Politechnika Wrocławska

Testowanie oprogramowania cz. 1

Marian Jureczko

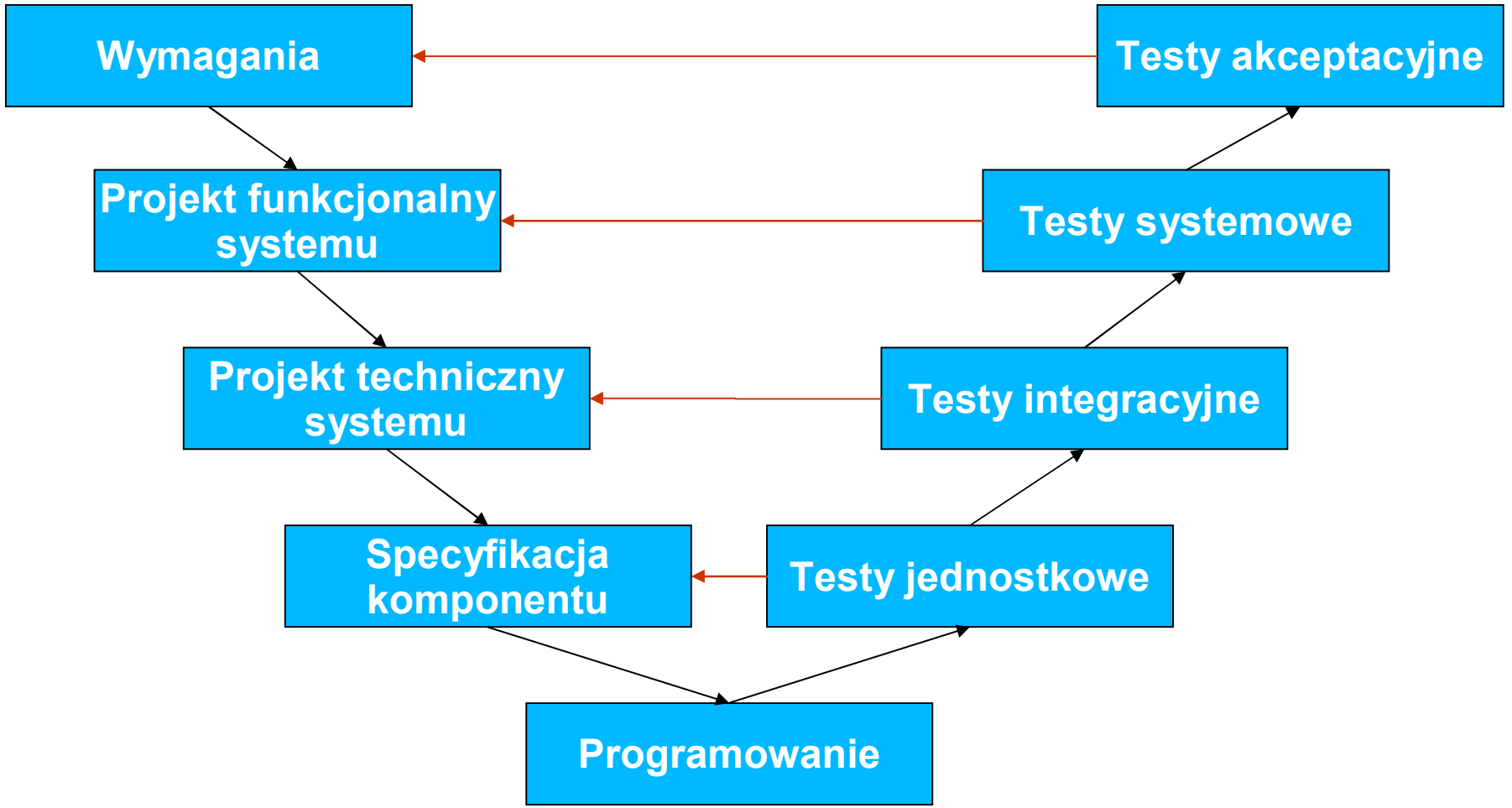
Agenda

- Rodzaje testów
- Techniki projektowania testów
- FitNesse
- Selenium

Literatura

- K. Beck: Test-Driven Development by example.
- J. Bergin: XP Testing a GUI with FIT, Fitnesse, and Abbot.
- A. Hunt: Pragmatic unit testing in Java with JUnit.
- L. Koskela: Test Driven: Practical TDD and Acceptance TDD for Java Developers.
- R. Mugridge: Fit for developing software: framework for integrated tests.
- G. J. Myers: Art of software testing.
- A. Spillner, T. Linz, H. Schaefer: Software Testing Foundations

Miejsce testów w procesie wytwarzania oprogramowania



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).

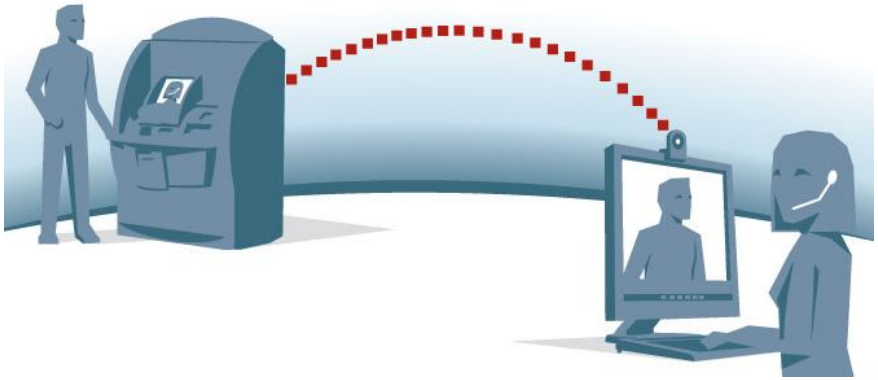
Testy integracyjne

Testy integracyjne to testy sprawdzające, że przetestowane w ramach testów jednostkowych komponenty (klasy, metody) dobrze ze sobą współpracują. Celem tych testów jest zatem wykrycie nieprawidłowości w interakcjach pomiędzy komponentami:

- niekompatybilne interfejsy komponentów,
- komponent wysyła dane o niepoprawnej syntaktyce,
- komponenty na różny sposób interpretują dane, które między sobą przesyłają,
- niespełnione ograniczenia czasowe nałożone na komunikację pomiędzy komponentami.

