# Krzysztof Owsiany

Twitter: **@k_owsiany**

Blog: **MrDev.pl**

Podcast: **After.conf**

# Agenda

- Testy automatyczne
- TDD
- GIT Flow
- Warsztaty
- Do kodu!

# Testy automatyczne

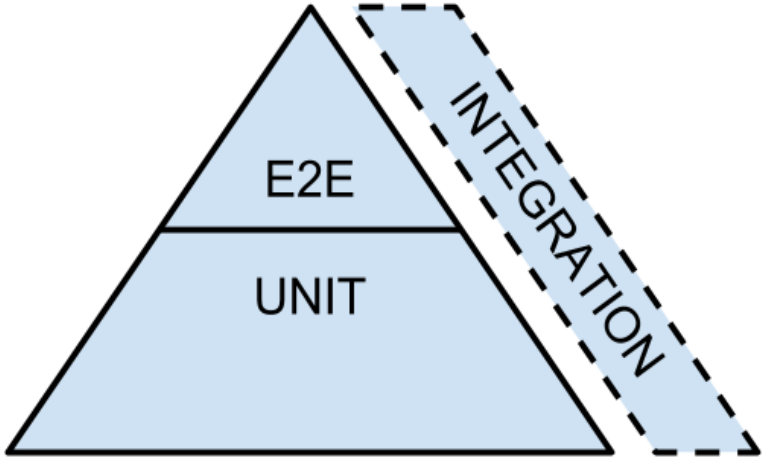
