



Testy automatyczne

Korzystające z jUnit

Cytaty

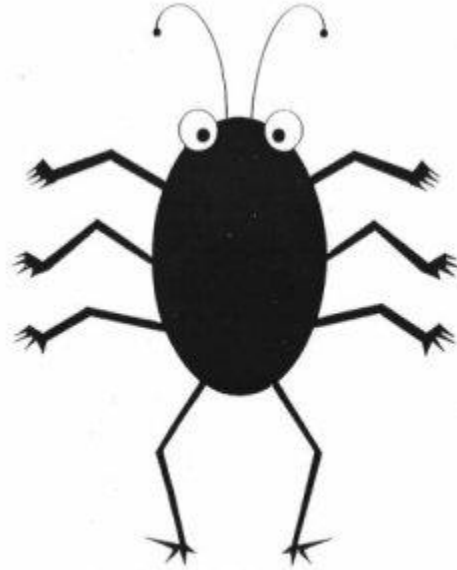
- Kiedy zawiesz się program konkurencji, to jest awaria. Kiedy zawiesz się własny program, to jest „drobiazg”. Często po awarii pojawia się komunikat typu „ID 02”. „ID” to skrót od „idiotyczny drobiazg”, a następująca po nim liczba wskazuje, przez ile miesięcy produkt informatyczny powinien być jeszcze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

Guy Kawasaki, The Macintosh Way

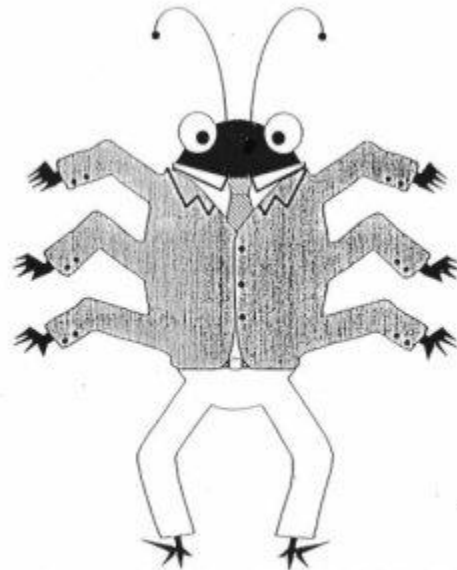
- Uwielbiam ostateczne terminy. Zwłaszcza podoba mi się świst, jaki wydają kiedy nas mijają.