Testy systemowe

Testy systemowe są wykonywane po zakończeniu integracji. Podczas ich przeprowadzania testowany system powinien być uruchomiony w środowisku możliwie bliskim docelowemu. Ich celem jest skonfrontowanie działania systemu ze stawianymi przed nim wymaganiami funkcjonalnymi.



Testy akceptacyjne

Testy akceptacyjne skoncentrowane są na punkcie widzenia klienta. Ich celem jest podjęcie decyzji, czy system (lub jego poszczególne funkcjonalności) nadają się do wdrożenia. W podejściu formalnym w ramach tych testów weryfikuje się, czy wypełniony został kontrakt. W testach akceptacyjnych konieczne jest zaangażowanie klienta zamawiającego system.

W metodykach zwinnych testy akceptacyjne mogą zastępować specyfikację wymagań.

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, AutoIt).

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 użycia, historia użytkownika...)	Wyzwalane przez potrzebę dodania nowych metod, klas..

Rodzaje testów

- Testy funkcjonalne
- Testy niefunkcjonalne
  - Testy obciążeniowe
  - Testy wydajnościowe
  - Testy odpornościowe
  - Testy użyteczności i ergonomii
- Testy regresyjne

Agenda

- Rodzaje testów
- Techniki projektowania testów
- FitNesse
- Selenium

Techniki projektowania testów

- Testy białej skrzynki
- Testy czarnej skrzynki

Klasy ekwiwalencji

- Pracownicy otrzymują dodatek świąteczny równy 20% miesięcznego dochodu jeżeli pracują w firmie dłużej niż 3 lata. Pracownicy pracujący od ponad 5 lat otrzymują 30%, a ci którzy pracują więcej niż 8 lat otrzymują 50%.

Klasa ekwiwalencji	Przykładowa wartość
0 <= x <= 3	2
3 < x <= 5	4
5 < x <= 8	7
8 < x	11

Spillner, Linz Schaefer: Software Testing Foundations

Klasy ekwiwalencji dla wartości niepoprawnych

- Pracownicy otrzymują dodatek świąteczny równy 20% miesięcznego dochodu jeżeli pracują w firmie dłużej niż 3 lata. Pracownicy pracujący od ponad 5 lat otrzymują 30%, a ci którzy pracują więcej niż 8 lat otrzymują 50%.

Klasa ekwiwalencji	Przykładowa wartość
0 > x	-2
X > 49	77

Spillner, Linz Schaefer: Software Testing Foundations

Wartości graniczne

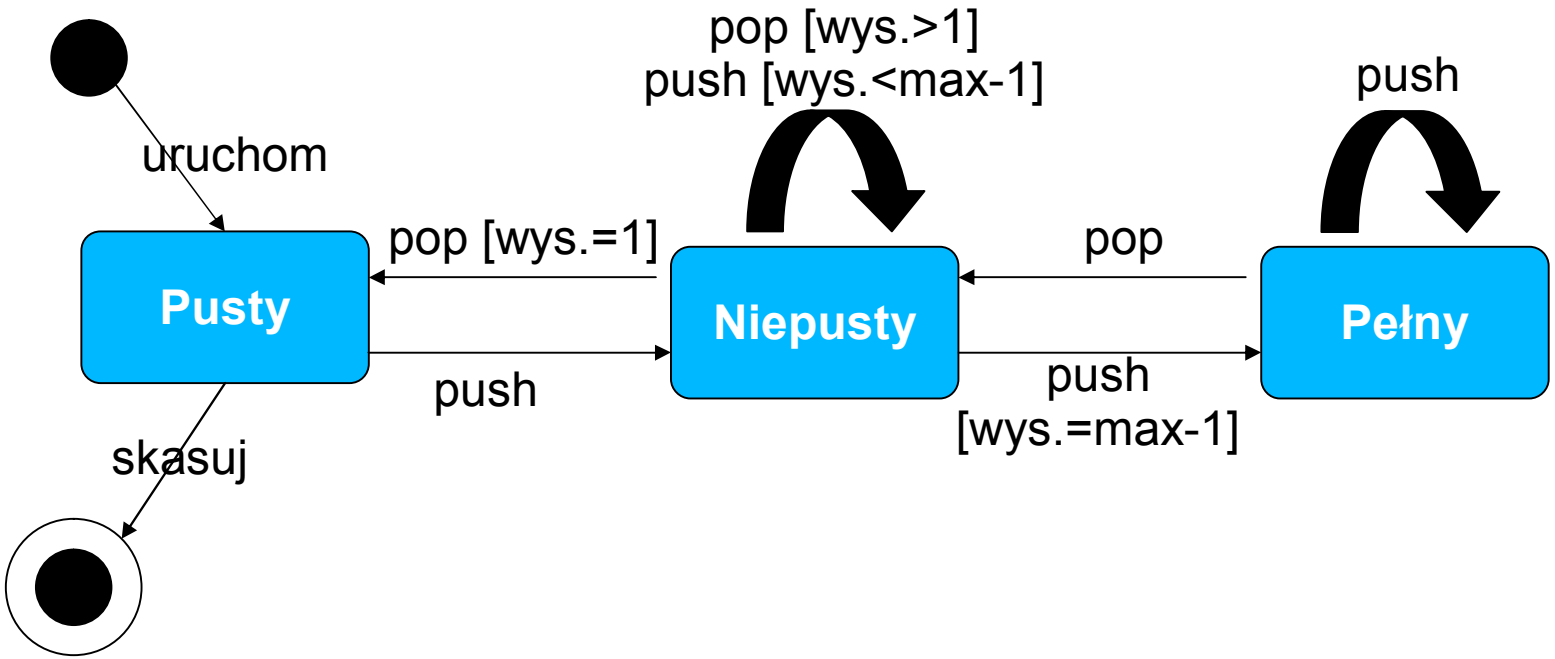
- Pracownicy otrzymują dodatek świąteczny równy 20% miesięcznego dochodu jeżeli pracują w firmie dłużej niż 3 lata. Pracownicy pracujący od ponad 5 lat otrzymują 30%, a ci którzy pracują więcej niż 8 lat otrzymują 50%.