**program** sprawdza **program**

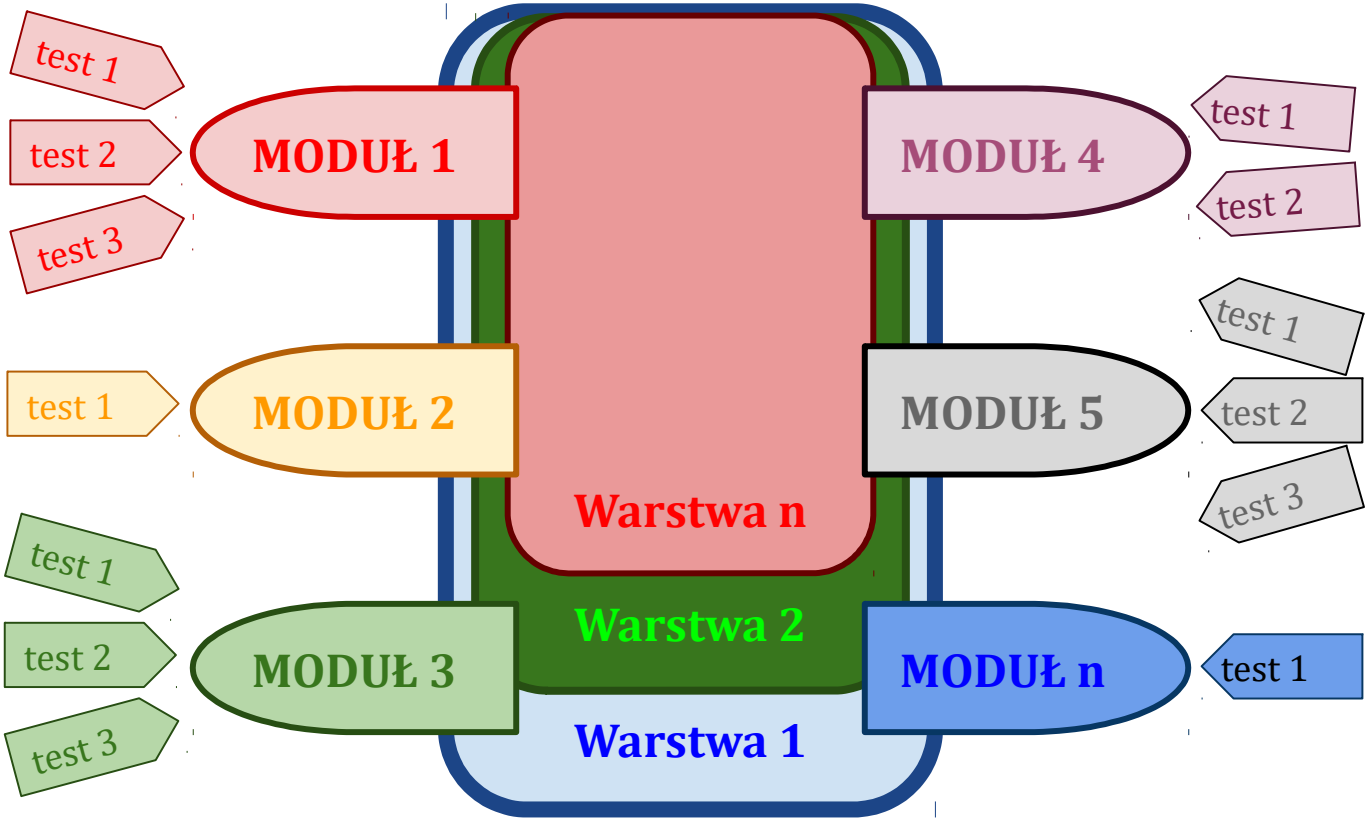
**test** sprawdza **funkcjonalność**

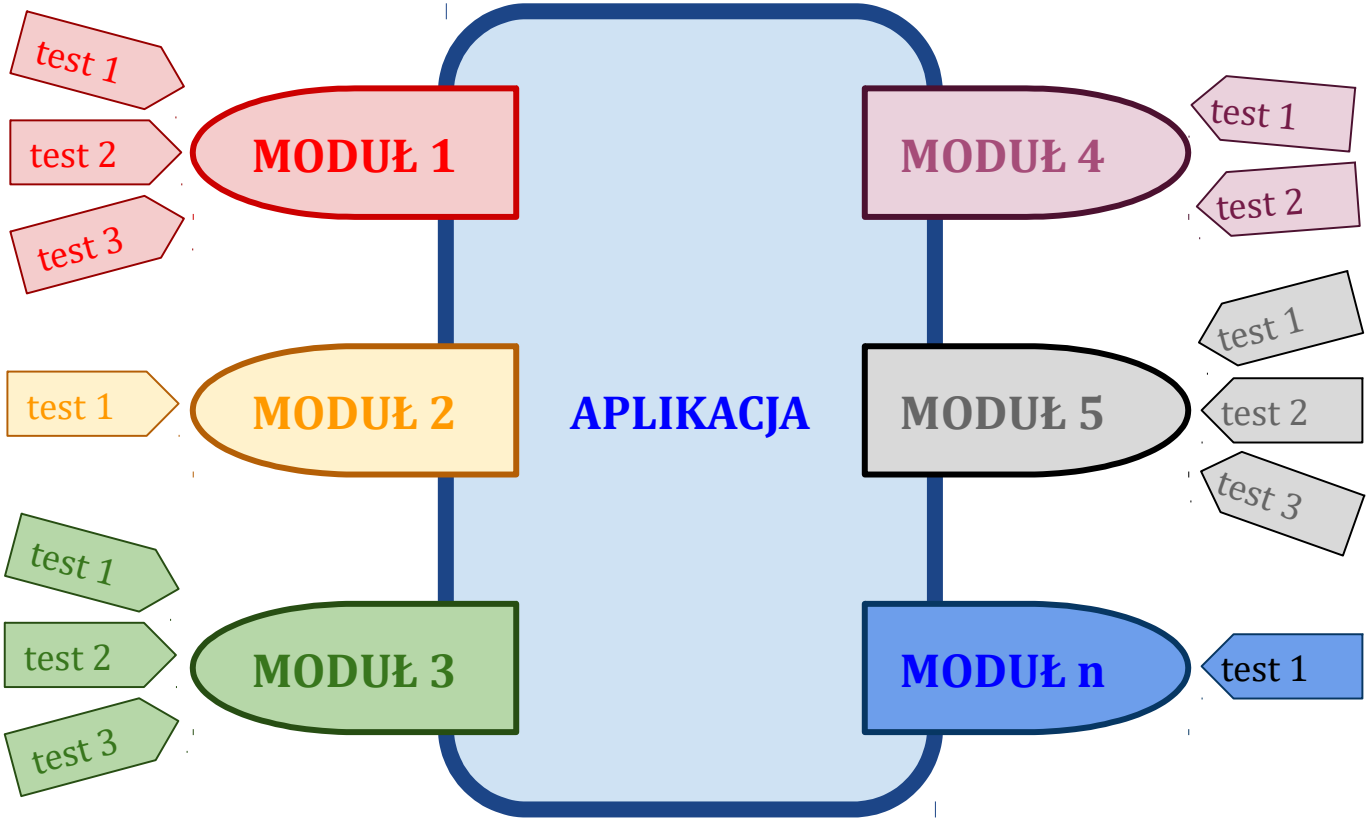
jednostkowe(**unit**)

użytkownika (**end to end**)

