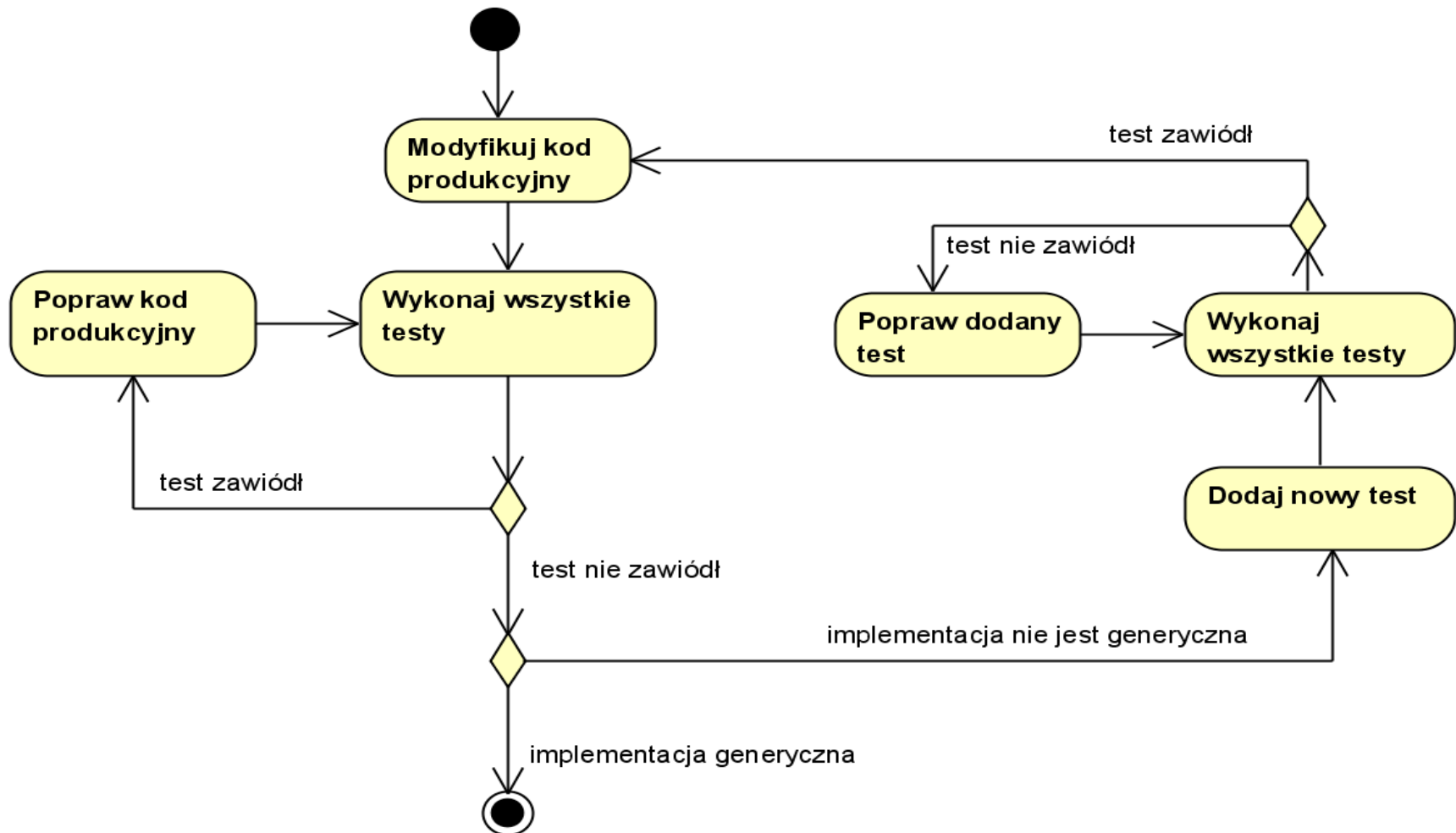
```
public MyDate yesterday() {  
    return new MyDate(new MyDate(„1.3.02”).days()-1);  
}
```



```
public MyDate yesterday() {  
    return new MyDate(this.days()-1);  
}
```

Triangulacja (Trinagulate)

Lech Madeyski: Test-first programming Experimentation and Meta-Analysis.

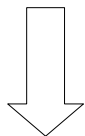




Triangulacja (Trinagulate)

Kent Beck: Test-Driven Development by example

```
public void testSum(){  
    assertEquals(4,plus(3,1));  
}
```



```
public void testSum(){  
    assertEquals(4,plus(3,1));  
    assertEquals(7,plus(3,4));  
}
```

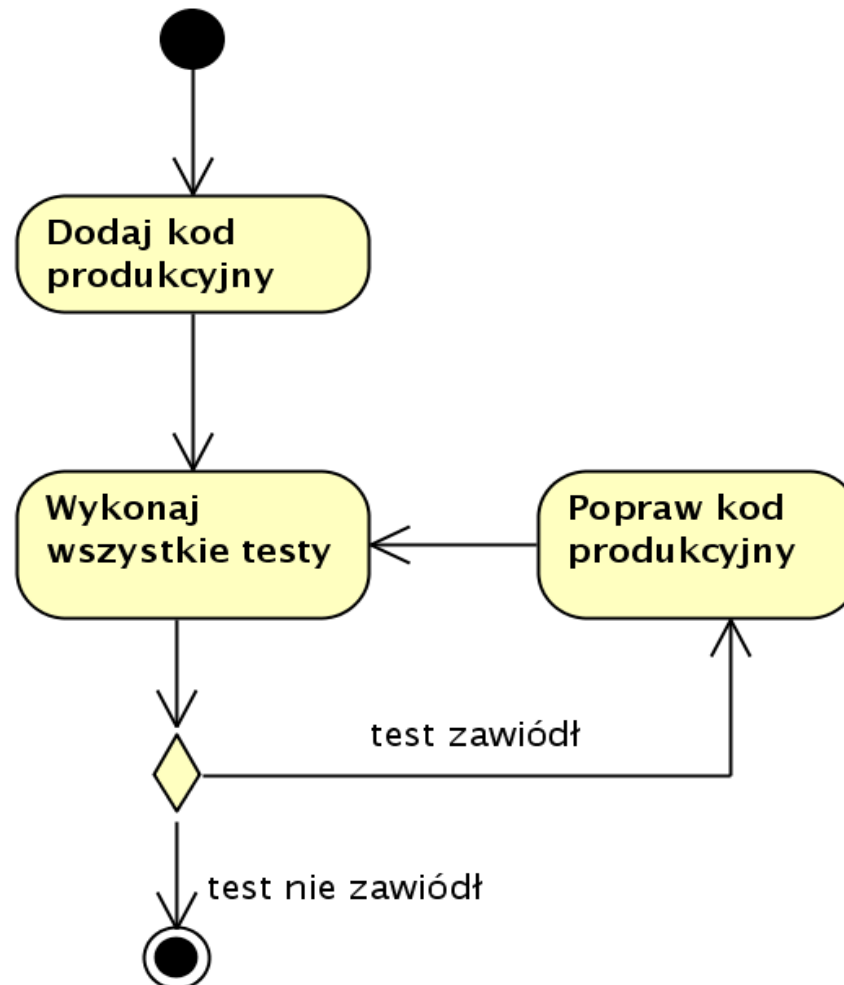
```
private int plus(int a, int b) {  
    return 4;  
}
```



```
private int plus(int a, int b) {  
    return a+b;  
}
```