Przypadki testowe:

- 0 lat – 1 dzień
- 0 lat
- 3 lata
- 3 lata + 1 dzień
- 5 lat
- 5 lat + 1 dzień
- 8 lat
- 8 lat + 1 dzień
- 49 lat
- 49 lat + 1 dzień

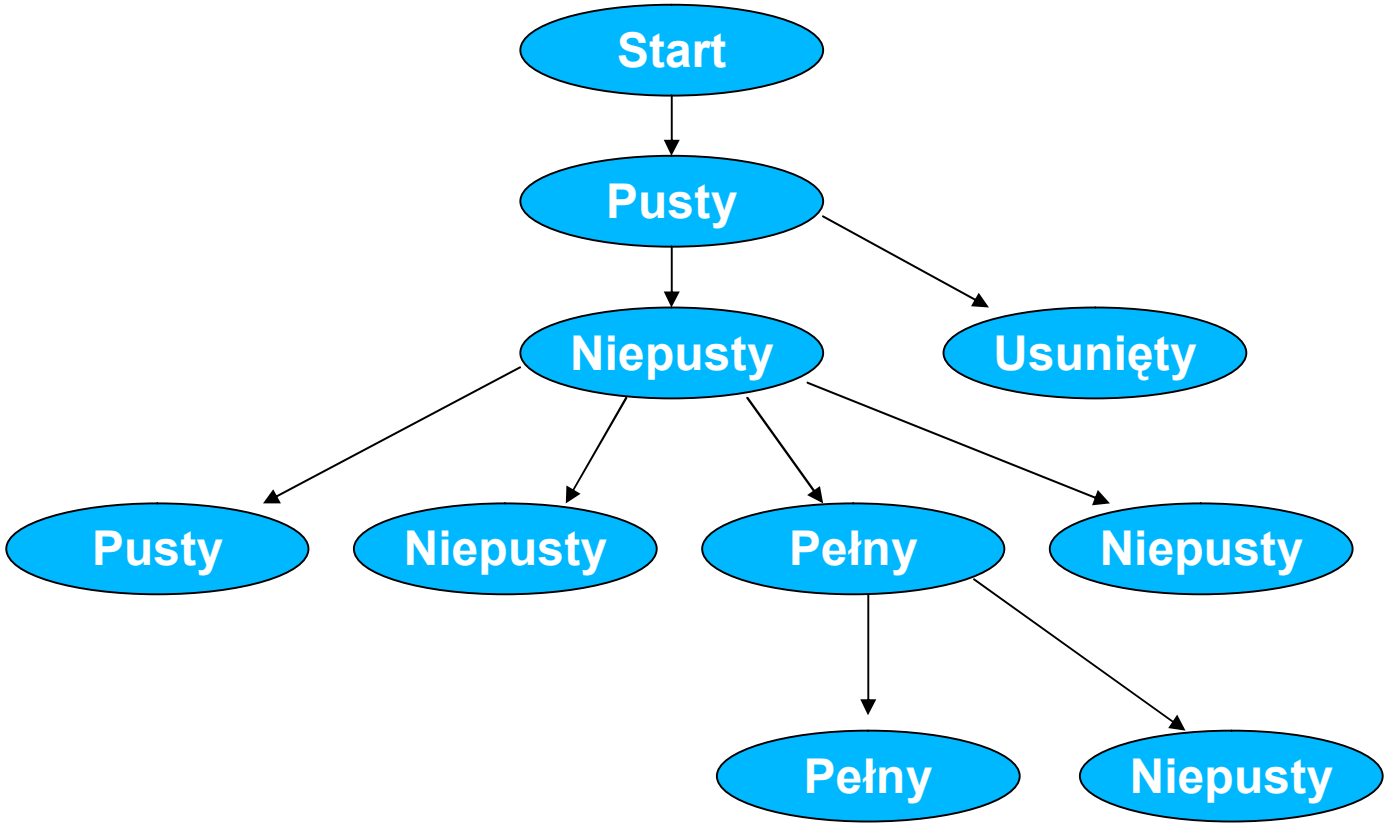
Przejścia między stanami

Stos



Spillner, Linz Schaefer: Software Testing Foundations

Przejścia między stanami - drzewo osiągalności



Spillner, Linz Schaefer: Software Testing Foundations

Tabela decyzyjna

Podejmowanie pieniędzy w bankomacie

Warunki:

- karta jest ważna,
- wprowadzono poprawny PIN,
- maksymalna liczba prób wprowadzenia numeru PIN wynosi 3,
- są dostępne środki na koncie i w bankomacie.

Potencjalne akcje systemu:

- odrzuć kartę,
- poproś o ponowne podanie numeru PIN,
- „połknij” kartę,
- poproś o podanie innej kwoty,
- wypłać pieniądze.

Spillner, Linz Schaefer: Software Testing Foundations

Tabela decyzyjna

karta jest ważna	N	T	T	T	T
wprowadzono poprawny PIN	-	N	N	T	T
3 niepoprawne PINy	-	N	T	-	-
dostępne środki	-	-	-	N	T
odrzuć kartę	T	N	N	N	N
poproś o ponowne podanie PINu	N	T	N	N	N
„połknij” kartę	N	N	T	N	N
poproś o podanie innej kwoty	N	N	N	T	N
wypłać pieniądze	N	N	N	N	T

Spillner, Linz Schaefer: Software Testing Foundations

Testowanie z wykorzystaniem przypadków użycia

Każdy przypadek użycia ma określony cel, a jego wykonanie powinno doprowadzić do określonego rezultatu. Następujące elementy powinny zostać uwzględnione podczas projektowania przypadków testowych:

- warunki uruchomienia,
- wszystkie możliwe przepływy sterowania,
- warunki zakończenia (stan systemu po zakończeniu przypadku użycia zarówno sukcesem jak i porażką)