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

Bug, skąd się wziął



BUG



FEATURE

Sukces vs porażka

Sukces

- Wykrycie błędów w procesie testowania

Porażka

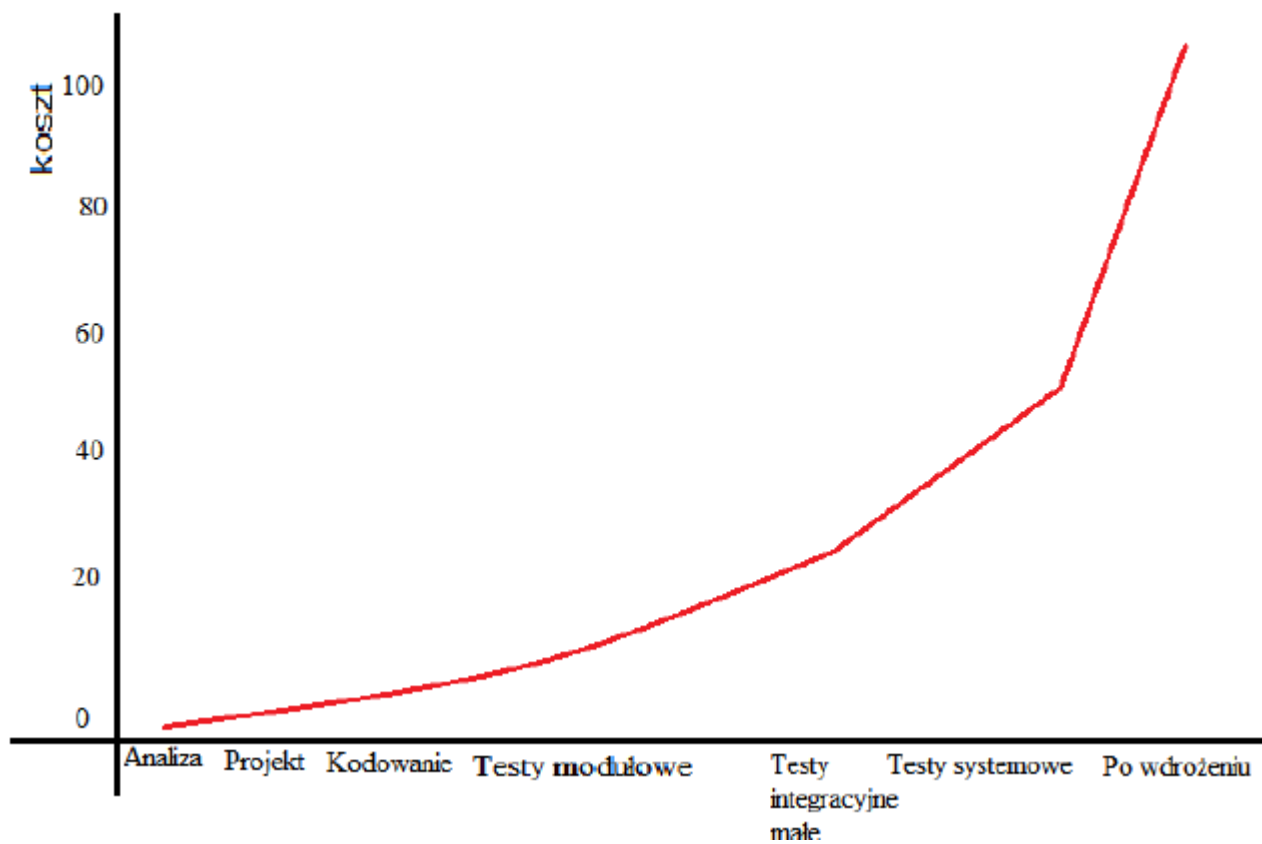
- Sytuacja odwrotna