Oczywista implementacja (Obvious Implementation)

Lech Madeyski: Test-first programming Experimentation and Meta-Analysis.





Oczywista implementacja (Obvious Implementation)

Kent Beck: Test-Driven Development by example

```
public void testSum(){  
    assertEquals(4,plus(3,1));  
}
```

```
private int plus(int a, int b) {  
    return a+b;  
}
```




Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

Dostawca	Udział	Cena	Suma
IBM	1000	25 USD	25000 USD
GE	400	150CHF	60000 CHF
		Suma	65000 USD

Z	Na	Kurs
CHF	USD	2

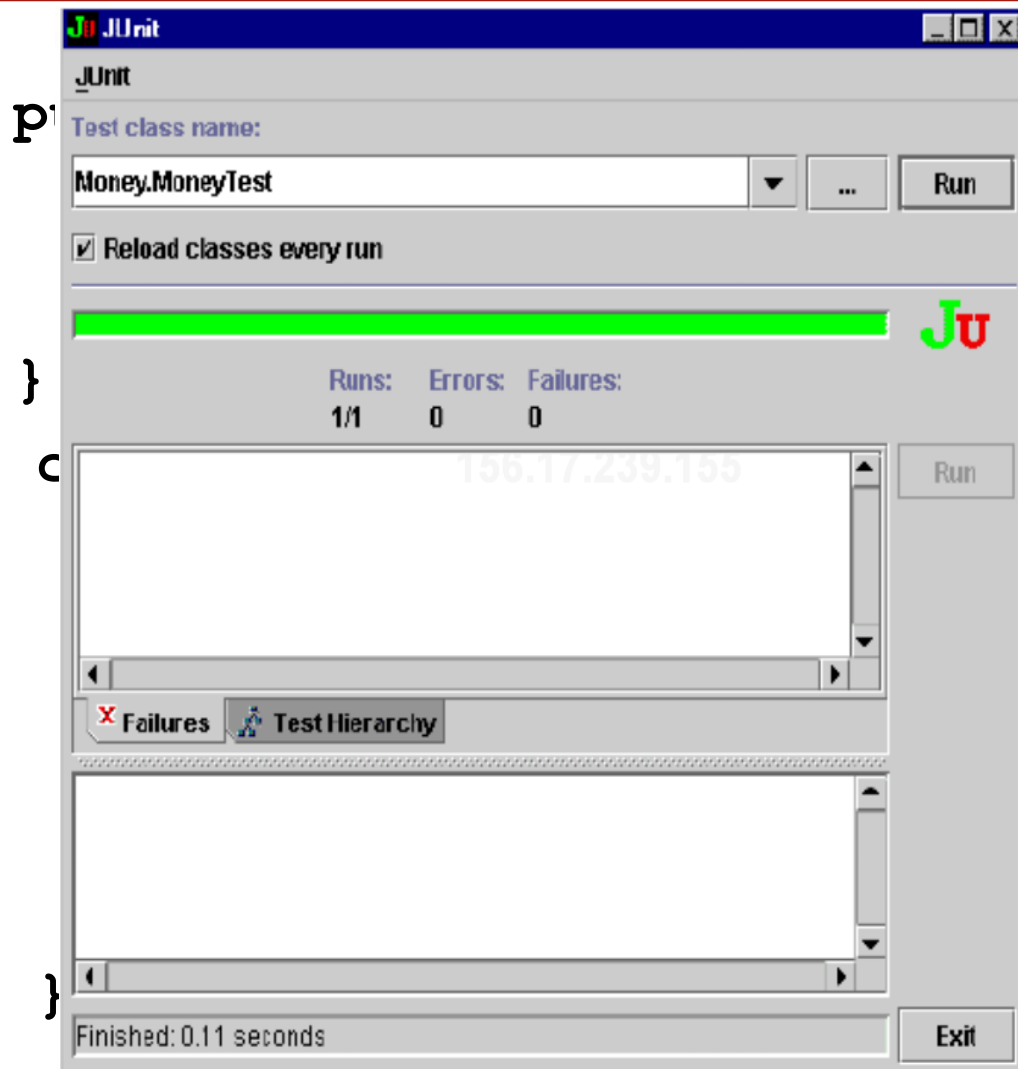
Zadania:

- $\$5 + 10\text{CHF} = \10 if kurs 2:1
- **$\$5 * 2 = \10**



Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

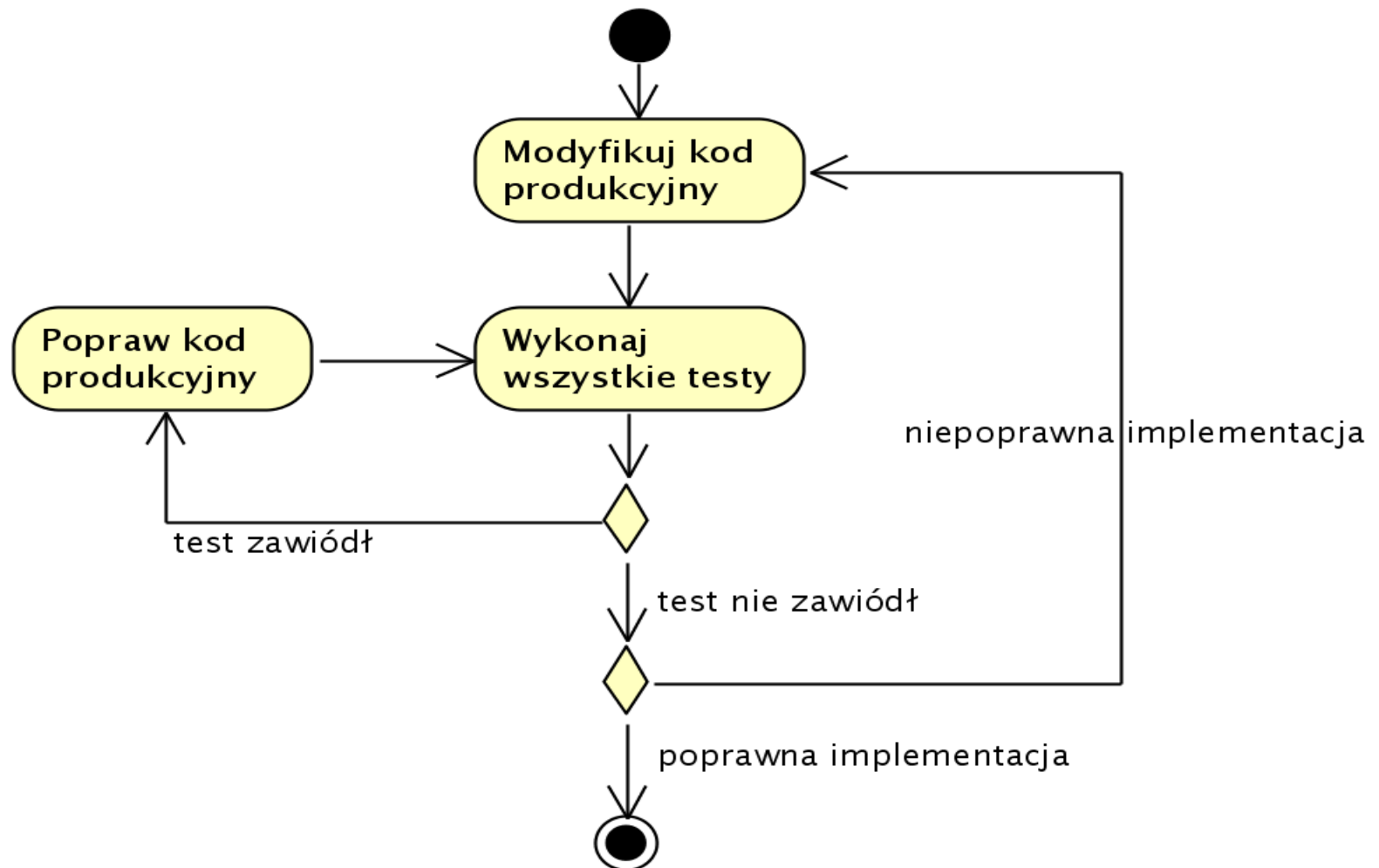


```
n () {  
    ;  
    nt) ;
```

- Publiczne pola
- Efekt uboczny mnożenia
- Typ int reprezentuje kwoty

1. Dodaj malutki test
2. Uruchom wszystkie testy – fail
3. Wykonaj drobną zmianę
4. Uruchom wszystkie testy – ok
5. Refaktoruj aby usunąć powtórzenia

Fałszywa implementacja (Fake It)





Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

Zadania:

- $\$5 + 10\text{CHF} = \10 if kurs 2:1
- ~~$\$5 * 2 = \10~~
- **Efekt uboczny mnożenia**
- 'amount' powinien być prywatny

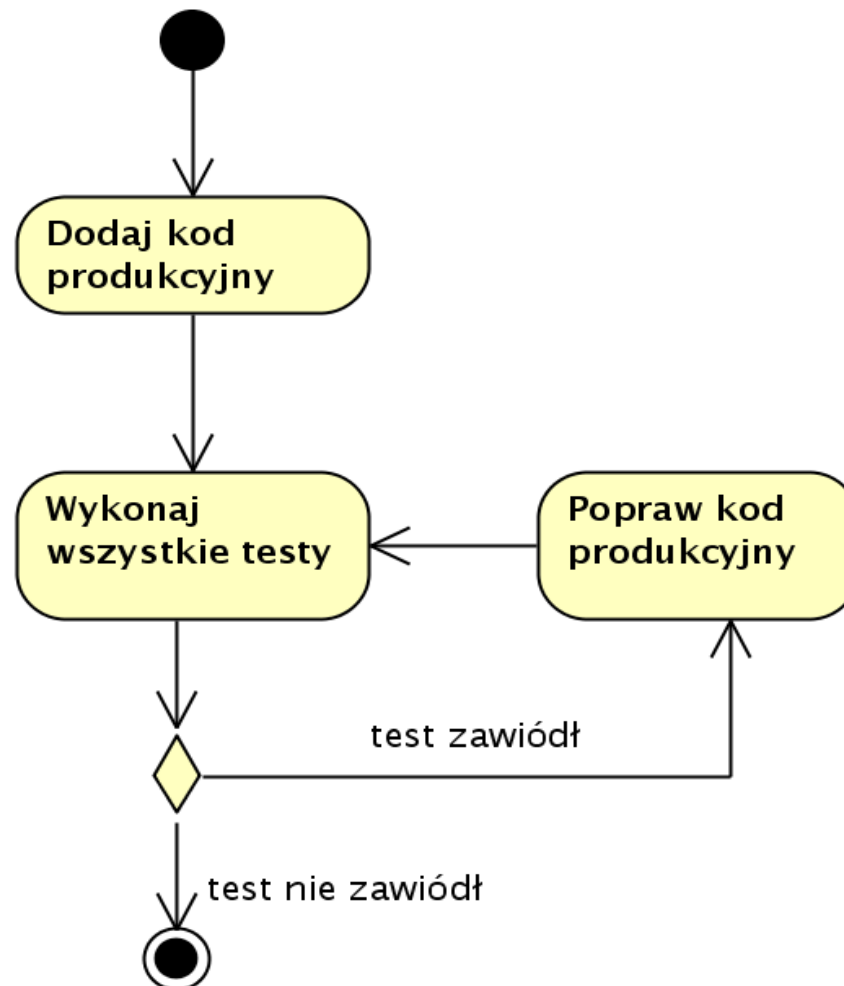


Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

```
public void testMultiplication() {  
    Dollar five = new Dollar(5);  
    Dollar product = five.times(2);  
    assertEquals(10, product.amount);  
    product = five.times(3);  
    assertEquals(15, product.amount);  
}  
  
Dollar times(int multiplier) {  
    return new Dollar(amount * multiplier) ;  
}
```