Pokrycie kodu testami

- Pokrycie instrukcji kodu źródłowego
- Pokrycie przepływu sterowania
- Pokrycie przepływu danych
- Mutation score

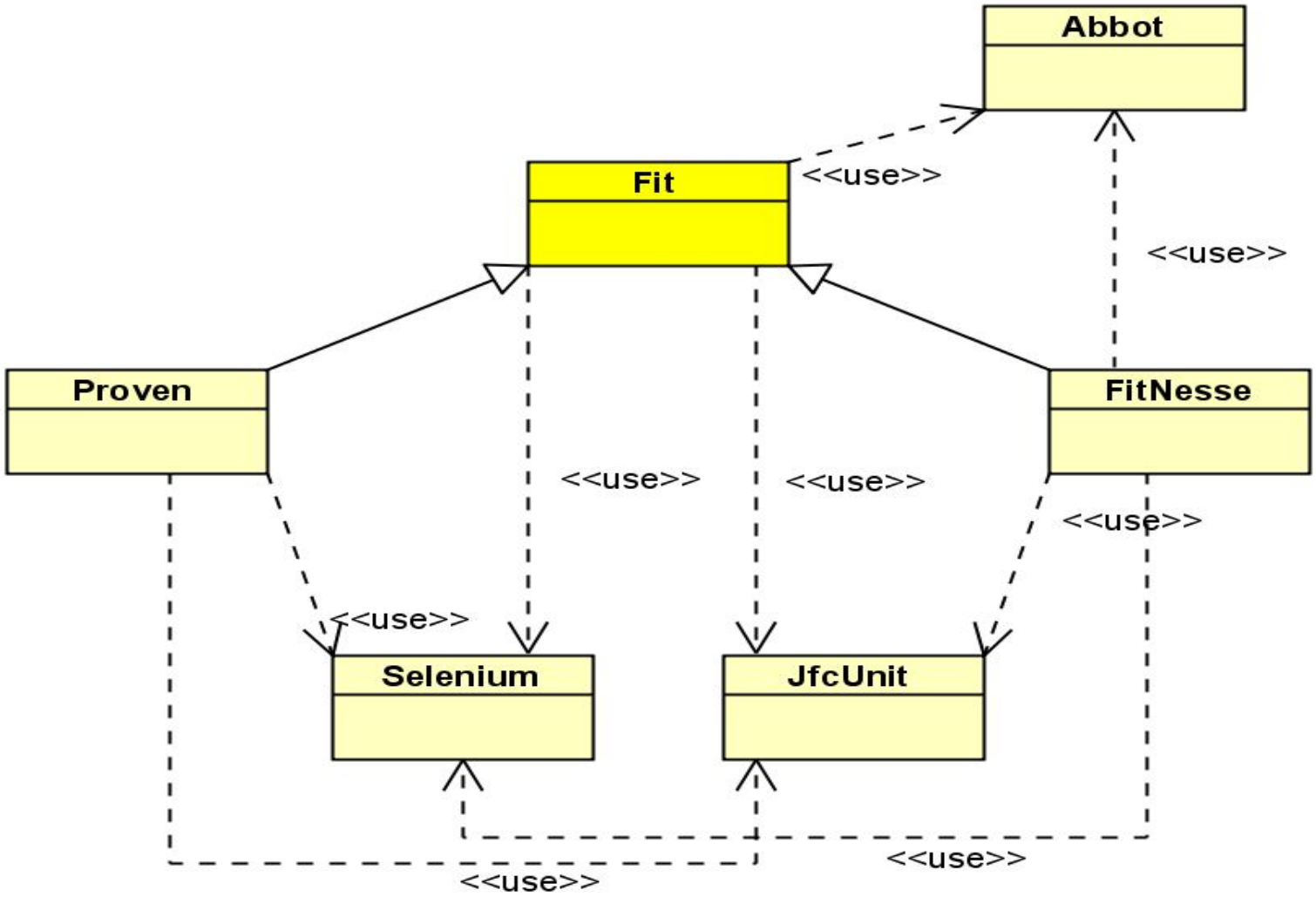
```
  :     /**
  :      * Sets the bank account's balance.
  :      *
  :      * @param  float $balance
  :      * @throws InvalidArgumentException
  :      * @access public
  :      */
  :     public function setBalance($balance)
  :     {
1 :         if ($balance >= 0) {
0 :             $this->balance = $balance;
0 :         } else {
1 :             throw new InvalidArgumentException;
  :         }
0 :     }
```

