Pomiar jakości kodu, testy jednostkowe, TDD i BDD

Łukasz Porębski

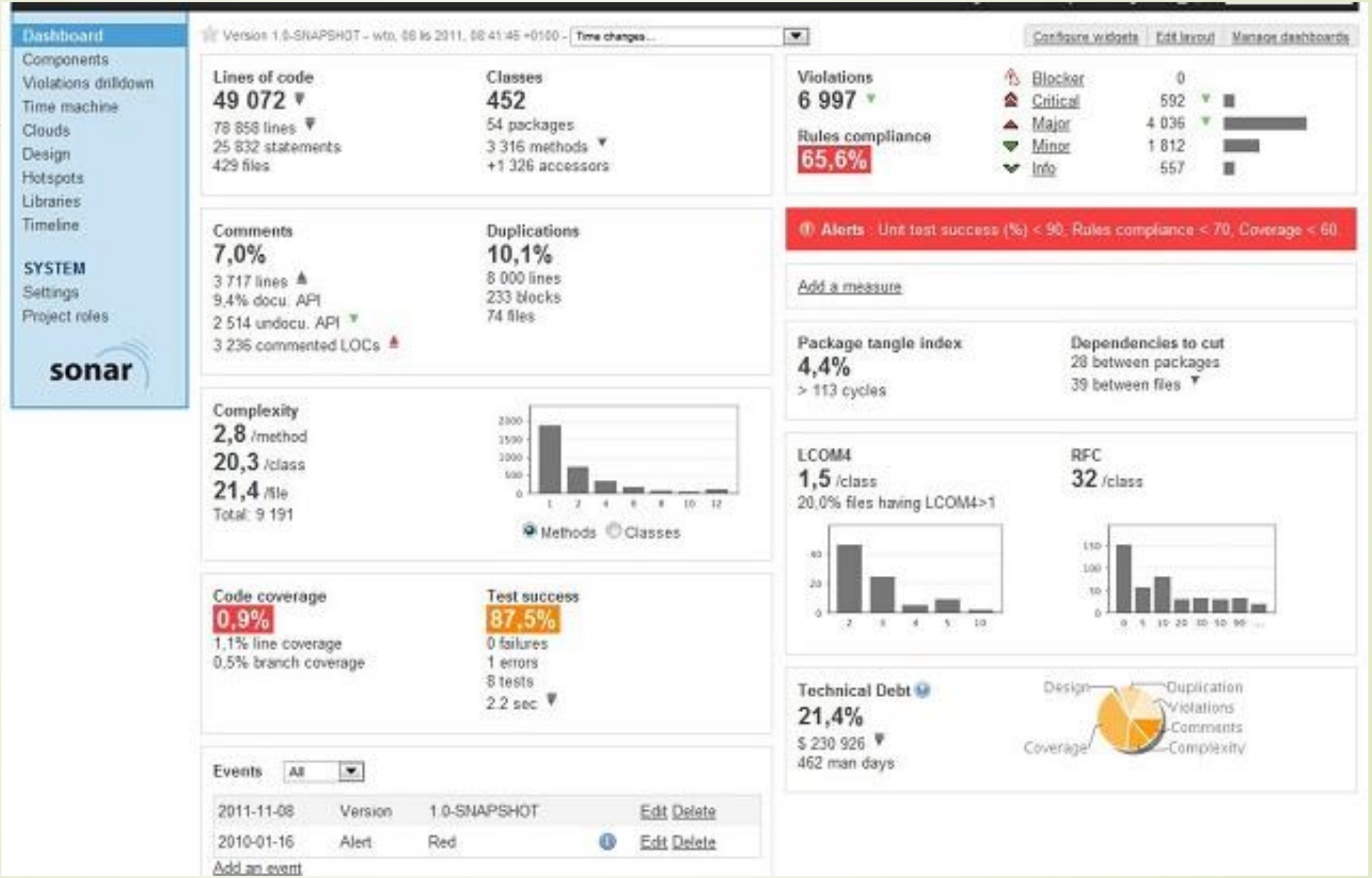
Co zaliczamy do pomiaru jakości kodu?