Oczywista implementacja (Obvious Implementation)





Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

Zadania:

- $\$5 + 10\text{CHF} = \10 if kurs 2:1
- ~~$\$5 * 2 = \10~~
- ~~Efekt uboczny mnożenia~~
- 'amount' powinien być prywatny
- Zaokrąglanie kwot (int)
- **equals()**
- hashCode()

Wzorzec projektowy
'Value Object'



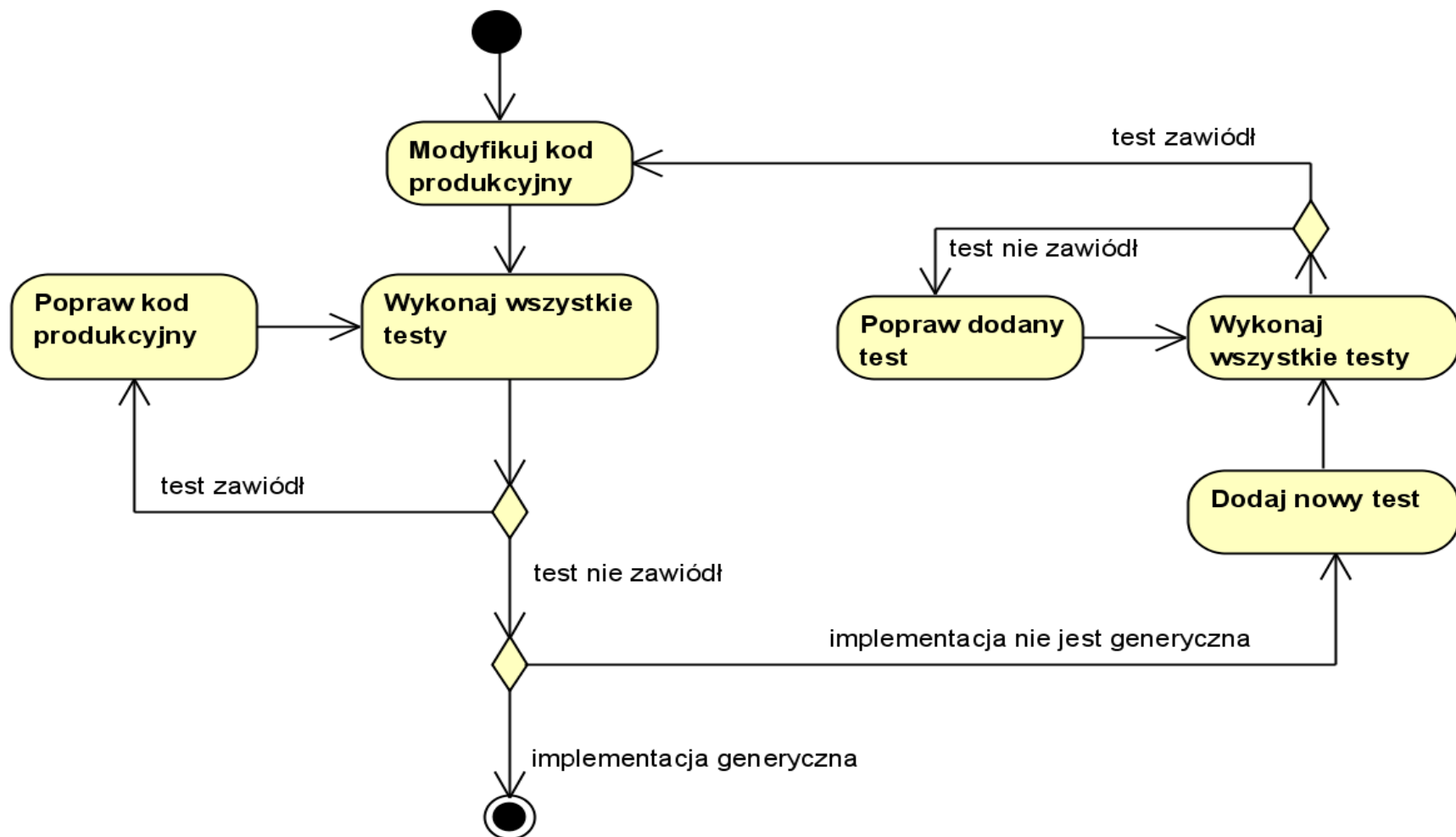
Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

```
public void testEquality() {  
    assertTrue(new Dollar(5).equals(  
        new Dollar(5)));  
    assertFalse(new Dollar(5).equals(  
        new Dollar(6))) ;  
}
```

```
public boolean equals(Object object) {  
    Dollar dollar = (Dollar) object;  
    return amount == dollar.amount;  
}
```


Triangulacja (Trinagulate)





Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

Zadania:

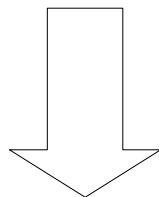
- $\$5 + 10\text{CHF} = \10 if kurs 2:1
- ~~$\$5 * 2 = \10~~
- ~~Efekt uboczny mnożenia~~
- **'amount' powinien być prywatny**
- Zaokrąglanie kwot (int)
- equals()
- hashCode()

Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

Refaktoryzacja testów:

```
public void testMultiplication() {  
    Dollar five = new Dollar(5);  
    Dollar product = five.times(2);  
    assertEquals(10, product.amount);  
    product = five.times(3);  
    assertEquals(15, product.amount);  
}
```



```
public void testMultiplication() {  
    Dollar five = new Dollar(5);  
    assertEquals(new Dollar(10), five.times(2));  
    assertEquals(new Dollar(15), five.times(3));  
}
```