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

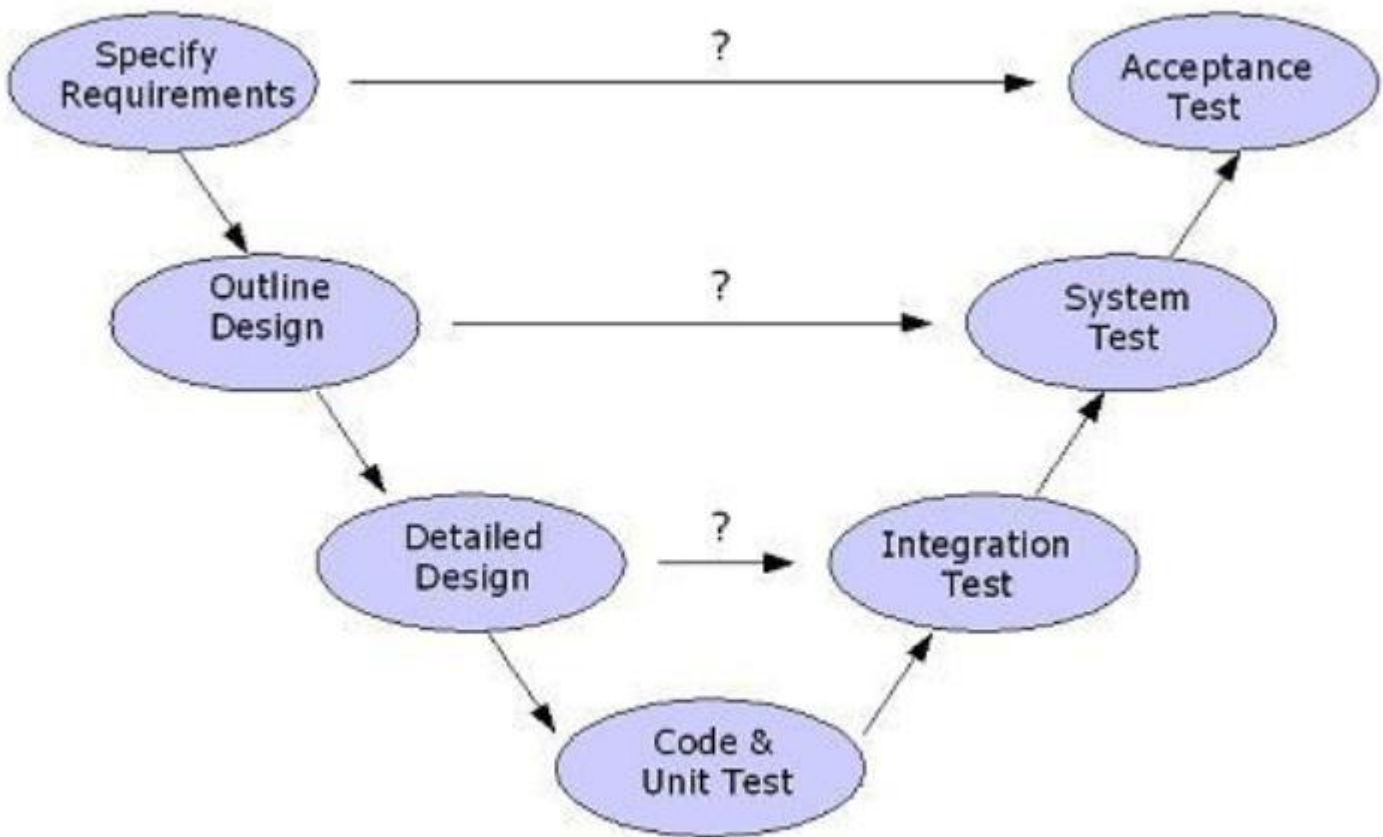
„„Błąd”” oprogramowania

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



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.