Jakość kodu można mierzyć na wiele sposobów. Kilka z nich to:

- pokrycie kodu testami automatycznymi
- pokrycie kodu komentarzami dokumentującymi(w tym jakość i kompletność tych komentarzy)
- pomiar złożoności kodu (algorytmy zliczające ilość pętli, instrukcji warunkowych, itp. na każdą funkcję)
- testy ilościowe: ilość klas, ilość linii kodu, itp.
- testy wykrywające nadużycia, złe praktyki oraz inne niezgodności ze standardami kodu

Źródło: https://www.3e.pl/pl/cms/case-studies/pomiar-jakosci-kodu.html

Program Sonar



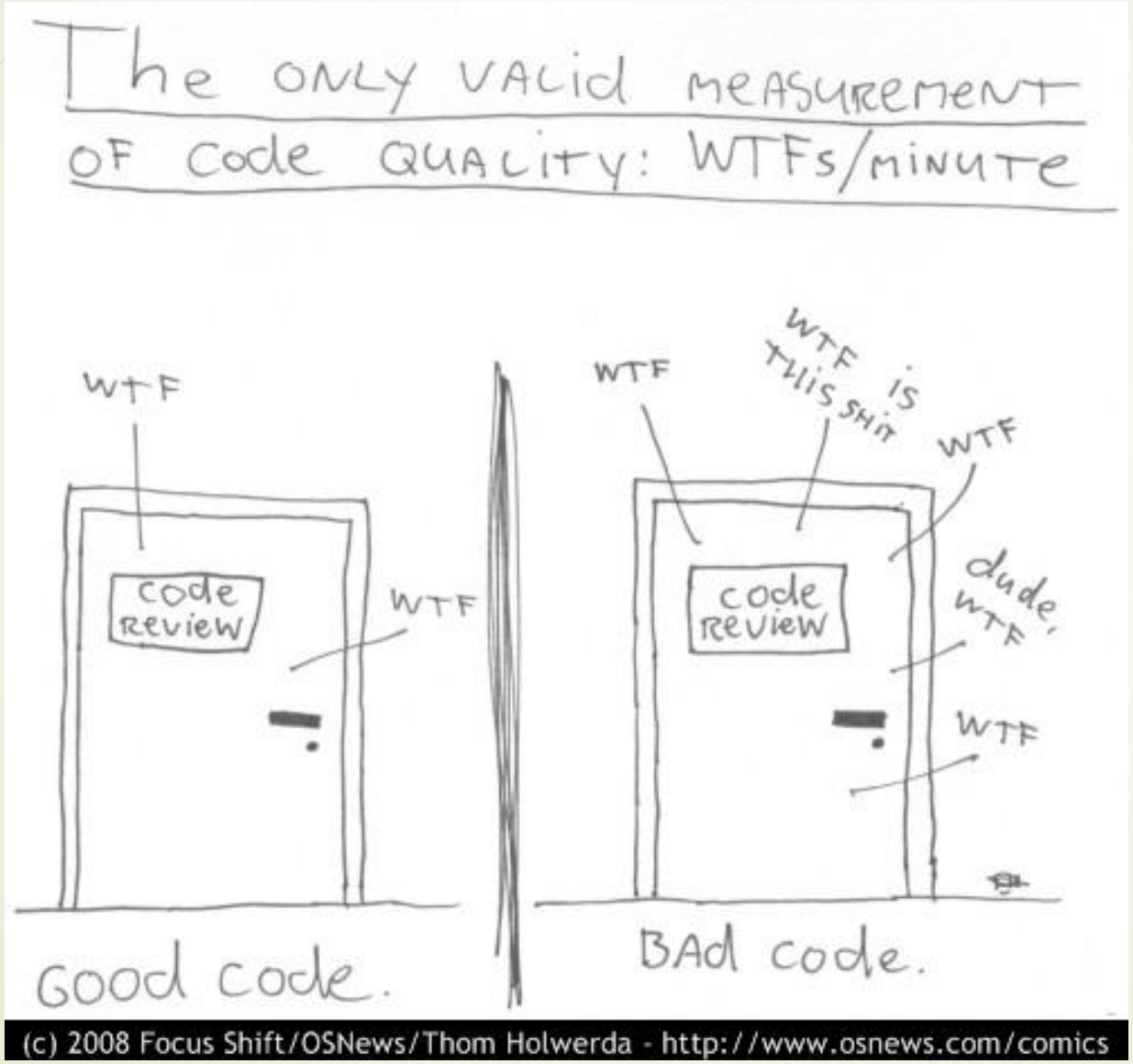
Źródło: https://www.3e.pl/pl/cms/case-studies/pomiar-jakosci-kodu.html

Czym jest code review?

