Testy automatyczne

Korzystające z jUnit

Cytaty

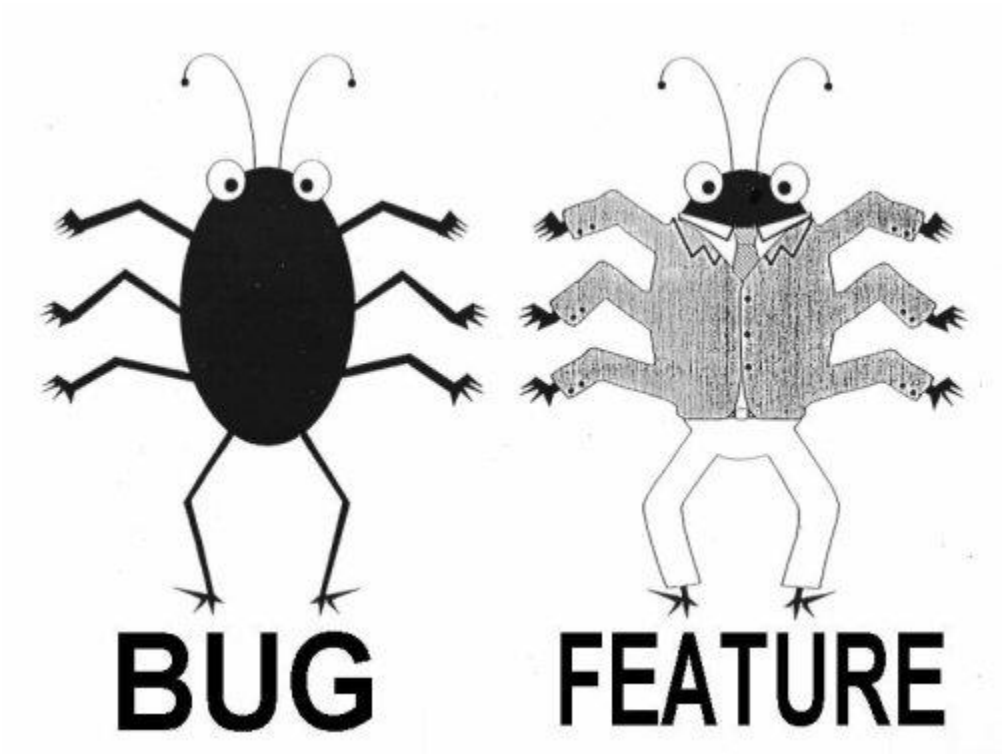
- *Kiedy zawiesza się program konkurencji, to jest awaria. Kiedy zawiesza się własny program, to jest „drobiazg". Często po awarii pojawia się komunikat typu „ID 02". „ID" to skrót od „idiotyczny drobiazg", a następująca po nim liczba wskazuje, przez ile miesięcy produkt informatyczny powinien być jeszcze testowany*

  Guy Kawasaki, The Macintosh Way

- *Uwielbiam ostateczne terminy. Zwłaszcza podoba mi się świst, jaki wydają kiedy nas mijają.*