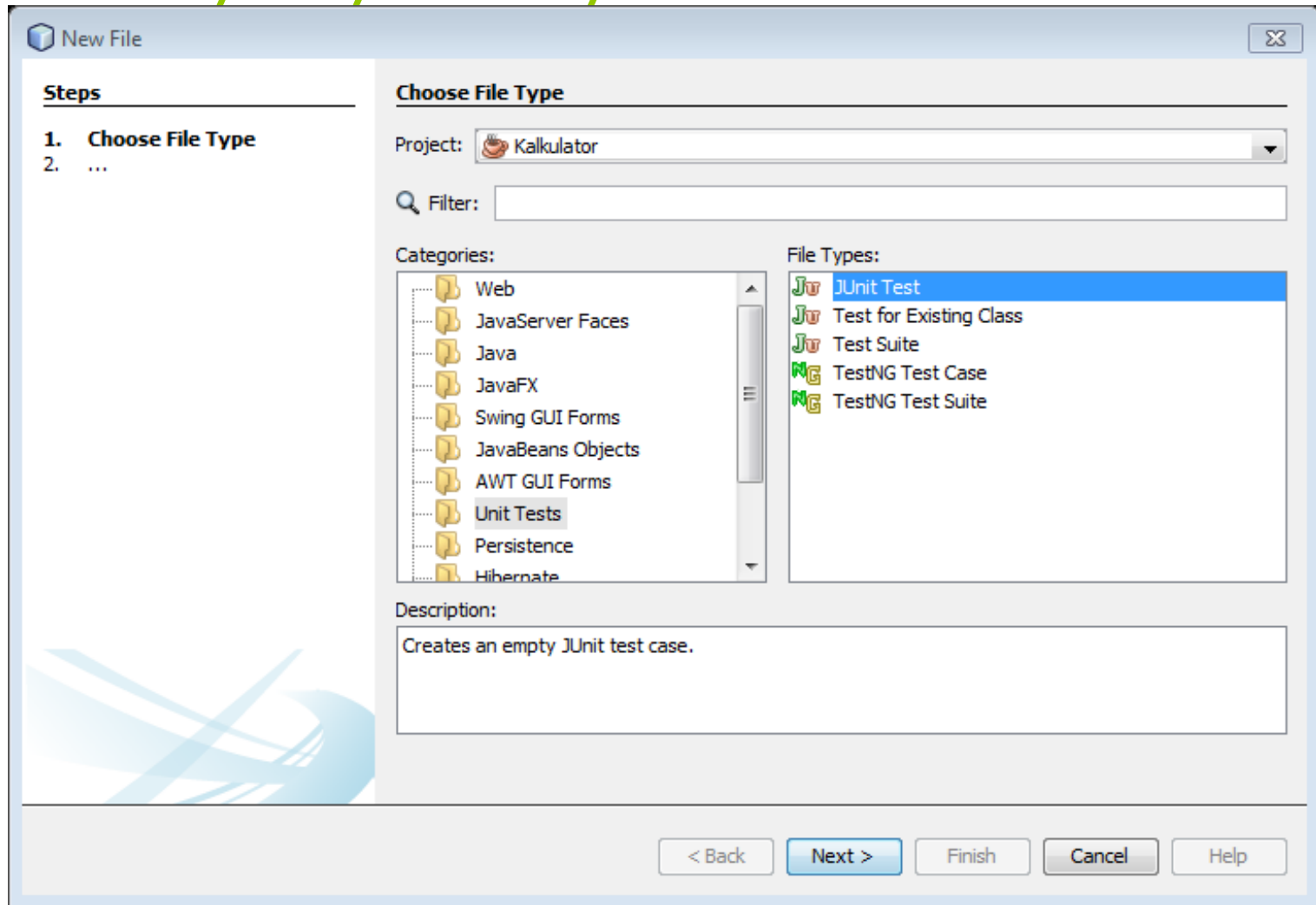
jUnit

- **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.

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 mnoż(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 {
34             return wynik;
35         }
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