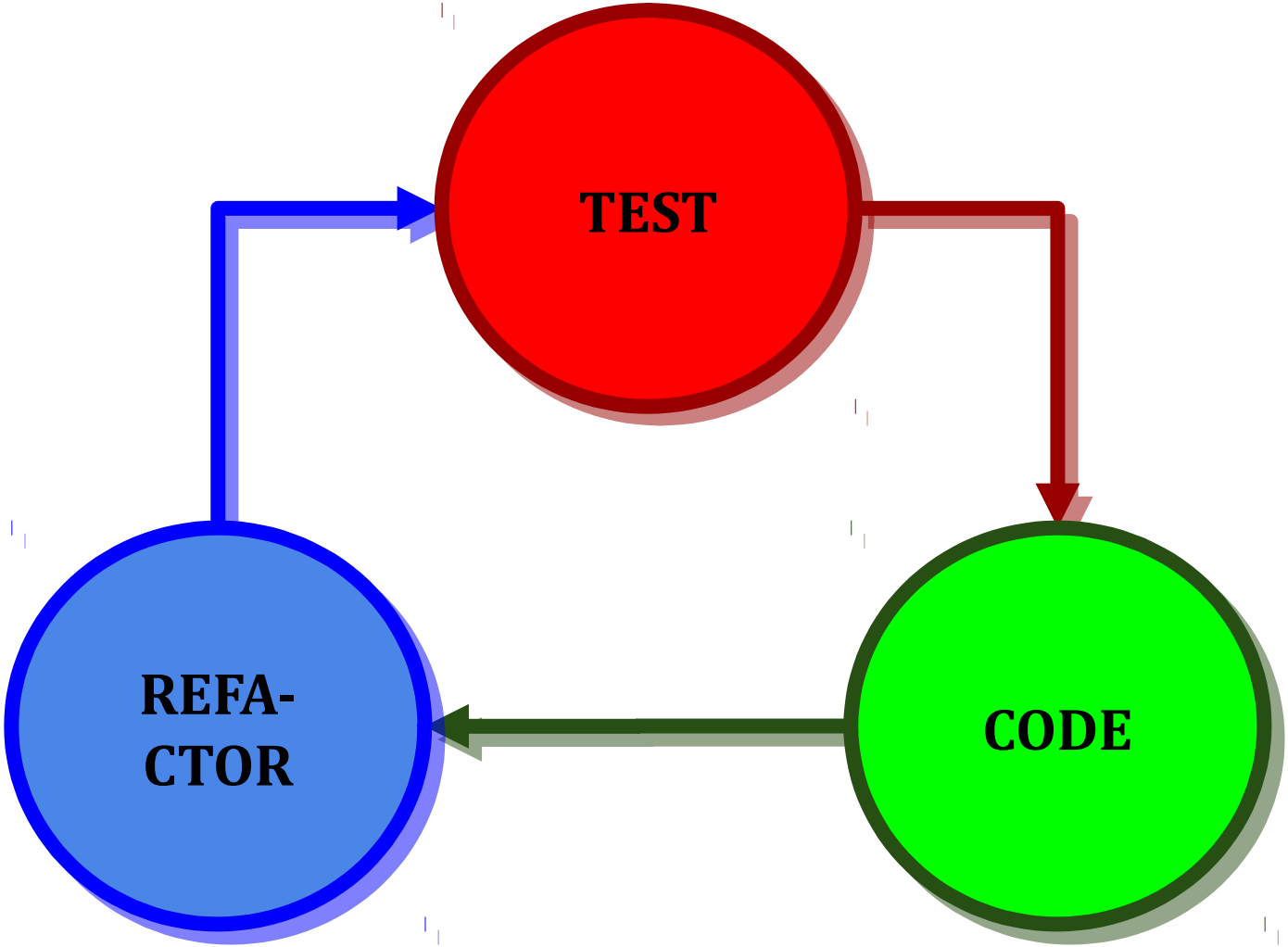
integracyjne (**integration**)




test 1
test 2
test 3
MODUŁ 1
MODUŁ 4
test 1
test 2
test 1
MODUŁ 2
MODUŁ 5
test 1
test 2
test 3
Warstwa n
test 1
test 2
test 3
MODUŁ 3
Warstwa 2
MODUŁ n
test 1
Warstwa 1