  Douglas Adams, „The Hitch Hiker's Guide to the Galaxy

Bug, skąd się wziął



Sukces vs porażka

Sukces

- Wykrycie błędów w procesie testowania

Porażka

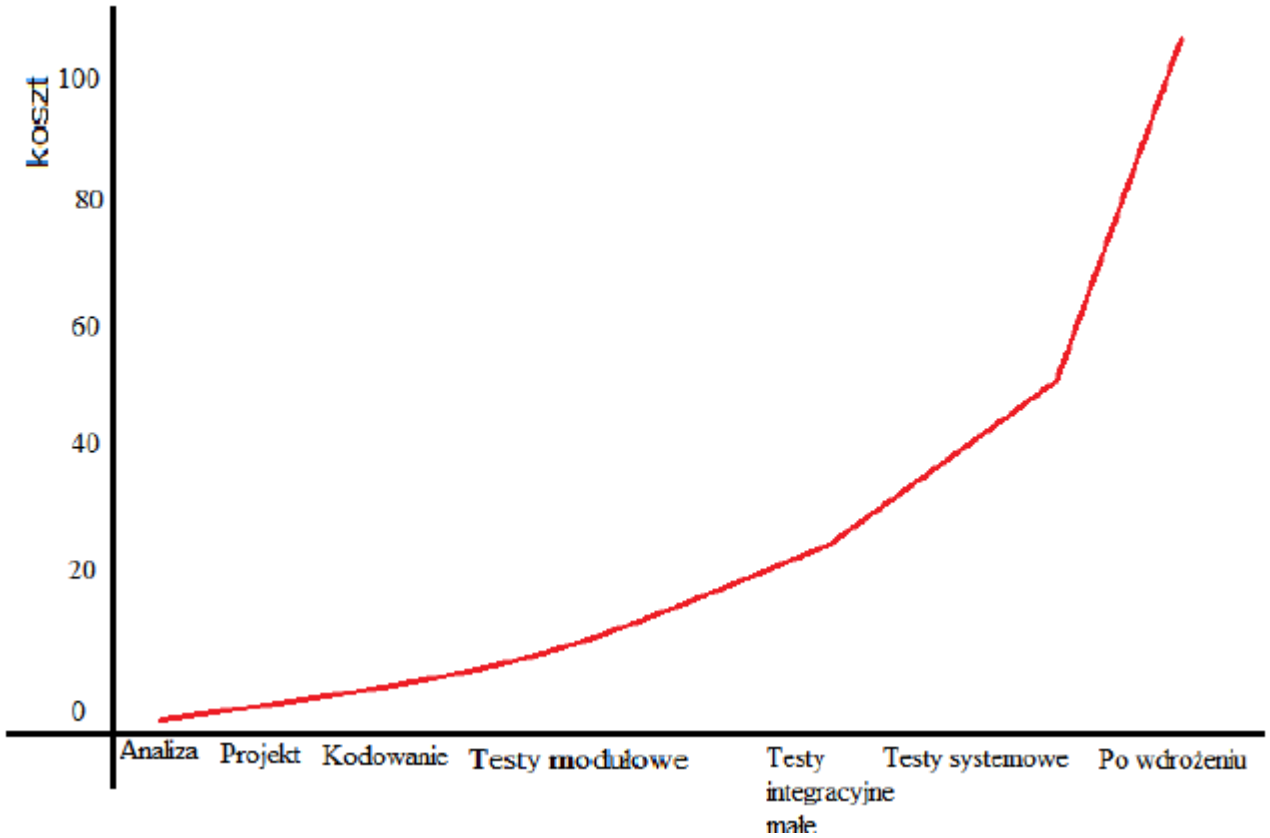
- Sytuacja odwrotna

Analogia: pacjent z subiektywnymi objawami, wykonanie badań. Dobre wyniki laboratoryjne (testy) są porażką => nie wiadomo co pacjentowi dolega.