Polega na sprawdzeniu całości kodu bądź jego fragmentu przez innego członka zespołu, zarówno pod względem ogólnej poprawności, jak i funkcjonalności. Przede wszystkim jednak praktyka ta pozwala na zastosowanie najlepszych z możliwych rozwiązań.

Źródło: https://www.webchefs.pl/blog/code-review-co-to-takiego/

Inna definicja Code review



Zalety code review

- Poprawa umiejętności programistów
- Większa czytelność kodu
- Wymiana wiedzy w zespole
- Wyłapanie błędów i literówek
- Współwłasność kodu
- Autorefleksja

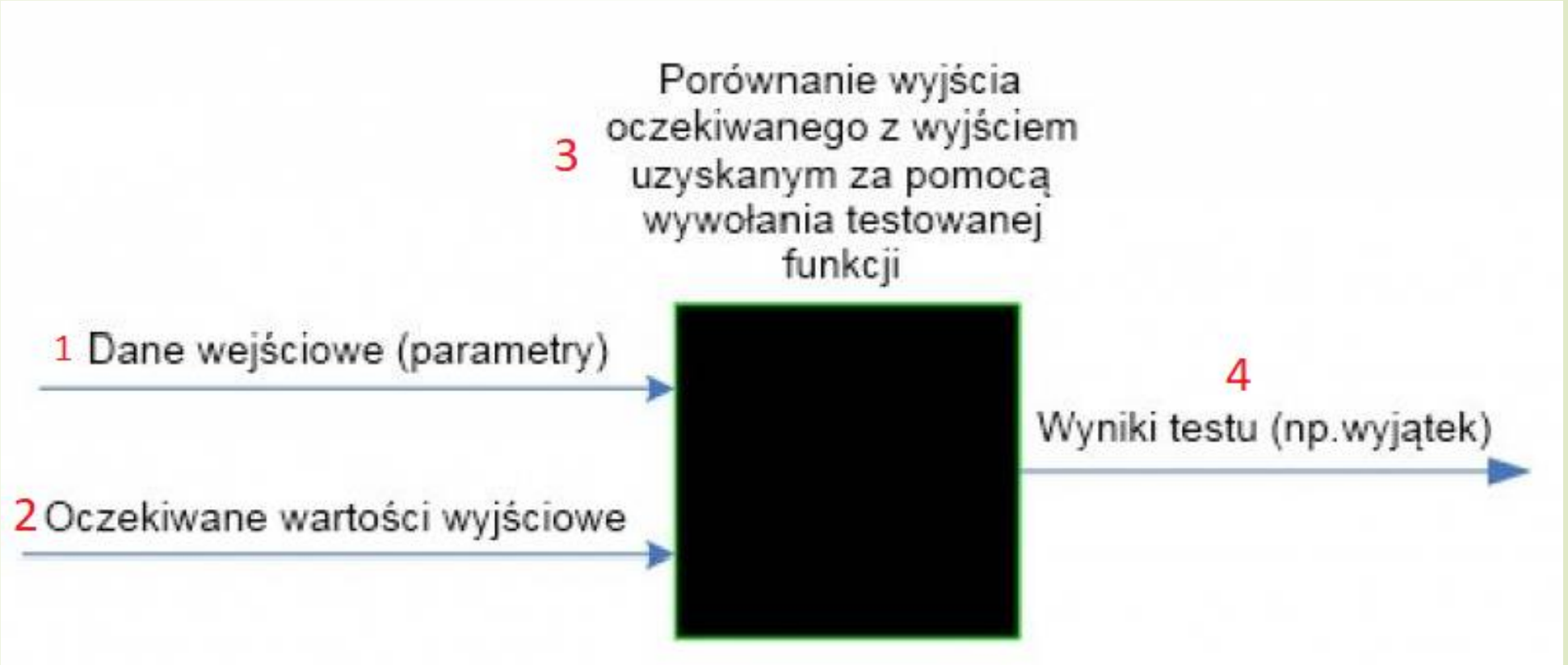
Źródła: https://teamquest.pl/blog/523_po-co-komu-code-review
http://webappcraft.blogspot.com/2017/04/code-review.html

Testy jednostkowe

Trochę teorii

Test jednostkowy są **kodem wykonujący inny kod w kontrolowanych warunkach**. Jego zadaniem jest weryfikacja (bez ingerencji programisty), że testowany kod działa poprawnie.

Są one przeprowadzane w kilku prostych krokach:



Jeden test jednostkowy powinny badać **bada jedną ścieżkę wykonania jednej metody**.