Programowanie przez testy w praktyce

Kent Beck: Test-Driven Development by example

```
public void testMultiplication() {  
    Dollar five = new Dollar(5);  
    assertEquals(new Dollar(10), five.times(2));  
    assertEquals(new Dollar(15), five.times(3));  
}
```

```
class Dollar{  
    Dollar(int amount){ ...  
    {  
    Dollar times(int multiplier){...  
    }  
    private int amount;  
}
```



Skutki stosowania programowania przez testy

- Wpływ na architekturę programu
- Programista jest zmuszony do dzielenia swojego zadania na mikro-zadania
- Programista jest zmuszony do 'zrozumienia' swojego zadania przed jego implementacją
- Bardzo wysokie pokrycie kodu testami jednostkowymi
- Wysoka jakość testów jednostkowych
- Dodatkowy czas, koszt związany z przygotowywaniem testów

Testy akceptacyjne

Testy akceptacyjne, to testy funkcjonalne których celem jest wykazanie, że wyspecyfikowane wymagania zostały poprawnie zaimplementowane.

W metodykach lekkich (np. XP) często stanowią integralną część specyfikacji i są automatyzowane przy pomocy jednego z wielu dostępnych narzędzi (Fitnesse, Fit, Selenium, BadBoy, Proven, Abbot, jfcUnit, Autolt).

Kiedy wszystkie testy akceptacyjne przypisane do historii użytkownika (przypadku użycia) zostaną poprawnie przeprowadzone historia jest uważana za poprawnie zaimplementowaną



Abbot
Java GUI Test
Framework





Testy akceptacyjne a jednostkowe

Filippo Ricca: Automatic Acceptance Testing with FIT/FitNesse

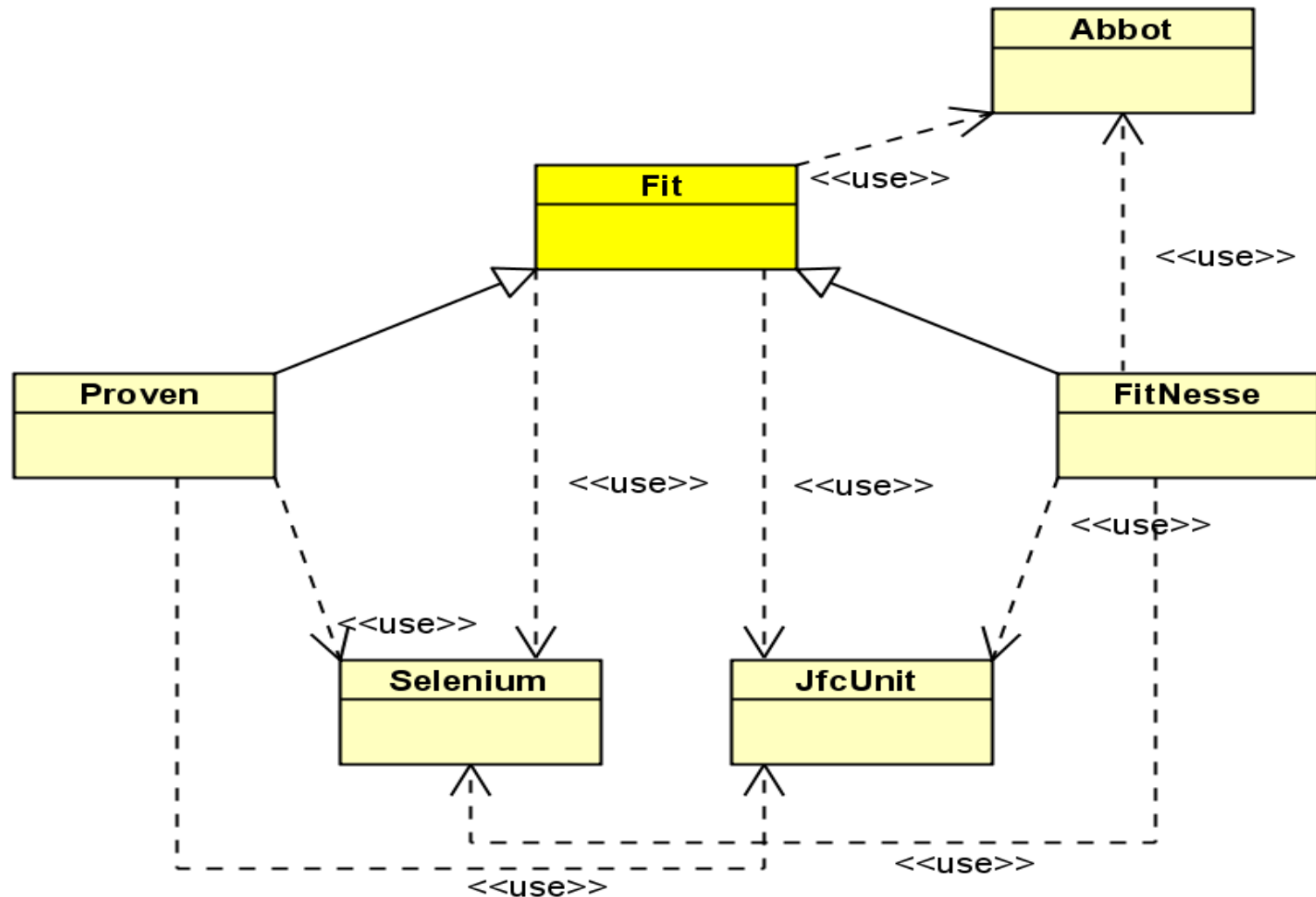
Testy akceptacyjne	Testy jednostkowe
Przygotowywane przez klienta i analityka systemowego	Przygotowywane przez programistów
Kiedy żaden z testów nie zawodzi przestań programować – system jest gotowy (XP)	Kiedy żaden z testów nie zawodzi napisz nowy test który zawiedzie (XP, TDD)
Celem jest wykazanie poprawności działania wyspecyfikowanej funkcjonalności	Celem jest znajdowanie błędów
Używane do weryfikowania kompletności implementacji; jako testy integracyjne i regresyjne; do wskazywania postępu w tworzeniu aplikacji; jako część kontraktu; jako dokumentacja wysokiego poziomu	Używane do znajdowania błędów w modułach (klasach, funkcjach, metodach, komponentach) kodu źródłowego; jako dokumentacja niskiego poziomu
Pisane przed implementacją a wykonywane po niej.	Pisane i wykonywane w trakcie implementacji
Wyzwalane przez wymaganie użytkownika (przypadek użycie, historia użytkownika...)	Wyzwalane przez potrzebę dodania nowych metod, klas..