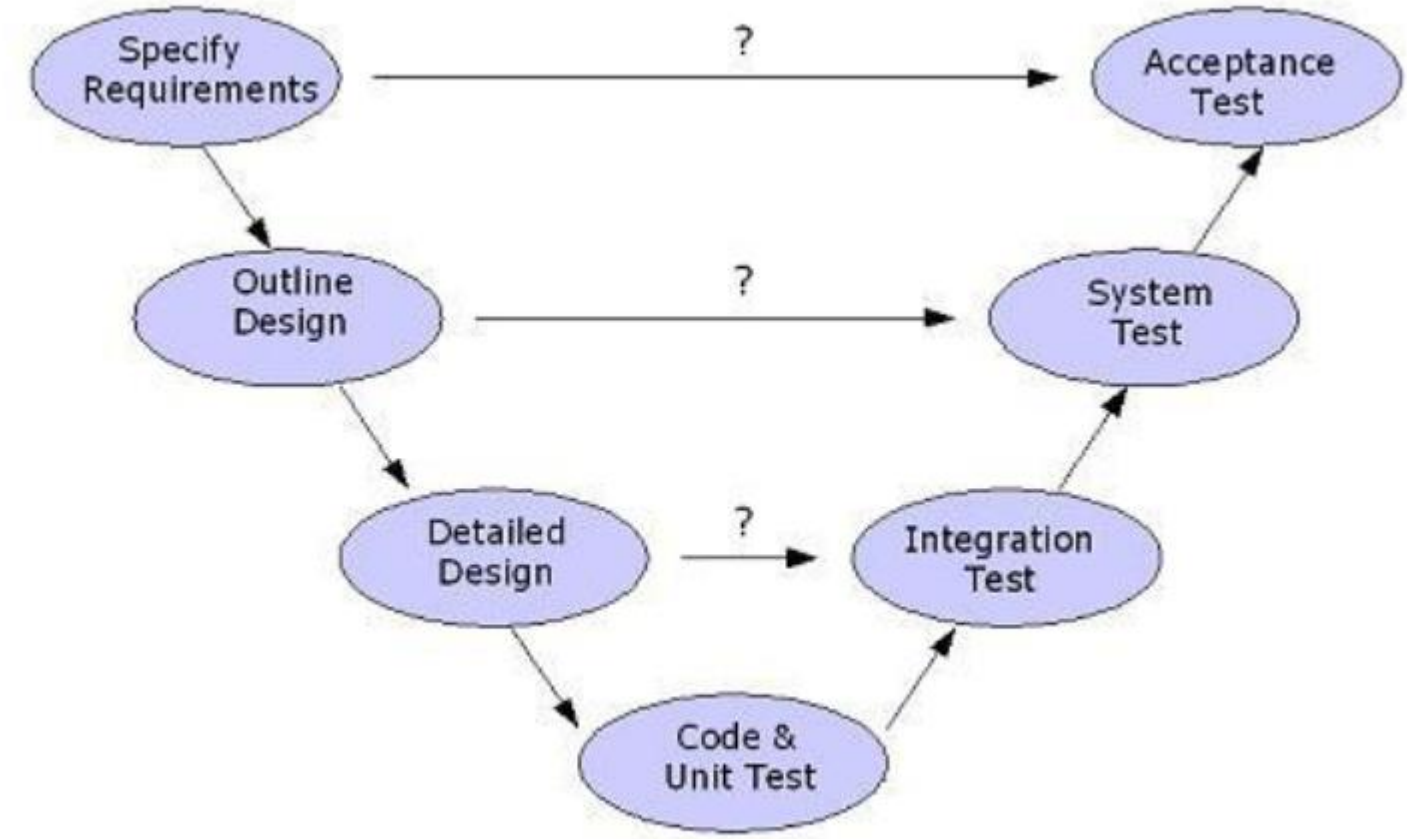
„„Błąd”” oprogramowania

1. Oprogramowanie nie wykonuje czegoś, co wg specyfikacji powinno
2. Oprogramowanie robi coś, czego nie powinno ☺
3. Oprogramowanie robi coś, o czym specyfikacja nie wspomina.
4. Oprogramowanie nie robi czegoś, czego nie ma w specyfikacji, ale być powinno.
5. Oprogramowanie jest trudne do zrozumienia, trudne do użycia, powolne – albo zdaniem testera „nieprawidłowe”

Koszty usuwania błędów



Model V


Specify Requirements
?
Acceptance Test
Outline Design
?
System Test
Detailed Design
?
Integration Test
Code & Unit Test

Testowanie totalne

- Programu nie da się przetestować całkowicie:
  - Dane wejściowe są ogromne
  - Dane wyjściowe jeszcze większe
  - Ścieżki działania programu są niezliczone
- Test nigdy nie udowodni braku błędów

Testy jednostkowe

- Test jednostkowy to fragment kodu napisany przez dewelopera, który wykonuje fragment testowanej aplikacji i ewaluuje jej wynik.
- Dobry program to taki, którego *pokrycie testami* jest maksymalne (czyli procent kodu testowanego)
- Unit testy pokrywają mały fragment kodu (zwykle jedną metodę lub maksymalnie klasę)
- Testy jednostkowe zapewniają zamierzone działanie kodu. Zwykle są wykonywane wielokrotnie w trakcie powstawania aplikacji aby być pewnym, że funkcjonalność działa przez cały czas pomimo zmian w kodzie.