Źródła: http://www.is.umk.pl/~grochu/wiki/doku.php?id=zajecia:znd:unit_test

https://devstyle.pl/2011/08/11/ut-1-co-to-sa-testy-i-po-co-sa-testy-jednostkowe/

Po co pisać testy jednostkowe?

- Wczesne wykrywanie błędów
- Odporność oprogramowania na błędy regresyjne, czyli błędy powstałe w wyniku poprawek kodu
- Ułatwienie refaktoringu i zmian w kodzie pokrytym testami
- Dokumentowanie i wyjaśnianie kodu. Test wyjaśnia jaką funkcjonalność realizuje jednostka kodu i jak należy jej używać.
- Lepiej zaprojektowane interfejsy i API. Testy zmuszają do lepszego przemyślenia rozwiązań i dokładnego określenia jakie zadania dana metoda ma wykonywać.
- Automatyzacja i powtarzalność: testy można uruchamiać regularnie o określonych porach lub na pewnych etapach produkcji. Oszczędność czasu w stosunku do ręcznego testowania.
- Możliwość przetestowania funkcjonalności bez uruchamiania całego oprogramowania
- Poprawienie architektury tworzonej aplikacji.

Źródło: http://www.is.umk.pl/~grochu/wiki/doku.php?id=zajecia:znd:unit_test

Pięć zasad dobrych testów jednostkowych

F jak **szybkie** (ang. fast).
Jeżeli przeprowadzanie testów trwa zbyt długo, nie chce się tego robić.

Czysty kod. Podręcznik dobrego programisty
R.C. Martin
Helion 2010
s. 151

I jak **niezależne** (ang. independent).
Powinno być możliwe przeprowadzanie testów w dowolnej kolejności i konfiguracji. Wprowadzenie zależności między testami ukrywa problemy i utrudnia ich diagnozę.

R jak **powtarzalne** (ang. repeatable).
Testy powinno dać się przeprowadzić w każdym środowisku i powinny dawać te same efekty.

S jak **samosprawdzające** (ang. self-validating).
Testy powinny dawać jeden rezultat „tak-nie".

T jak **na czas** (ang. timely).
Testy powinny być pisane w odpowiednim momencie.

Jaki to jest „odpowiedni" moment?
Pełna odpowiedź innym razem.
Teraz powiem tylko, że chodzi o pisanie testów **przed** napisaniem kodu produkcyjnego.

Źródło: https://github.com/oleklamza/pio-wyklady/blob/master/pio.5.testowanie.pdf

Testy jednostkowe VS testy integracyjne

Podstawową różnicą między obydwoma rodzajami testów jest to, że testy jednostkowe testują pojedyncza część kodu, natomiast testy integracyjne mają na celu sprawdzenie kilka komponentów działających razem.

W teście jednostkowym wszystkie zależności powinny być zastąpione przez tzw. *mock objects*, które symulują ich zachowanie.

W teście integracyjnym możemy skorzystać z kilku lub wszystkich zależności systemu.

Źródło: https://dariuszwozniak.net/posts/kurs-tdd-2-testy-jednostkowe-a-testy-integracyjne/

Testy jednostkowe VS testy integracyjne

Zagadnienie	Test jednostkowy	Test integracyjny
Zależności	Testowany jednostkowy element (klasa, metoda) w izolacji.	Testowana więcej niż jedna wewnętrzna lub zewnętrzna zależność.
Punkt awarii	Tylko jeden potencjalny punkt awarii (jedna logiczna asercja per test*).	Wiele potencjalnych punktów awarii.
Szybkość działania	Bardzo szybko, dużo poniżej 1 sekundy.	Może trwać długo, ze względu na czasochłonne operacje np. dostęp do bazy danych, I/O, operacje na sesji.
Konfiguracja	Test musi działać na każdej maszynie bez dodatkowej konfiguracji.	Test może być zależny od konfiguracji, np. machine.config (login/hasło) do bazy danych.