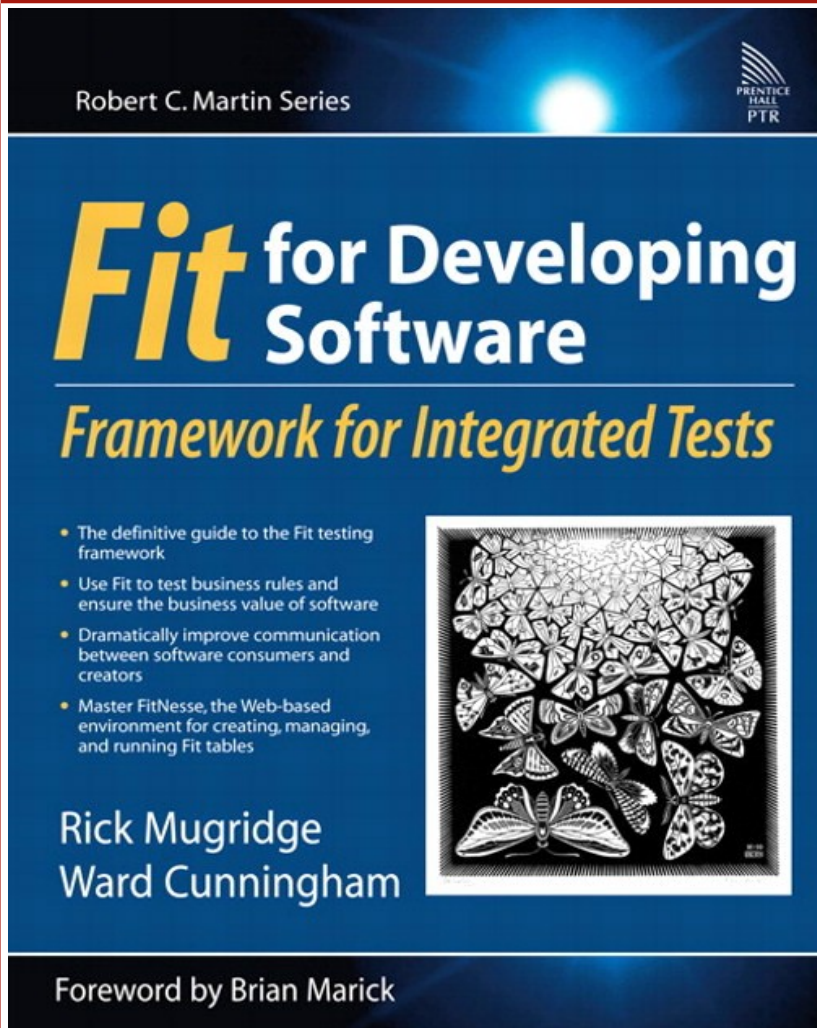
Testy akceptacyjne - narzędzia

- Testowanie przez GUI, Capture & Replay
 - Aplikacje desktopowe
 - Abbot (<http://abbot.sourceforge.net>)
 - JfcUnit (<http://jfcunit.sourceforge.net/>)
 - Aplikacje internetowe
 - Selenium (<http://seleniumhq.org/>)
- Kompleksowe rozszerzalne frameworki
 - Fit (<http://fit.c2.com/>)
 - Fitnesse (<http://fitnesse.org/>)
 - Proven

Testy akceptacyjne - narzędzia



Testy akceptacyjne z Fit(Nesse)



- Wykonywalne testy zapisywane w tabelach (HTML, XML)
- Testy z tabel łączone z testowanym systemem przy pomocy pisanych przez programistów *fixtures*
- Rozszerzalna architektura - jako *fixture* można zaimplementować integrację z innym frameworkim
- Wyniki testów w czytelnej, graficznej postaci

Testy akceptacyjne z Fit(Nesse)

Tabela z testami

Move		
direction	valid?	message?
W	true	
W	false	-You can't go that way-
N	false	-You can't go that way-
S	true	
S	true	
E	false	-You can't go that way-
S	false	-You can't go that way-

- Logika biznesowa w postaci prostej tabeli (HTML; w FitNesse również XML)
- Ułatwiają zrozumienie wymagań klienta
- Mogą być tworzone przy pomocy dowolnego edytora HTML (w FitNesse również przez Wiki)

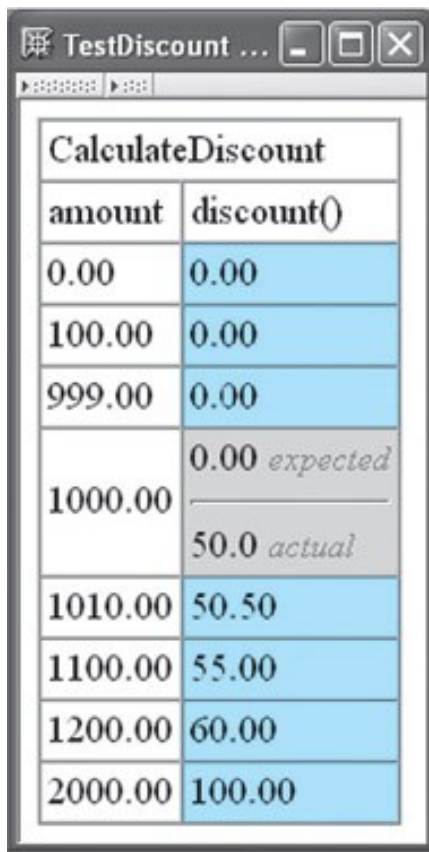
Testy akceptacyjne z Fit(Nesse) Fixture

```
public class Move extends  
    ColumnFixture{  
    public String direction;  
    private String msg;  
  
    public boolean valid() {  
        return doTest();  
    }  
  
    public String message() {  
        return msg;  
    }  
  
    private boolean doTest() {...}  
}
```