jUnit

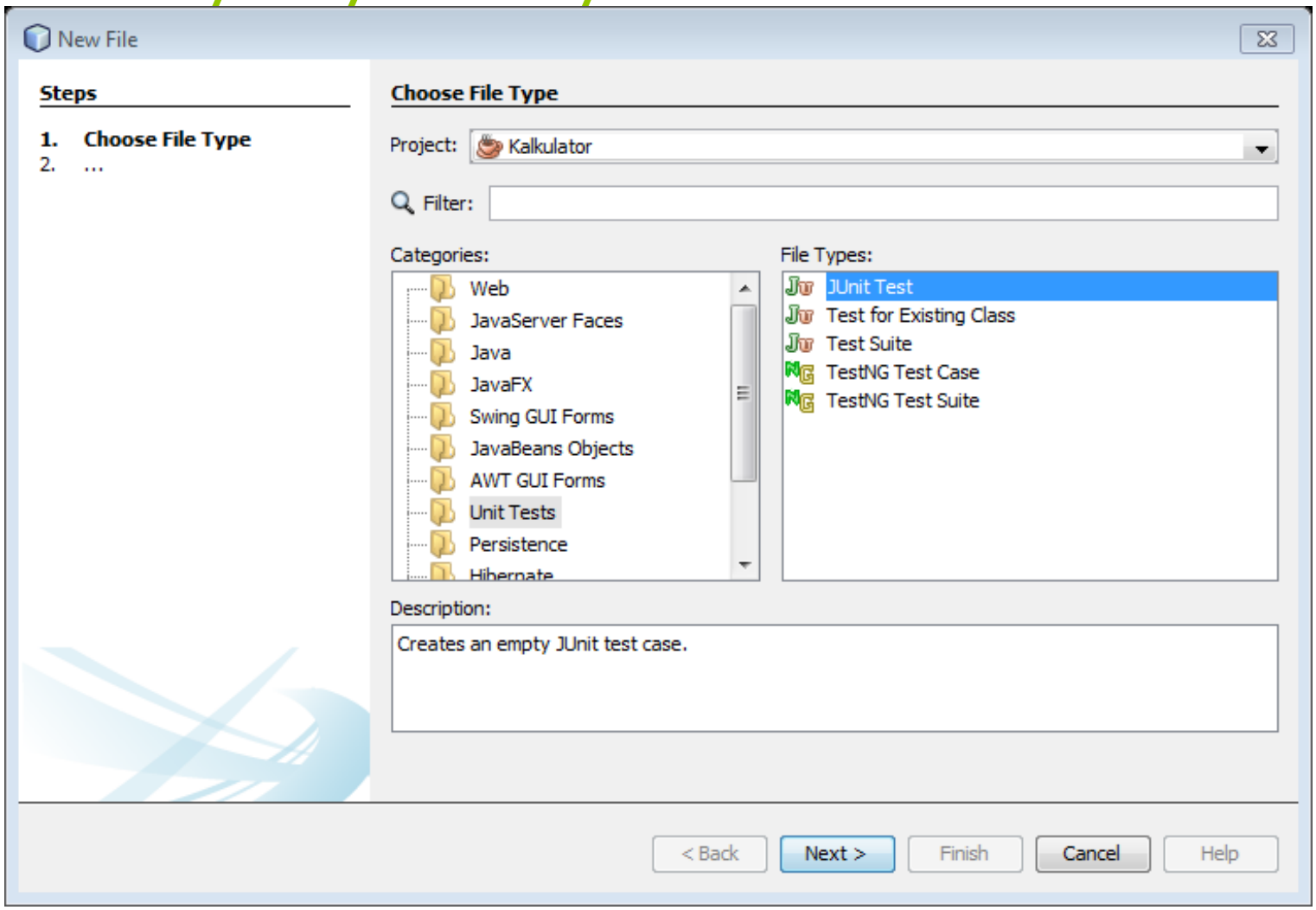
- **JUnit** jest narzędziem służącym do tworzenia powtarzalnych testów jednostkowych oprogramowania pisanego w języku Java.
- Cechy JUnit:
  - metoda najmniejszą jednostką testowania,
  - przypadki testowe,
  - oddzielenie testów od kodu,
  - wiele mechanizmów uruchamiania,
  - tworzenie raportów,
  - integracja z różnymi środowiskami programistycznymi.

http://www.vogella.com/tutorials/JUnit/article.html http://junit.org/

Testujemy klasę

```
12  public class KalkMain {
13
14      double dodaj(double l1, double l2) {
15          return l1 + l2;
16      }
17
18      double odejmij(double l1, double l2) {
19          return l1 - l2;
20      }
21
22      double mnóż(double l1, double l2) {
23          return l1 * l2;
24      }
25
26      double dziel(double l1, double l2) {
27          double wynik = 0;
28          try {
29              wynik = l1 / l2;
30          } catch (ArithmeticException e) {
31              System.out.println("Wyjatek");
32              throw new ArithmeticException();
33          } finally {
35              return wynik;
36          }
37      }
```