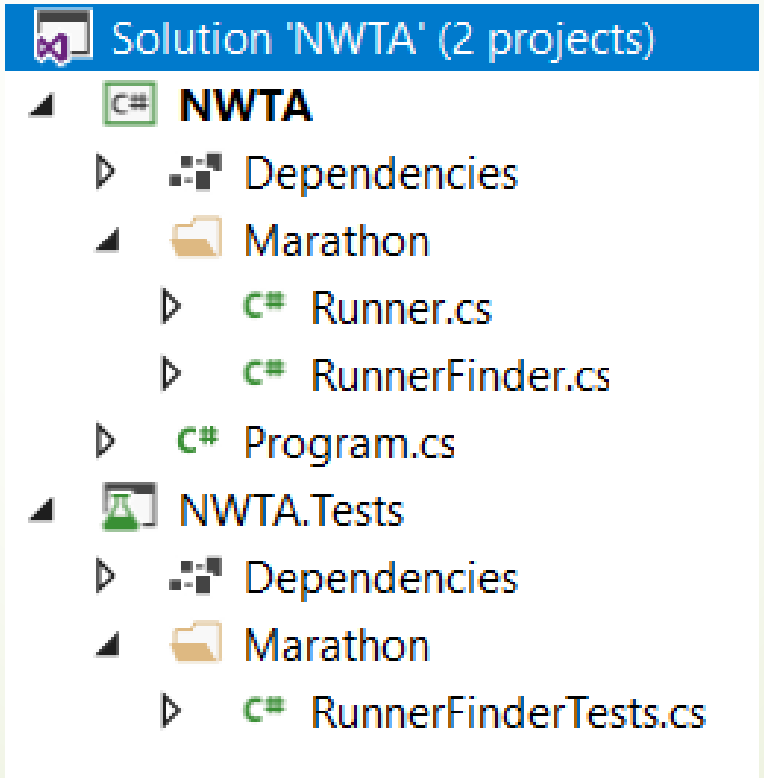
* test jednostkowy może zawierać więcej niż jeden Assert pod warunkiem, że wszystkie testują jeden element

Źródło: https://dariuszwozniak.net/posts/kurs-tdd-2-testy-jednostkowe-a-testy-integracyjne/

Kilka praktycznych porad

Struktura

- Dla każdego projektu w aplikacji w której jest zawarta jakaś logika panien być osobny projekt zawierający testy.
- Projekt testowy powinien być zakończony słowem Tests
- Struktura projektu testowego powinna być tożsama ze strukturą projektu testowanego
- Klasy testujące powinny mieć taką samą nazwę jak klasy testowane oraz dopisek Tests



Struktura

Strukturę testu jednostkowego definiuje zasada Arrange–Act–Assert (AAA):

- Arrange - wszystkie dane wejściowe,
- Act - działanie na metodzie/funkcji/klasie testowanej,
- Assert - upewnienie się, że zwrócone wartości są zgodne z oczekiwanymi.

Jakie korzyści płyną ze stosowania tego wzorca?

Przede wszystkim porządek; wzorzec zapewnia logiczny porządek w pojedynczym teście — część przygotowania danych wejściowych jest odseparowana od części weryfikacyjnej.

Ponadto, nie mieszamy naszych asercji w trakcie wywołania testowanego obiektu.

Źródło: https://dariuszwozniak.net/posts/kurs-tdd-3-struktura-test-czyli-arrange-act-assert/

```
[Test]
0 references
public void Should_ConcatStartNumberAndRunnersName_When_Called()
{
    //Arrange
    var runner = new Runner { Name = "Tomek", StartNumber = 15 };

    //Act
    var result = _runnerFinder.ConcatStartNumberAndRunnersName(runner);

    //Assert
    result.Should().Be( expected: "15-Tomek");
}
```

Nazewnictwo metod testowych

- CoSorawdzamy_OczekiwanyRezultat
- NazwaMetody_CoSprawdzamy_OczekwianyRezultat
- NazwaMetody_OczekiwanyRezultat_CoSprawdzamy
- Should_OczekiwanyRezultat_When_CoSprawdzamy
- FunkcjaDoPrzetestowania (np. IsNotAccesDeniedIfAgeOfUserLessThan18)
- When_CoSprawdzamy_Then_NazwaMetody_Should_OczekiwanyRezultat
- When_CoSprawdzamy_Expect_OczekiwanyRezultat
- Given_WarunkiWstepne_When_CoSprawdzamy_Then_OczekwianyRezultat

Źródło: http://pisz-kod.pl/2018/06/21/nazewnictwo-testow-budowa-testu/

Czego nie testować?

- Wygenerowanego kodu (gettery, setery, konstruktory, *.designer.cs itp.)
- Klas mających wiele zależności i zajmujących się głównie przekazywaniem obiektów między nimi
- Metod prywatnych
- Fasad Frameworków, bibliotek itp.
- Kodu nie zawierającego logiki (konfiguracje, modele, encje itp.)
- Kontrolerów

Źródło: https://devstyle.pl/2011/11/30/ut-4-co-testowac-a-czego-nie-testowac/

Co testować?