- Fixture to klasa pośrednicząca pomiędzy tabelą ze scenariuszem testowym a testowanym systemem
- Zazwyczaj przygotowywane są przez programistów

Testy akceptacyjne z Fit(Nesse)

Wyniki wykonania testów

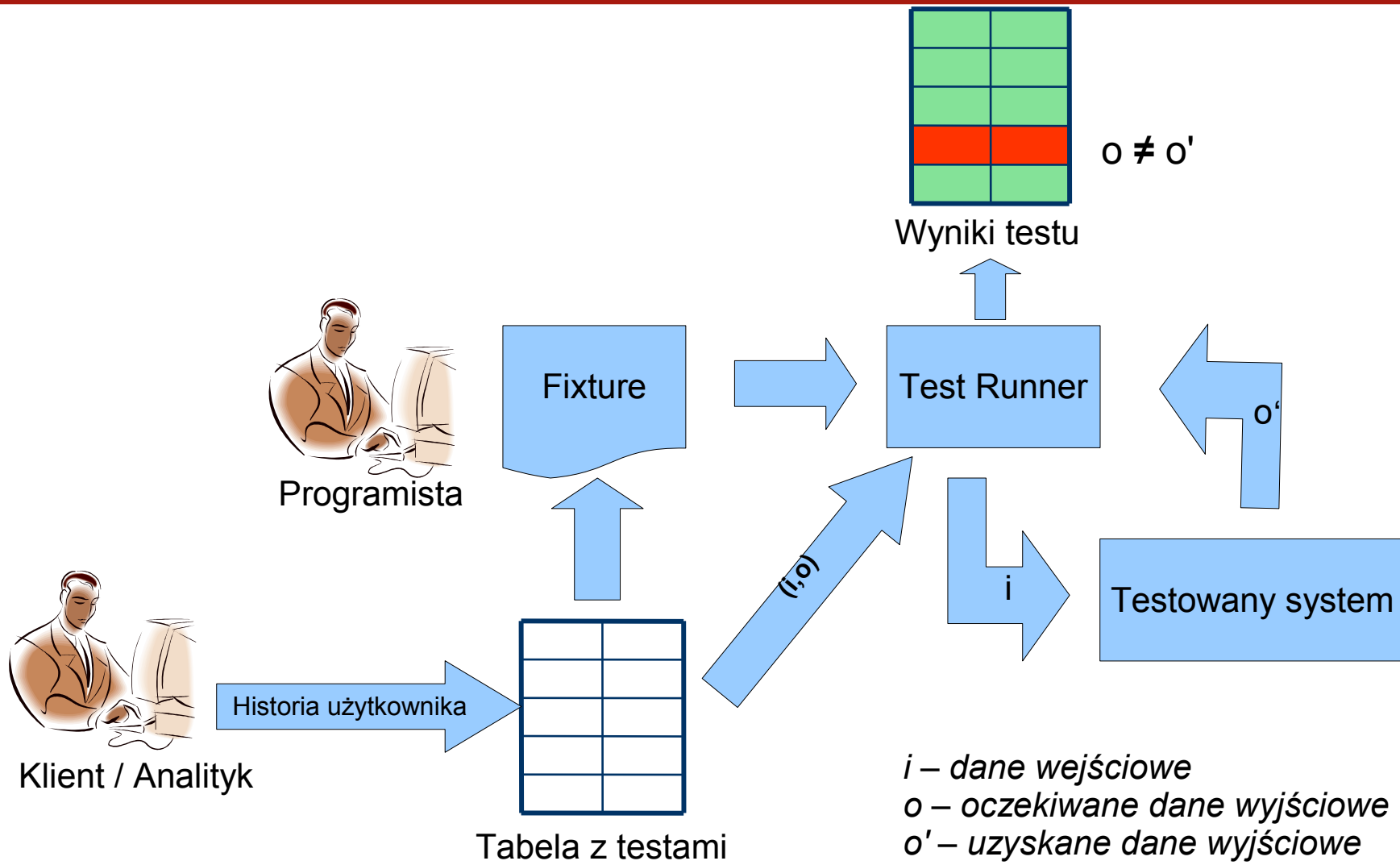


CalculateDiscount	
amount	discount()
0.00	0.00
100.00	0.00
999.00	0.00
1000.00	0.00 <i>expected</i>
	50.0 <i>actual</i>
1010.00	50.50
1100.00	55.00
1200.00	60.00
2000.00	100.00

- Wyniki działania systemu są porównywane z wartościami zapisanymi w scenariuszu testowym. Rozbieżności są raportowane
- Raport w postaci tabeli bazującej na scenariuszu testowym

Testy akceptacyjne z Fit(Nesse)

Filippo Ricca: Automatic Acceptance Testing with FIT/FitNesse



Predefiniowane typy k

- **ColumnFixture** - każdy wiersz to ...
kolumn dostarcza danych wejściowych
oczekiwane wyjście testowanego prog

- **RowFixture**

- **ActionFixt**
przypadkow
(proces bizn

- **SetUp Fixtur**
Fixture, Con

TestCredit - Mozilla

months	reliable	balance	allow
14	tr		
0	tr		
24	fa		
18	tr		
12	tr		

Tes...

user	room
anna	lotr
luke	lotr

TestBuyItems ...

fit.ActionFixture		
start	BuyActions	
check	total	00.00
enter	price	12.00
press	buy	
check	total	12.00
enter	price	100.00
press	buy	
check	total	112.00

	0.00
	0.00
	0.00
ected	1000.00 expected
ual	0.0 actual

złość

akcji

table

Testowanie przez GUI - Swing

Joseph Bergin: XP Testing a GUI with FIT, Fitnesse, and Abbot



fit.ActionFixture		
start	calculator2003.CalculatorGuiFixture	
enter	delay	500
check	value	0
press	five	
press	three	
press	plus	
press	five	
press	equals	
check	value	58
press	minus	
press	two	
press	equals	
check	value	56

Enter, check i press to metody

Metody z tej kolumny muszą
zostać zdefiniowane przez
programistę w klasie
CalculatorGuiFixture.