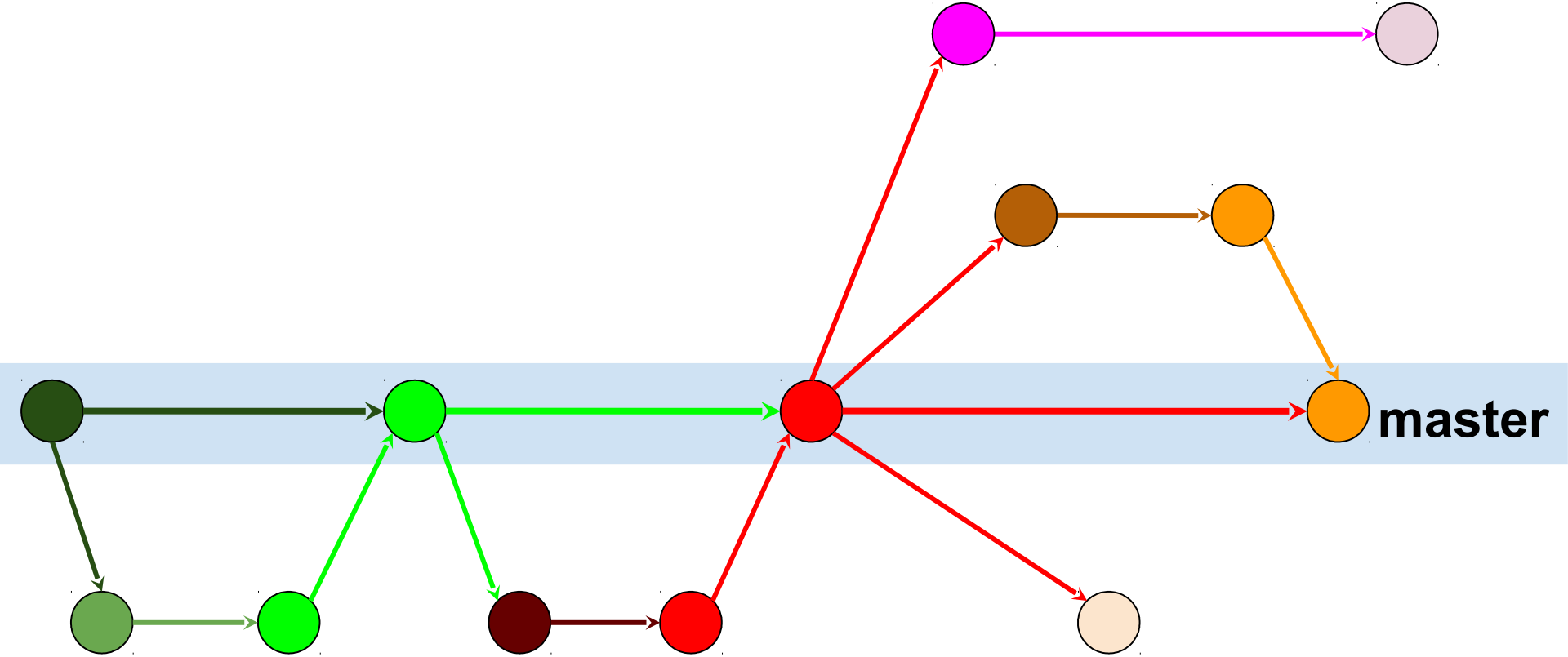

test 1
test 2
test 3
MODUŁ 1
test 1
MODUŁ 2
test 1
test 2
test 3
MODUŁ 3
APLIKACJA
MODUŁ 4
test 1
test 2
MODUŁ 5
test 1
test 2
test 3
MODUŁ n
test 1