CAŁĄ RESZTĘ

Źródło: https://devstyle.pl/2011/11/30/ut-4-co-testowac-a-czego-nie-testowac/

Trochę kodu

Testowana klasa

```
12 references
public class Runner
{
    7 references
    public string Name { get; set; }
    8 references
    public int StartNumber { get; set; }
}
```

```
3 references
public class RunnerFinder
{
    private const int StartNumberLength = 4;

    4 references
    public List<string> FindRunnersWhoseStartNumberContainsGivenNumber
        (IReadOnlyCollection<Runner> runners, int startNumber)
    {
        if (runners == null || !runners.Any())
            throw new ArgumentException(message:"Runners collection cannot be empty.");

        var foundRunners = runners.Where(
            runner => runner.StartNumber.ToString().Contains(startNumber.ToString()));

        var foundRunnersList = new List<string>();
        foreach (var foundRunner in foundRunners)
            foundRunnersList.Add(ConcatStartNumberAndRunnersName(foundRunner));

        return foundRunnersList;
    }

    3 references
    public string ConcatStartNumberAndRunnersName(Runner runner)
        => $"{runner.StartNumber}-{runner.Name}";
}
```

Testy 1/5

```
0 references
public class RunnerFinderTests
{
    private RunnerFinder _runnerFinder;

    [SetUp]
    0 references
    public void Setup()
    {
        _runnerFinder = new RunnerFinder();
    }

    [Test]
    0 references
    public void Should_ConcatStartNumberAndRunnersName_When_Called()
    {
        //Arrange
        var runner = new Runner { Name = "Tomek", StartNumber = 15 };

        //Act
        var result = _runnerFinder.ConcatStartNumberAndRunnersName(runner);

        //Assert
        result.Should().Be( expected: "15-Tomek");
    }
```

Testy 2/5

```
[Test]
0 references
public void Should_ReturnRunnersWhoseStartNumberContainsGivenNumber_When_GivenNumberIs77()
{
    var runners = GetRunners();
    var expectedResult = new List<string>
    {
        "77-Ola",
        "177-Tomek",
        "774-Wojtek"
    };

    var result = _runnerFinder.FindRunnersWhoseStartNumberContainsGivenNumber(
        runners, startNumber: 77);

    result.Should().HaveCount(expected: 3);
    result.Should().BeEquivalentTo(expectedResult);
}
```

Testy 3/5

```
[Test]
0 references | Lukasz, 6 days ago | 1 author, 1 change
public void Should_ReturnEmptyList_When_ThereAreNoRunnersWhoseStartNumberContainsGivenNumber(
{
    var runners = GetRunners();
    var startNumber = 77;
    var expectedResult = new List<string>
    {
        "77-Ola",
        "177-Tomek",
        "774-Wojtek"
    };

    var result = _runnerFinder.FindRunnersWhoseStartNumberContainsGivenNumber(
        runners, startNumber);

    result.Should().HaveCount(expected: 3);
    result.Should().BeEquivalentTo(expectedResult);
}
```

Testy 4/5

```
[Test]
0 references | Lukasz, 6 days ago | 1 author, 1 change
public void Should_ThrowArgumentException_When_RunnersCollectionIsNull()
{
    IReadOnlyCollection<Runner> runners = null;
    var startNumber = 0;

    Action act = () => _runnerFinder
        .FindRunnersWhoseStartNumberContainsGivenNumber(runners, startNumber);

    act.Should().Throw<ArgumentException>();
}

[Test]
0 references | Lukasz, 6 days ago | 1 author, 1 change
public void Should_ThrowArgumentException_When_RunnersCollectionIsEmpty()
{
    var runners = new List<Runner>();
    var startNumber = 0;

    Action act = () => _runnerFinder
        .FindRunnersWhoseStartNumberContainsGivenNumber(runners, startNumber);

    act.Should().Throw<ArgumentException>();
}
```

Testy 5/5

```
2 references | Lukasz, 6 days ago | 1 author, 1 change
private IReadOnlyCollection<Runner> GetRunners()
{
    return new List<Runner>
    {
        new Runner{Name = "Ola", StartNumber = 77},
        new Runner{Name = "Tomek", StartNumber = 177},
        new Runner{Name = "Wojtek", StartNumber = 774},
        new Runner{Name = "Krysia", StartNumber = 5},
    };
}
```