Politechnika Wrocławska

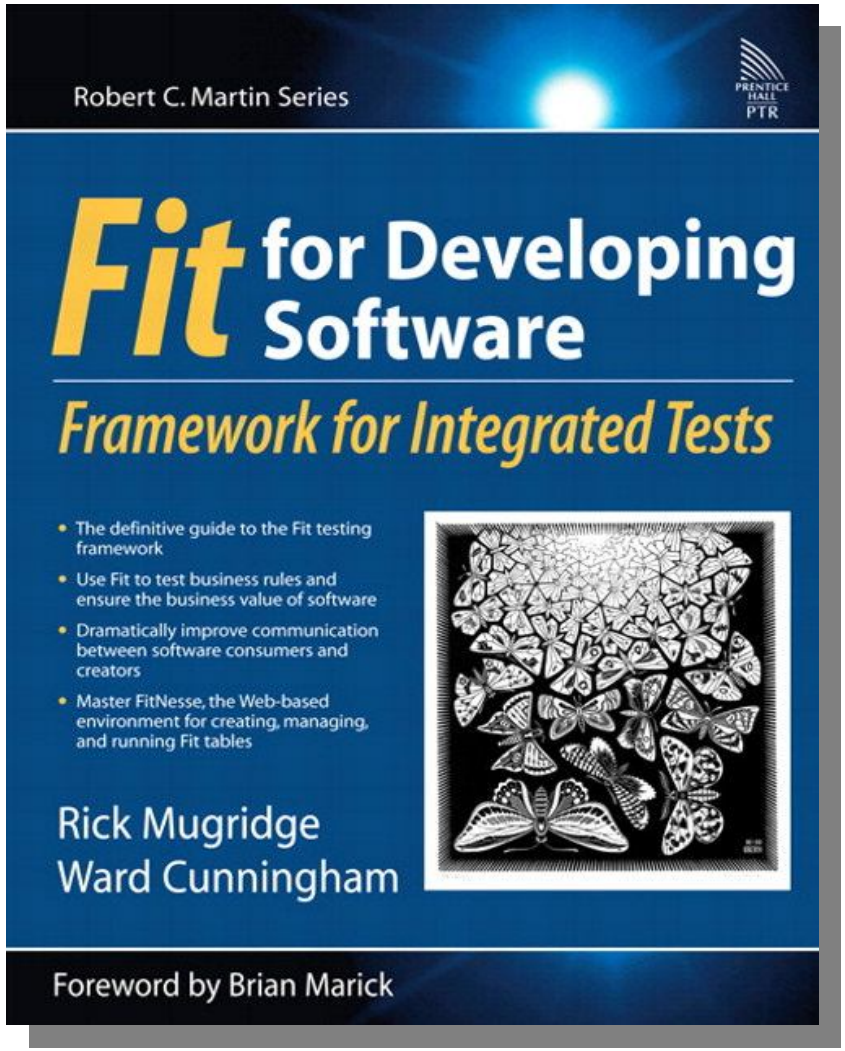
Agenda

- Rodzaje testów
- Techniki projektowania testów
- FitNesse
- Selenium

Narzędzia powiązane z FitNesse



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-

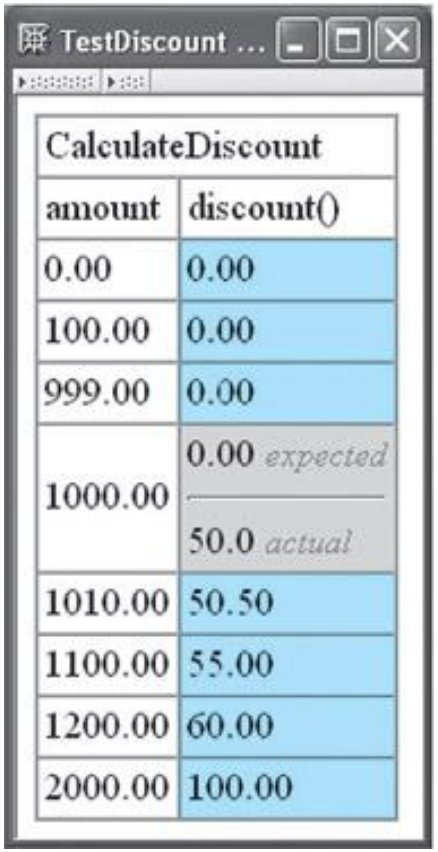
- Logika biznesowa w postaci prostej tabeli (HTML; w FitNesse również XML)
- Ułatwiają zrozumienie wymagań klienta
- Mogą być tworzone przy pomocy dowolnego edytora HTML (w FitNesse również przez Wiki)