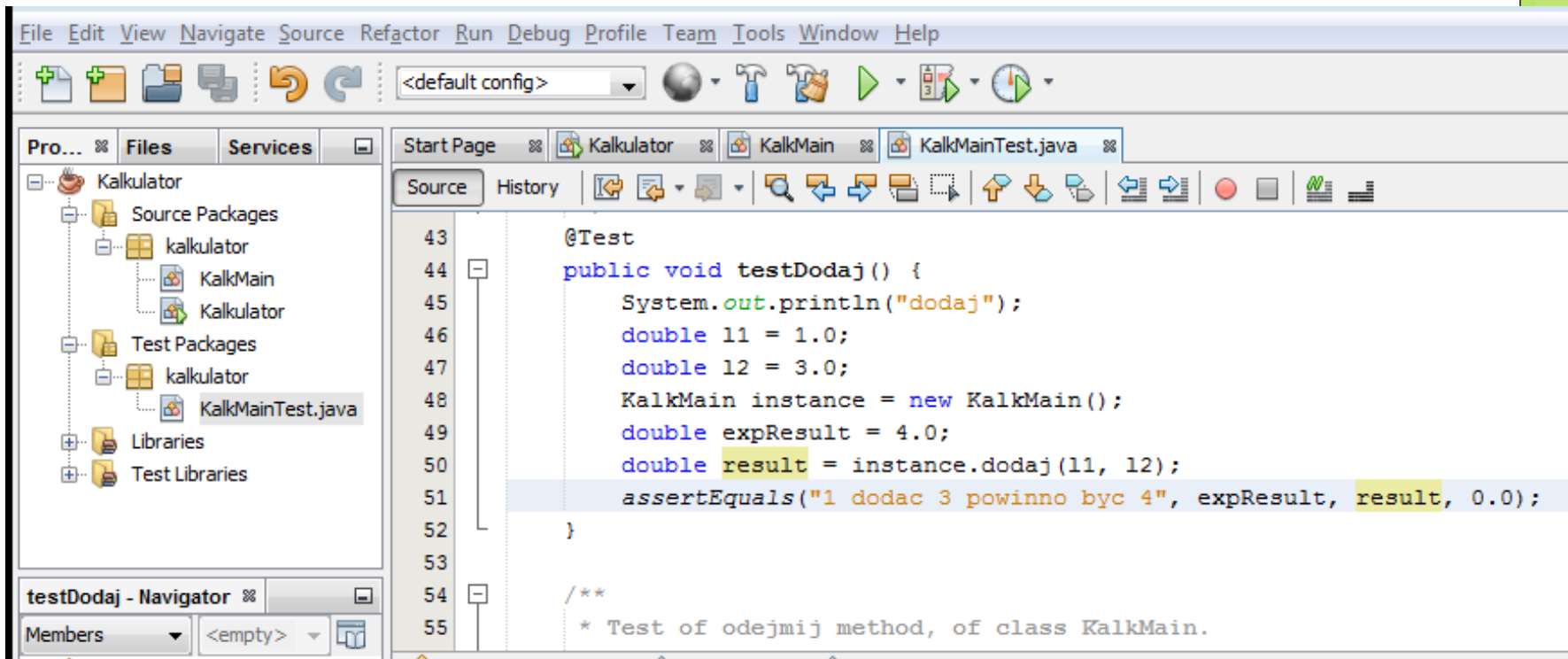
```
@BeforeClass  
public static void method()
```

- Metoda wykonywana raz przed wszystkimi testami

```
@AfterClass  
public static void method()
```

- Metoda wykonywana raz po wszystkich testach

Test dla dodawania



The screenshot shows an IDE window with the following components:

- Menu Bar:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help.
- Toolbar:** Includes icons for file operations, a configuration dropdown set to "<default config>", and execution buttons (run, debug, test).
- Project Explorer:** Shows a project named "Kalkulator" with sub-packages "Source Packages" and "Test Packages". Under "Source Packages", there is a "kalkulator" package containing "KalkMain" and "Kalkulator". Under "Test Packages", there is a "kalkulator" package containing "KalkMainTest.java".
- Source Editor:** Displays the code for "KalkMainTest.java". The code is as follows:

```
43     @Test
44     public void testDodaj() {
45         System.out.println("dodaj");
46         double l1 = 1.0;
47         double l2 = 3.0;
48         KalkMain instance = new KalkMain();
49         double expectedResult = 4.0;
50         double result = instance.dodaj(l1, l2);
51         assertEquals("1 dodac 3 powinno byc 4", expectedResult, result, 0.0);
52     }
53
54     /**
55      * Test of odejmij method, of class KalkMain.
```
- testDodaj - Navigator:** Shows "Members" and "<empty>".

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

The screenshot shows the Test Results window in an IDE. At the top, there are tabs for 'Output - Kalkulator (test)', 'Versioning Output', 'Usages', and 'Test Results'. The active tab is 'Test Results'. Below the tabs, the test runner 'kalkulator.KalkMainTest' is shown with a progress bar at 50,00%. The test results are as follows:

- 2 tests passed, 2 tests failed. (0,077 s)
- kalkulator.KalkMainTest Failed
 - testOdejmij Failed: The test case is a proto
 - testMnóz Failed: The test case is a prototy
 - testDziel passed (0,0 s)
 - testDodaj passed (0,001 s)

On the right side of the window, the following text is displayed:

```
odejmij  
mnóz  
dziel  
dodaj
```


Ćwiczenia

Zad 1

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

Zad 2

- Napisz aplikację losującą 10^7 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