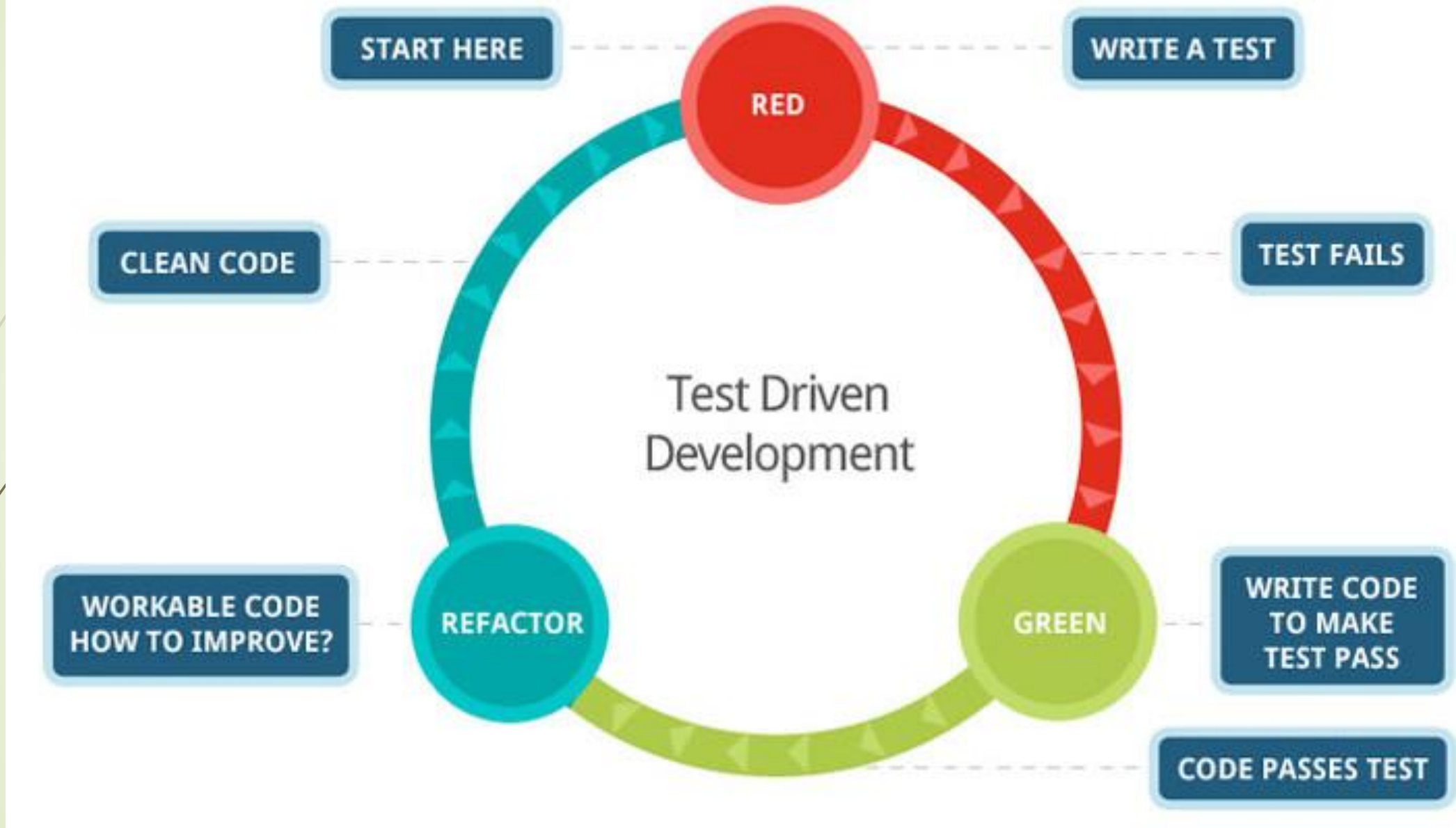
TDD

Czym jest TDD

TDD (Test Driven Development) to podejście do tworzenia oprogramowania, które zakłada, że przed napisaniem właściwej funkcjonalności programista zaczyna od utworzenia testu.

Test ten powinien testować funkcjonalność, którą dopiero chcemy napisać.

Źródło: https://www.samouczekprogramisty.pl/test-driven-development-na-przykladzie/



Źródło: https://haselt.com/blog/coding-dojo-with-tdd/PIC_featured.jpg

Faza Red

Pierwszym krokiem jest napisanie testu. Test ten nie może się powieść, ponieważ sama funkcjonalność jeszcze nie jest zaimplementowana. Możliwe, że nawet po napisaniu takiego testu kod nie będzie się kompilował. Może się tak stać w przypadku, gdy napisałeś test dla metody, która jeszcze nie istnieje.