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

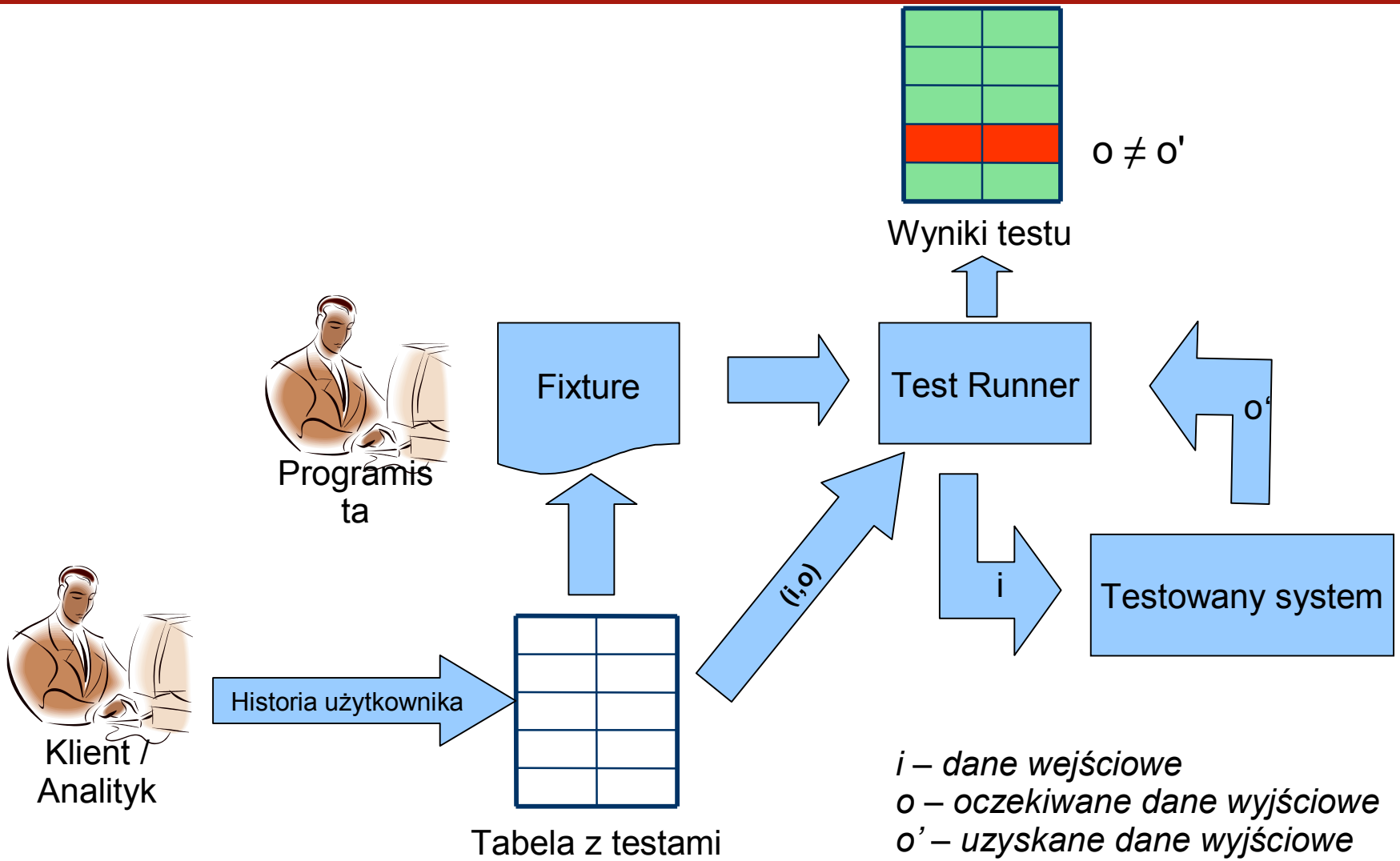


- Wyniki działania systemu są porównywane z wartościami zapisanymi w scenariuszu testowym. Rozbieżności są raportowanie
- Raport w postaci tabeli bazującej na scenariuszu testowym

Przykładowa tabela z Rick Mugridge: Fit for developing software: framework for integrated tests.

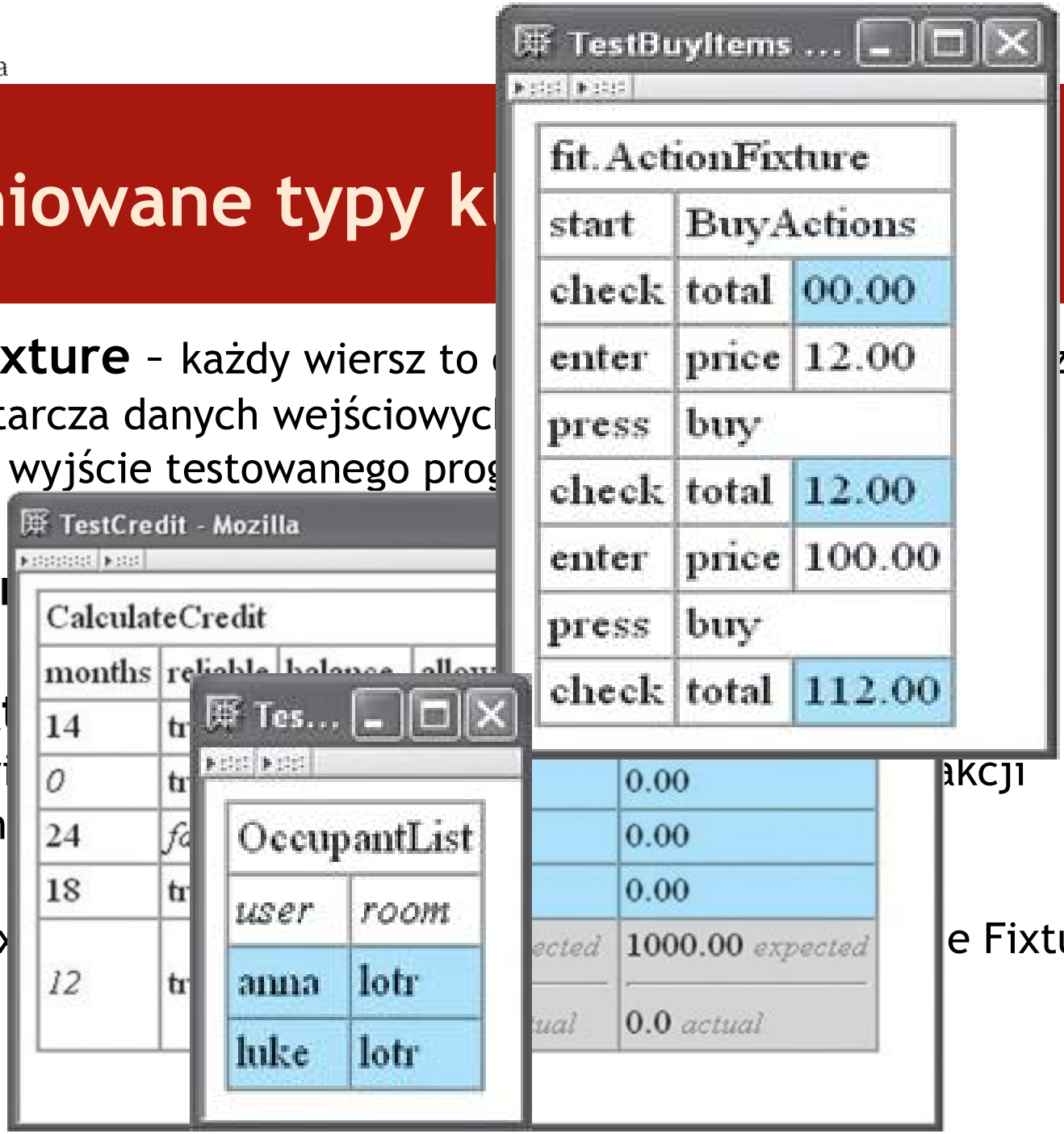
Testy akceptacyjne z Fit(Nesse)

Filippo Ricca: Automatic Acceptance Testing with FIT/FitNesse



Predefiniowane typy k

- **ColumnFixture** – każdy wiersz to
  kolumn dostarcza danych wejściowyc
  oczekiwane wyjście testowanego prog
- **RowFixtu**
- **ActionFix**
  przypadkow
  (proces bizn
- Summary Fix



Przykładowe tabele z Rick Mugridge: Fit for developing software: framework for integrated tests.