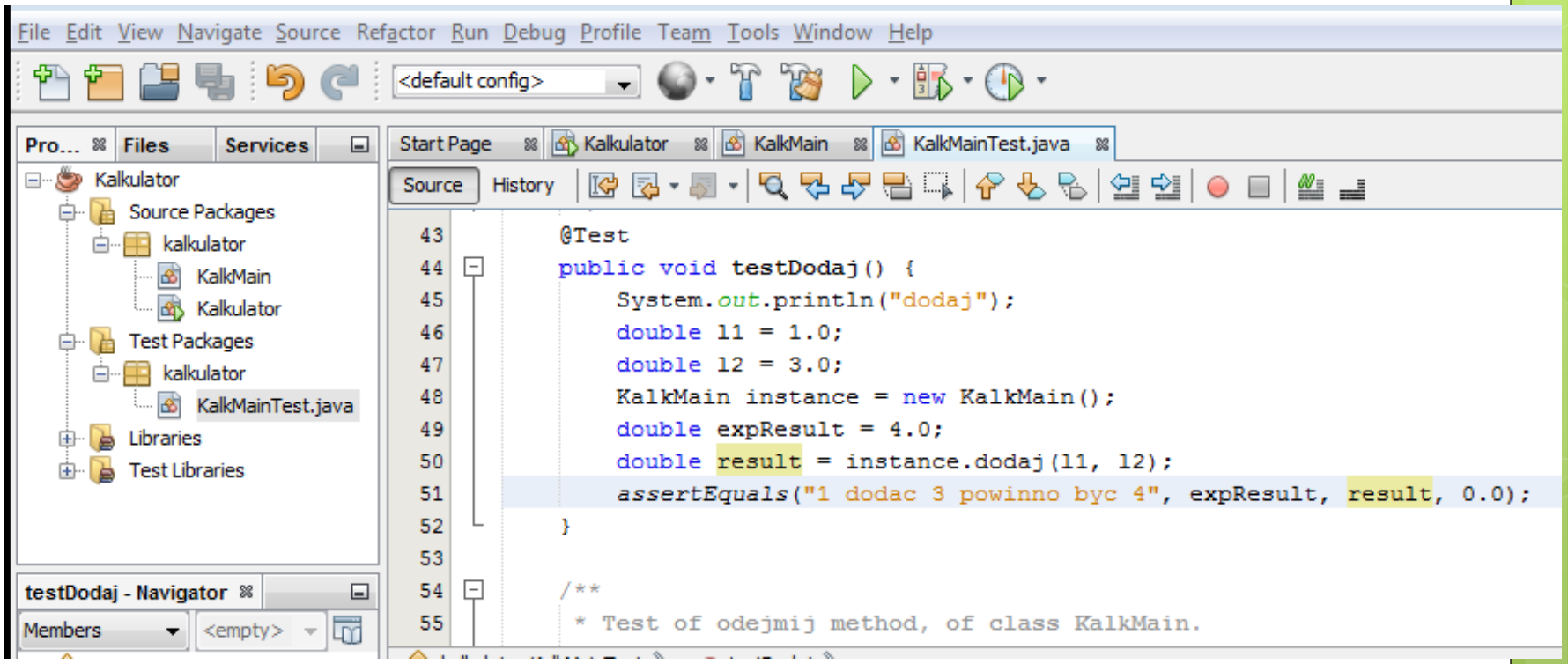
Tworzymy testy



Możliwe testy

@Test public void method()	• W ten sposób oznaczamy metodę do wykonania jako test
@Test (expected = Exception.class)	• Metoda powinna zwrócić wyjątek „Exception”
@Test(timeout=100)	• Metoda powinna się wykonać w mniej niż 100ms
@Before public void method()	• Metoda wykonywana przed każdym testem
@After public void method()	• Metoda wykonywana po każdym teście
@BeforeClass public static void method()	• Metoda wykonywana raz przed wszystkimi testami
@AfterClass public static void method()	• Metoda wykonywana raz po wszystkich testach