Źródło: https://www.samouczekprogramisty.pl/test-driven-development-na-przykladzie/

Faza Green

Kolejnym krokiem jest napisanie kodu, który implementuje brakującą funkcjonalność. W tym momencie istotne jest to aby ten kod nie był „idealny”. Chodzi o możliwe jak najszybszą implementację, która spełnia założenia testu, który był napisany w poprzedniej fazie.

Następnie potwierdzamy to, że nasza implementacja działa jak powinna uruchamiając testy jednostkowe. Jeśli wszystko jest w porządku całość powinna zakończyć się testami jednostkowymi, które przechodzą.

Ważne jest aby w tej fazie uruchamiać wszystkie dotychczas napisane testy jednostkowe.

Źródło: https://www.samouczekprogramisty.pl/test-driven-development-na-przykladzie/

Faza Refactor

Refaktoryzacja to proces, w którym zmieniamy kod w taki sposób, że nie zostaje zmieniona jego funkcjonalność. Mówi się o „oczyszczaniu" kodu, doprowadzaniu go do lepszego stanu. Przykładem refaktoryzacji może być wydzielenie oddzielnej metody, która usuwa powielony kod czy stworzenie zupełnie nowej klasy odpowiedzialnej za pewną część zadań danej klasy.

Jest to ostatnia z trzech faz cyklu TDD. Może się zdarzyć, że faza refaktoryzacji nie zawsze jest konieczna. Usprawnianie dobrego kodu na siłę nie koniecznie może prowadzić do dobrych rezultatów.

Źródło: https://www.samouczekprogramisty.pl/test-driven-development-na-przykladzie/

BDD

Czym jest BDD?

BDD (Behavior Driven Development) jest procesem wytwarzania oprogramowania w oparciu o konkretną strukturę formułowania wymagań.

Cała aplikacja budowana jest z komponentów, małymi historyjkami które opowiadają o tym jak powinien zachować się program w określonym scenariuszu – z ang. stories.

Każde story budowany jest na schemacie given,when,then

Źródło: https://www.kainos.pl/blog/testowanie-w-duchu-bdd/

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

Źródło: https://github.com/oleklamza/pio-wyklady/blob/master/pio.6.przeglad_technik.pdf

Trochę kodu

Scenariusz

```
1   Feature: Calculator
2           In order to avoid silly mistakes
3           As a math idiot
4           I want to be told the sum of two numbers
5
6   @mytag
7   Scenario: Add two numbers
8           Given I have entered 50 into the calculator
9           And I have also entered 70 into the calculator
10          When I press add
11          Then the result should be 120 on the screen
```

Testy 1/2

```
[Binding]
0 references | Lukasz, 6 days ago | 1 author, 2 changes
public class CalculatorSteps
{
    private int _result;
    private readonly Calculator _calculator = new Calculator();

    [Given(@"I have entered (.*) into the calculator")]
    0 references | Lukasz, 6 days ago | 1 author, 1 change
    public void GivenIHaveEnteredIntoTheCalculator(int number)
    {
        _calculator.FirstNumber = number;
    }

    [Given(@"I have also entered (.*) into the calculator")]
    0 references | Lukasz, 6 days ago | 1 author, 1 change
    public void GivenIHaveAlsoEnteredIntoTheCalculator(int number)
    {
        _calculator.SecondNumber = number;
    }
```

Testy 2/2

```
[When(@"I press add")]
0 references | Lukasz, 6 days ago | 1 author, 1 change
public void WhenIPressAdd()
{
    _result = _calculator.Add();
}

[Then(@"the result should be (.*) on the screen")]
0 references | Lukasz, 6 days ago | 1 author, 1 change
public void ThenTheResultShouldBeOnTheScreen(int expectedResult)
{
    Assert.AreEqual(expectedResult, actual: _result);
}
```

Implementacja

```
2 references | Lukasz, 6 days ago | 1 author, 1 change
public class Calculator
{
    2 references | Lukasz, 6 days ago | 1 author, 1 change
    public int FirstNumber { get; set; }
    2 references | Lukasz, 6 days ago | 1 author, 1 change
    public int SecondNumber { get; set; }

    1 reference | Lukasz, 6 days ago | 1 author, 1 change
    public int Add()
    {
        return FirstNumber + SecondNumber;
    }
}
```

KONIEC