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.tester.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;
 TextField display = 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",
```

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);
}
```

Politechnika Wrocławska

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/applicati
    onContext.xml").getAutowireCapableBeanFactory();
   loginService =
    (LoginService)beanFactory.getBean("loginService");
   loginService.validateUser(username, password);
}
```

Testowanie aplikacji internetowych przez interfejs użytkownika

http://www.fitnesse.info/webtest

- WebTest Fixtures – integracja Fitnesse z Selenium
- Fitnesse 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

Politechnika Wrocławska

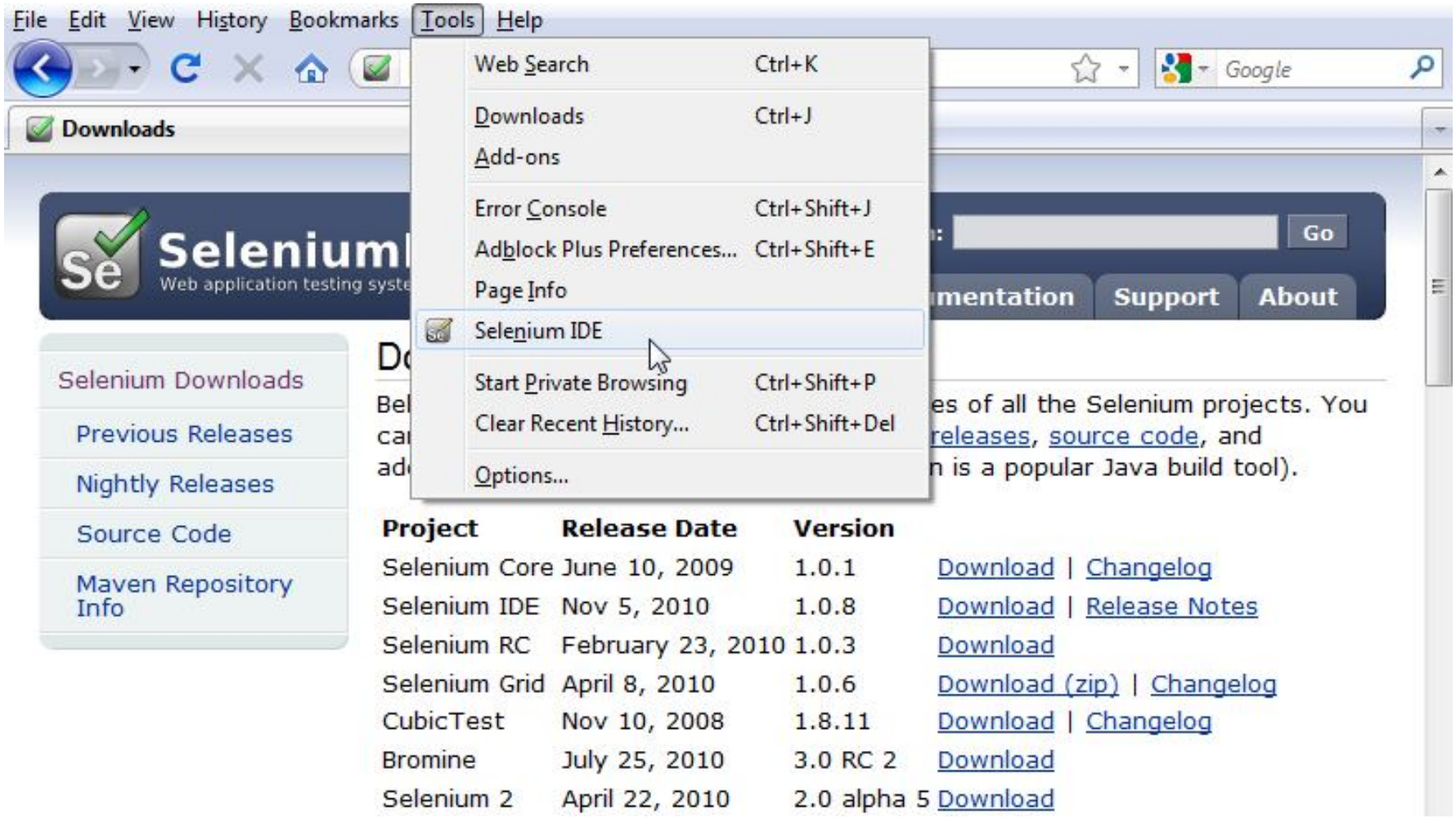
Agenda

- Rodzaje testów
- Techniki projektowania testów
- FitNesse
- Selenium

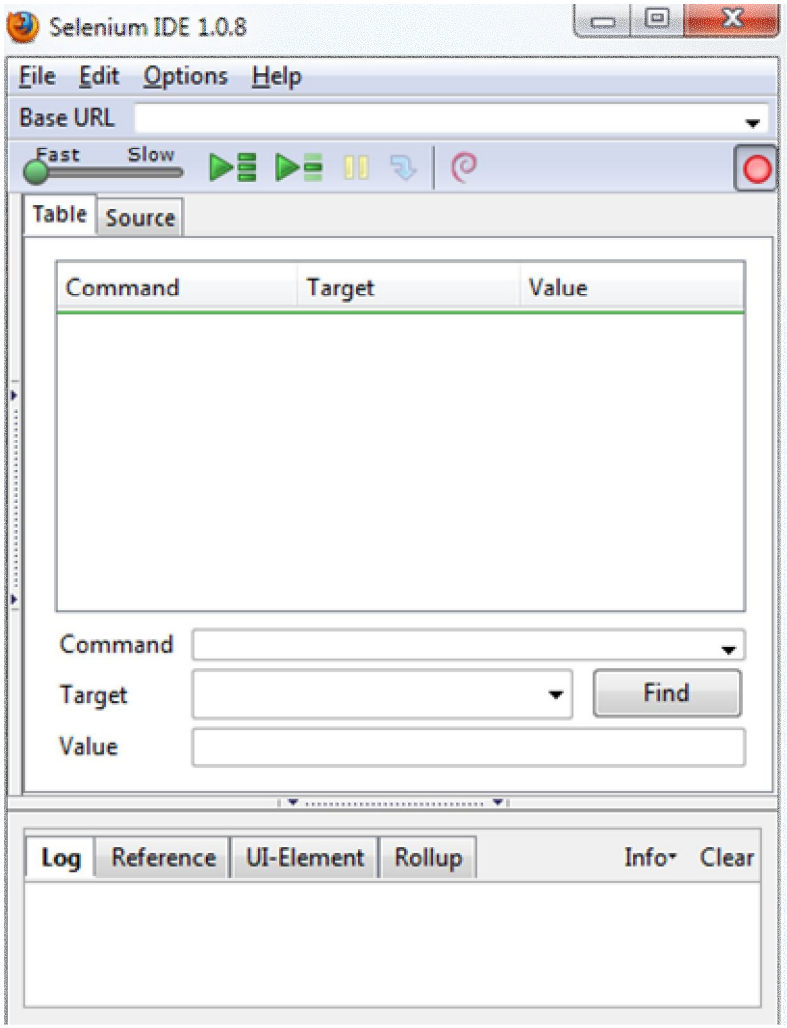
Selenium

Selenium to narzędzie do automatyzacji testów akceptacyjnych dla aplikacji internetowych. Integruje się z przeglądarką internetową i umożliwia nagrywanie akcji wykonywanych przez użytkownika. Zapisane akcje można przekonwertować do wybranego języka programowania (np. Java, C#) i wykonywać dowolną ilość razy jako testy regresyjne.