# **T**est-**D**riven **D**evelopment

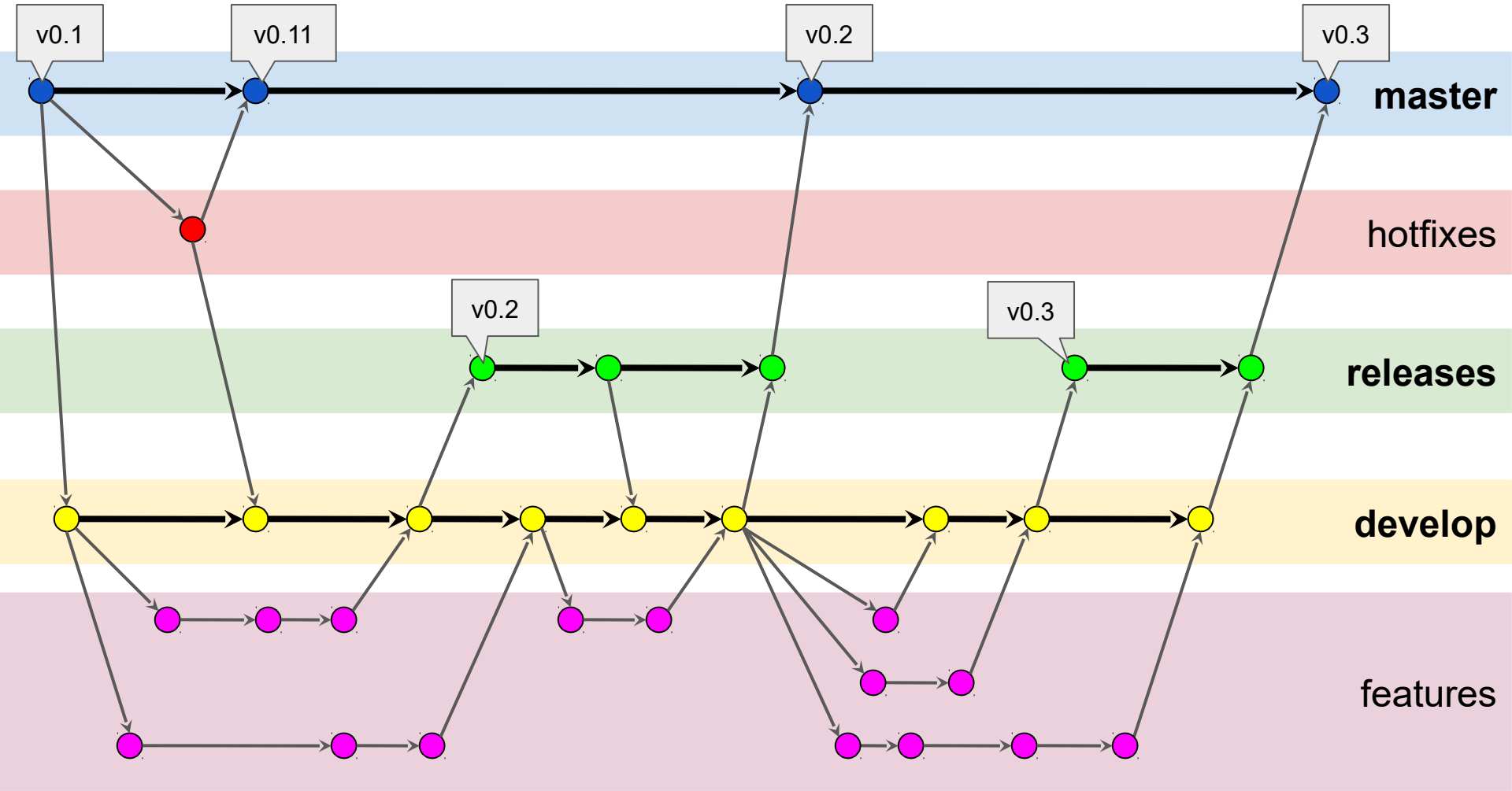

TEST
CODE
REFA-
CTOR

# Git, Git Flow, GitHub

# git

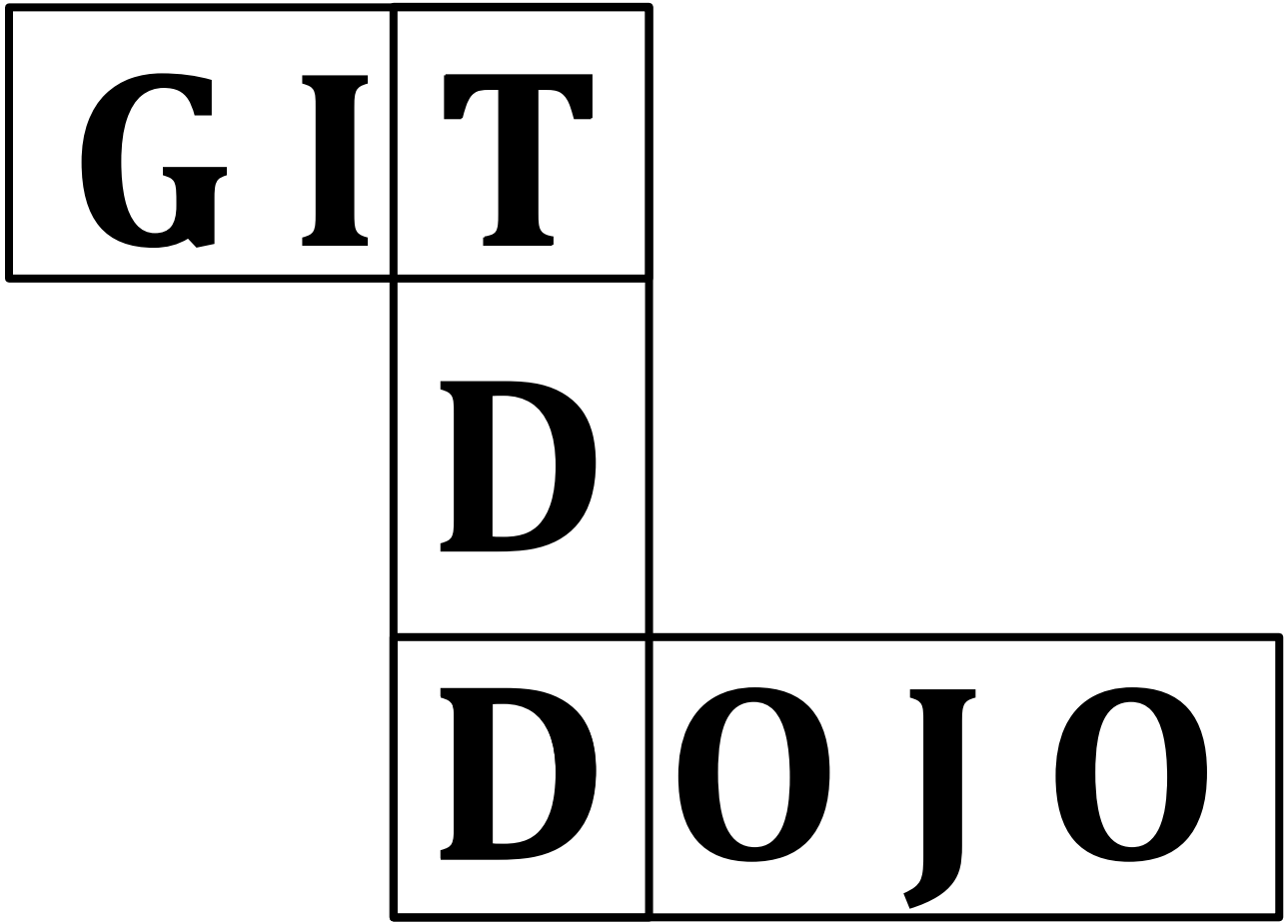

master

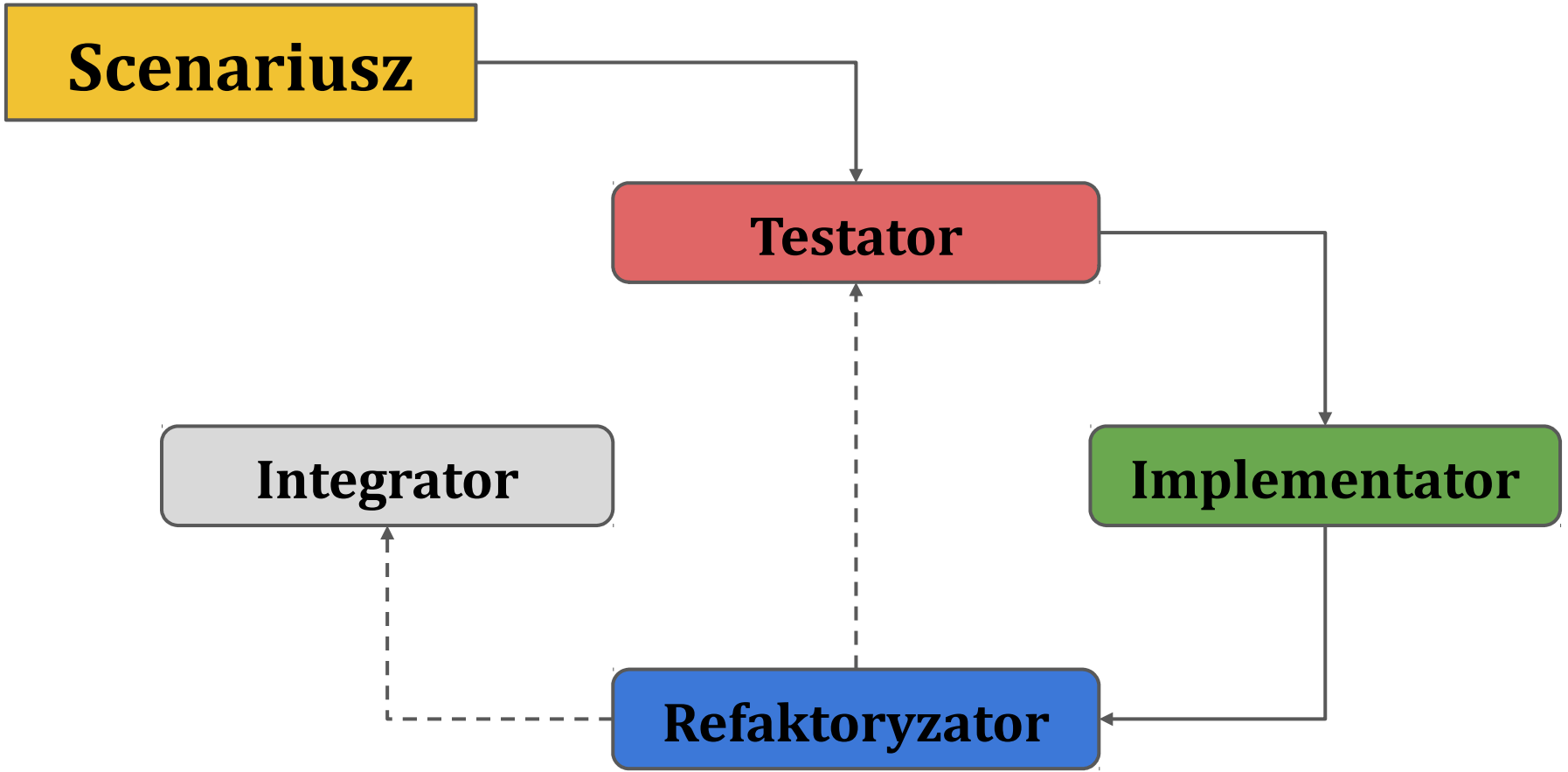
# git flow


v0.1
v0.11
v0.2
v0.3
master
hotfixes
v0.2
v0.3
releases
develop
features

# GitHub

# Warsztaty


GIT
D
DOJO

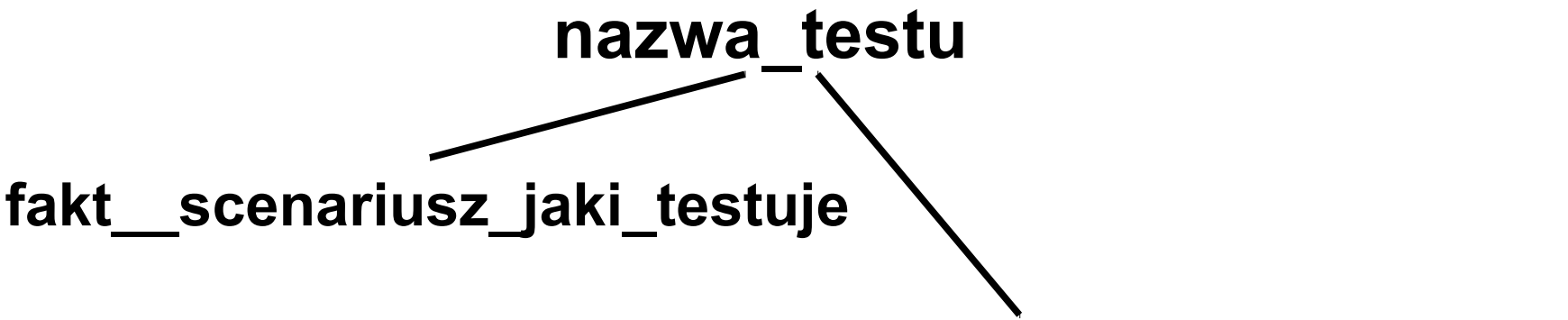

Scenariusz
Testator
Integrator
Implementator
Refaktoryzator

```
public class OperationTest{
        [Fact] lub [Theory][InlineData(var1,var2)]
        public void nazwa_testu()        {
                //arrange