Testowanie przez GUI - Swing

- **Abbot**

- Wykorzystuje mechanizm refleksji do uzyskiwania dostępu do klas interfejsu graficznego działającego programu.
- Aby wykorzystać możliwości Abbota w Fitnesse należy napisać specjalny Fixture, w naszym przypadku jest to `CalculatorGuiFixture.java`.
- Zamiast Abbota można użyć `JfcUnit`.

- **FitNesse**

- `CalculatorGuiFixture.java` dziedziczy po klasie `Fixture`, więc może zostać podłączony do tabeli ze scenariuszem testowym.



Testowanie przez GUI - Swing

```
package calculator2003;
```

```
import java.awt.*;
```

```
import fit.Fixture;
```

```
import junit.extensions.abbot.*;
```

```
import abbot.script.ComponentReference;
```

```
import abbot.testers.ComponentTester;
```

```
public class CalculatorGuiFixture extends Fixture{ //from FIT
```

```
    public Calculator calc = new Calculator(); // The GUI to be tested
```

```
    Button button5 = null; // References to objects in the GUI
```

```
    Button button2 = null;
```

```
    Button button3 = null;
```

```
    Button buttonEquals = null;
```

```
    Button buttonPlus = null;
```

```
    Button buttonMinus = null;
```



Testowanie przez GUI - Swing

```
class GuiTest extends ComponentTestFixture{ // From Abbot's JUnit extensions
```

```
public GuiTest(String name){  
    super(name);  
}
```

```
public void setUp() throws Exception{
```

```
    testBasic = new ComponentTester();
```

```
    ComponentReference ref = new ComponentReference("twoButton", Button.class, "twoButton", "2");
```

```
    button2 = (Button)getFinder().findComponent(ref);
```

```
    ref = new ComponentReference("threeButton", Button.class, "threeButton", "3");
```

```
    button3 = (Button)getFinder().findComponent(ref);
```

```
    ref = new ComponentReference("fiveButton", Button.class, "fiveButton", "5");
```

```
    button5 = (Button)getFinder().findComponent(ref);
```

```
    ref = new ComponentReference("equalsButton", Button.class, "equalsButton", "equals");
```

```
    buttonEquals = (Button)getFinder().findComponent(ref);
```



Testowanie przez GUI - Swing

```
...
private int delay = 0;

public void delay(int d){ //Control the speed of the robot
    delay = d;
}
private void click(Button button){
    testBasic.actionClick(button); //robot clicks the button
    testBasic.actionDelay(delay);
}
public void five() throws Exception{
    click(button5);
}
public void three() throws Exception{
    click(button3);
}
public void two() throws Exception{
    click(button2);
}
public void equals() throws Exception{
    click(buttonEquals);
}
public void plus() throws Exception{
    click(buttonPlus);
}
```



Testowanie przez GUI - Swing

fit.ActionFixture		
start	calculator2003.CalculatorGuiRobotFixture	
enter	delay	500
check	value	0
press	five	
press	three	
press	plus	
press	five	
press	equals	
check	value	58
press	minus	
press	two	
press	equals	
check	value	56



Testowanie aplikacji internetowych - Spring

<http://agileshrugged.com/blog/?p=33>

- Nie można nadpisać instancjonowania klas Fixture
- Ale można wewnątrz klasy Fixture załadować kontekst Springa

```
public boolean isValid() {  
    BeanFactory beanFactory = new  
    ClassPathXmlApplicationContext("classpath:/spring/applicationContext.xml").getAutowireCapableBeanFactory();  
    loginService = (LoginService)beanFactory.getBean("loginService");  
    loginService.validateUser(username, password);  
}
```



Testowanie aplikacji internetowych przez interfejs użytkownika

<http://www.fitnessse.info/webtest>

- WebTest Fixtures - integracja Fitnessse z Selenium
- Fitnessse przejmuje kontrolę na przeglądarką internetową i wykonuje scenariusze testowe
- Szablon testu:

```
com.neuri.webfixture.PlainSeleniumTest
```

```
start browser firefox localhost 4444 http://www.google.com 
```

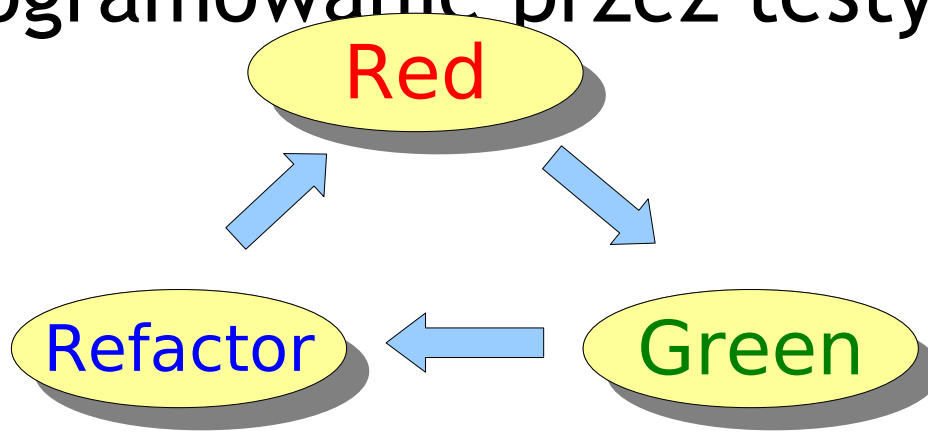
```
open http://www.google.com 
```

```
# call test methods here
```

```
shutdown browser
```

Podsumowanie

- Testy jednostkowe
 - Pisane przez programistów; służą do testowania 'jednostek' (klas, metod); JUnit
- Testy akceptacyjne
 - Pisane przez klienta lub analityka; służą do testowania funkcji biznesowych systemu; Fit, FitNesse, Selenium...
- Programowanie przez testy





Politechnika Wrocławska

Happy testing...





Politechnika Wrocławska

Instrukcje laboratoryjne:

http://gromit.iar.pwr.wroc.pl/p_inf/Fitnessse-instrukcja.html

http://gromit.iar.pwr.wroc.pl/p_inf/JUnit-instrukcja.html