Test dla dodawania

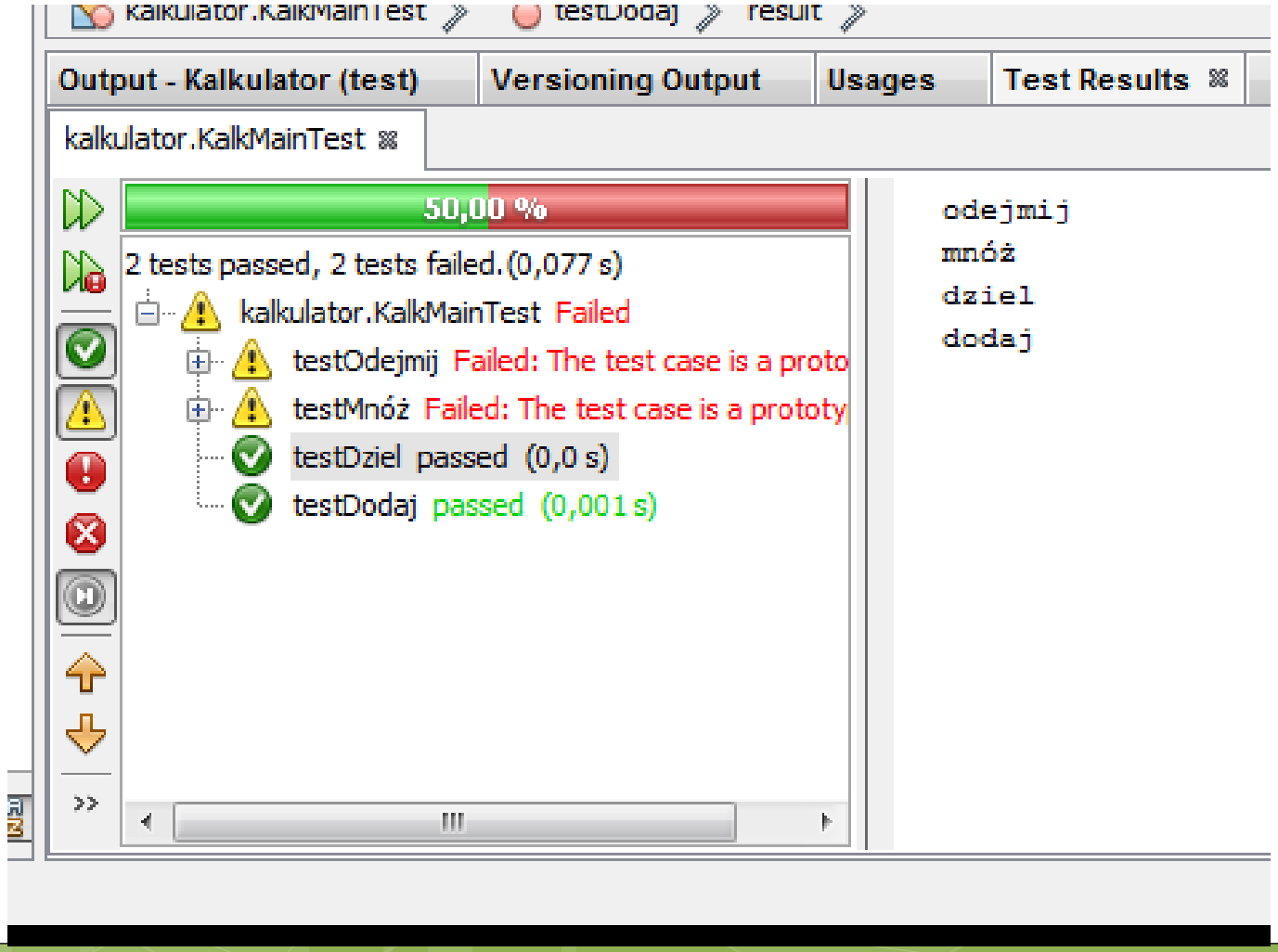


Możliwe asercje

fail()	• Zawsze się nie uda (zwróci test negatywny)
assertTrue([message], boolean condition)	• Sprawdzi, czy warunek jest prawdziwy
assertEquals([String message], expected, actual, tolerance)	• Sprawdzi czy expected == actual +/- tolerance. Do tego ewentualnie wyświetli message
assertNull([message], object)	• Czy obiekt jest nullem
assertSame([String], expected, actual)	• Czy obiekty są takie same

http://junit.sourceforge.net/javadoc/org/junit/Assert.html

Wykonujemy testy



Ćwiczenia

Zad 1

- Napisz aplikację rozwiązującą równanie kwadratowe
- Dopisz do niej testy jednostkowe, wykonaj je i sprawdź działanie

Zad 2

- Napisz aplikację losującą 1E7 liczb i następnie je sortującą.
- Dopisz testy jednostkowe sprawdzające czas wykonania

Zad 3

- Dopisz testy jednostkowe pokrywające co najmniej 75% kodu dla wybranej aplikacji z poprzednich zajęć