Selenium



Selenium – panel kontrolny



Selenium – tworzenie testu

- Fragment nagranego scenariusza testowego

Command	Target	Value
open	/	
waitForPageToLoad		
clickAndWait	xpath=id('menu_download')/a	
assertTitle	Downloads	
verifyText	xpath=id('mainContent')/h2	Downloads

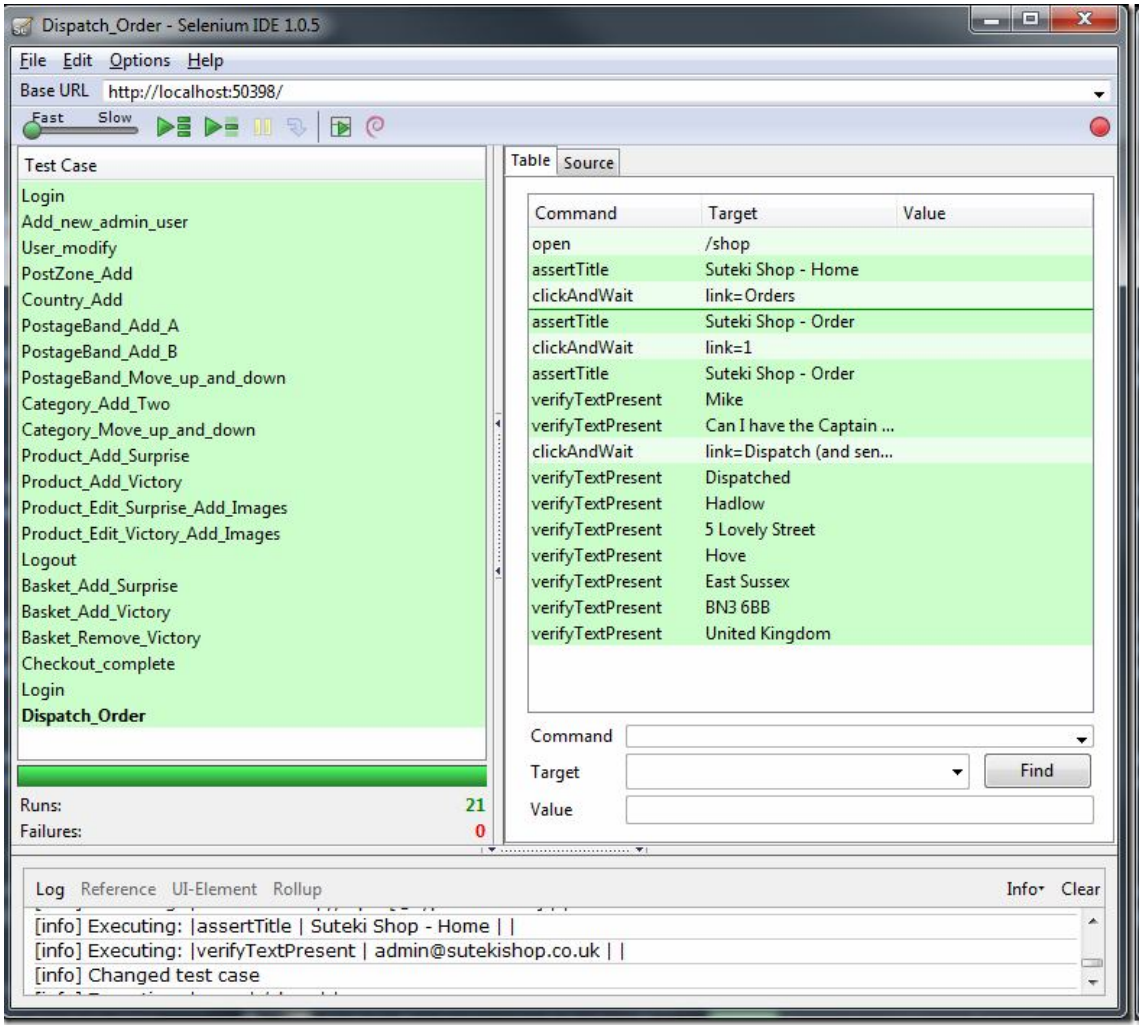
Selenium – zestawy testów

```
<html>
<head>
	<title>Test Suite Function Tests - Priority 1</title>
</head>
<body>
 <table>
  <tr><td><b>Suite Of Tests</b></td></tr>
  <tr><td><a href="./Login.html">Login</a></td></tr>
  <tr><td><a href="./SearchValues.html">Test Searching</a></td></tr>
  <tr><td><a href="./SaveValues.html">Test Save</a></td></tr>
 </table>
</body>
</html>
```

Selenium – najważniejsze polecenia

- open
- click/clickAndWait
- verifyTitle/assertTitle
- verifyTextPresent
- verifyElementPresent
- verifyText
- verifyTable
- waitForPageToLoad
- waitForElementPresent

Selenium – wyniki testu



Politechnika Wrocławska

Happy testing...

Politechnika Wrocławska

Wersja elektroniczna wykładu,
instrukcje laboratoryjne:

http://staff.iiar.pwr.wroc.pl/marian.jureczko