                _fixture.arrange_operation();

                act();

                //assert
                _fixture.assert();
        }

        private void act()   {
                _fixture.act();
        }

        public OperationText(){
                _fixture = OperationFixture.Create();
        }

        private readonly OperationFixture _fixture;
}
```

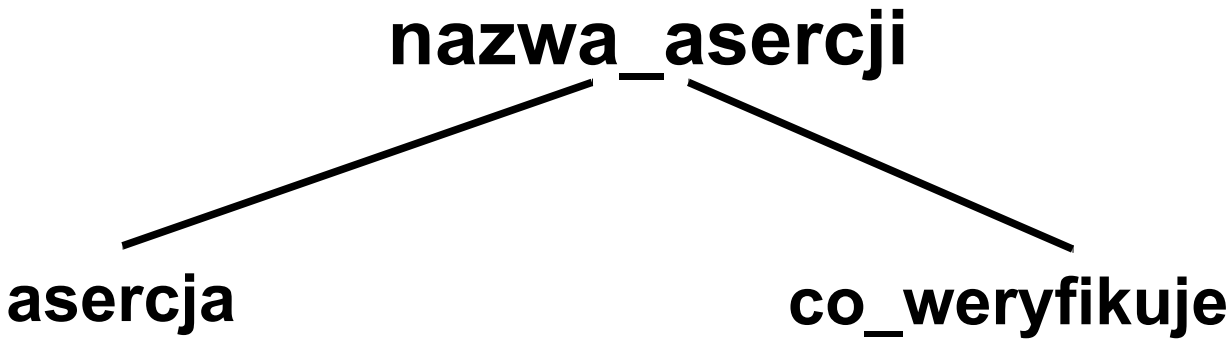
```
public class OperationFixture{
        private Operation _operation;

        public OperationFixture(){
                _operation = new Operation();
        }

        public void arrange_operation()       {
                _operation.Set(0);
        }

        private void act()   {
                _operation.Run();
        }

        public void assert()        {
                _operation.Result
                    .ShouldBe(0);
        }
}
```

```
public class OperationTest{
    [Fact] lub [Theory][InlineData(var1,var2)]
    public void nazwa_testu()       {
        //arrange
        _fixture.arrange_operation();

        act();

        //assert
        _fixture.assert_throw_exception();
    }

    private void act()  {
        _fixture.act();
    }

    public OperationText(){
        _fixture = OperationFixture.Create();
    }

    private readonly OperationFixture _fixture;
}
```

```
public class OperationFixture{
    private Operation _operation;
    private Action _act;

    public OperationFixture(){
        _operation = new Operation();
    }

    public void arrange_operation()      {
        _operation.Set(0);
    }

    private void act()  {
        _act = () => operation.Run();
    }

    public void assert_throw_exception()      {
        _act
            .ShouldThrow<Exception>()
            .WithMessage(”xyz”);
    }
}
```




poprawny_wynik_dodawania__gdy_liczba_a_i_liczba_b_są_całkowite




asercja__poprawny_wynik_dodawania

# Do kodu!