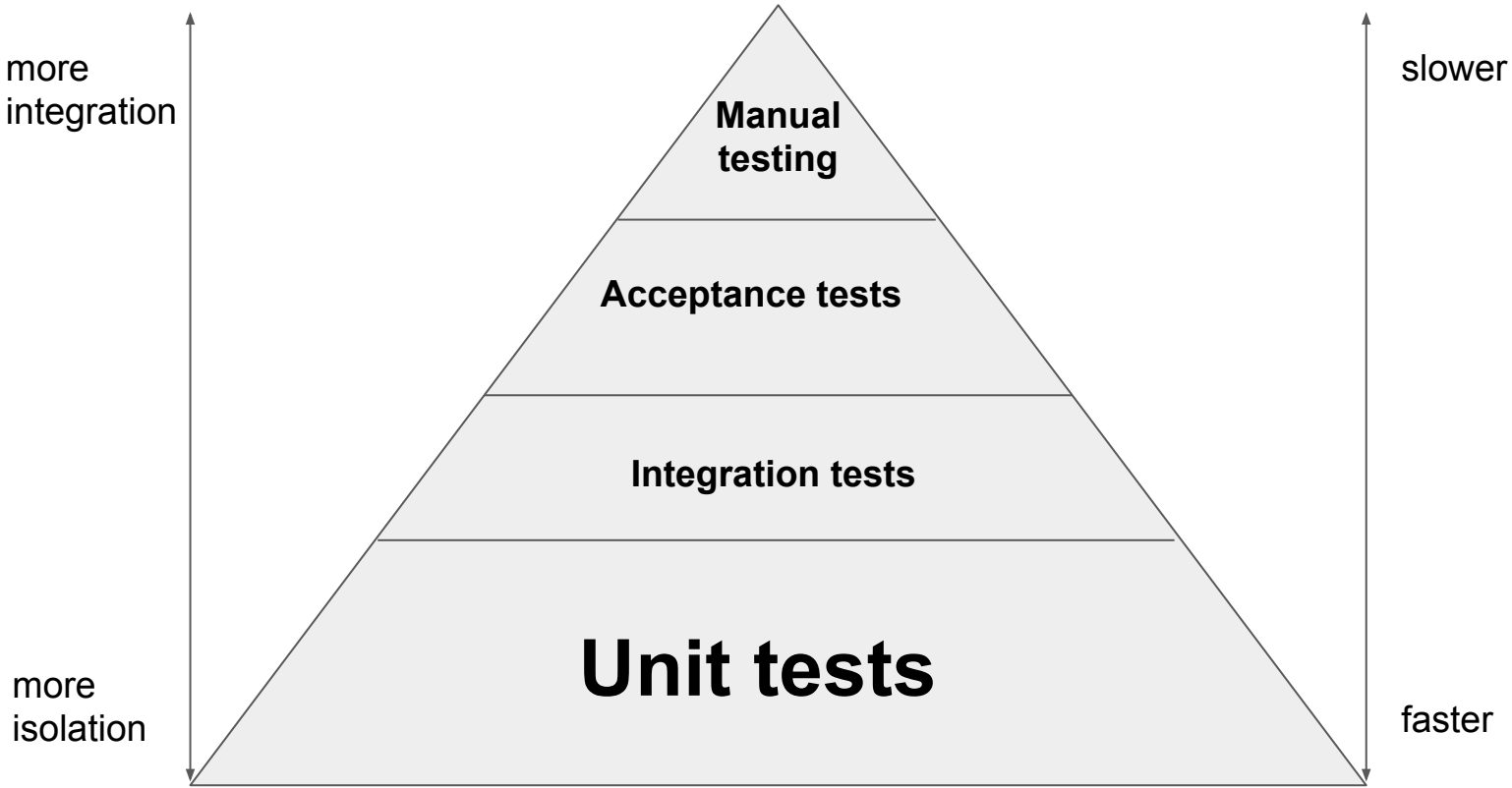
Kodo testavimas (*Unit Testing*) ir jo įtaka projektavimui

2020-10-05

VISMA

Testų tipai



VISMA

Kodo testavimas (Unit testing)

VISMA

Kas yra kodo testavimas?

- A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work. (http://artofunittesting.com)

DEMO (area calculator)

Kam to reikia?

- Galima labai greitai atlikti regresinį testavimą, kai kode daromi nedideli pakeitimai arba refaktorinamas kodas
- Priverčia projektuoti taip, kad klasės būtų kuo mažiau priklausomos nuo kitų (loosely coupled)

Regresija (Regression)

- Funkcionalumas, kuris veikė prieš tai - nebeveikia
  - Kai kodas keičiasi ir auga
  - Naujo funkcionalumo sukūrimas
  - Klaidos ištaisymas

Regresinis automatinis testavimas

Testai yra paleidžiami po kiekvieno pakeitimo

- Automatiškai (skriptų pagalba, IDE pagalba ir kt.)
- Užtikriname, kad nesugriovėme prieš tai veikiančio funkcionalumo.
- Užtikrinama, jog senos ištaisytos klaidos vis dar yra ištaisytos.
- Užtikrinamas minimalus lygis veikiančio funkcionalumo.

VISMA

DEMO (smart calculator)

VISMA

Unit testai turi būti

- Greiti.
  - Kiek užtrunka praleisti visus testus?
- Patikimi.
  - Veikia tik atmintyje.
  - Neturi prieigos prie tinklo, failų sistemos, duomenų bazių ir kt. šaltinių, kurie sulėtintų veikimą ar kitaip darytų įtaką testo veikimui
  - Nepriklauso nuo datos / laiko
  - Nepriklauso nuo leidimo eilės
- Tikslūs.
  - Testuoja mažus funkcionalumo gabaliukus.
  - Unit vs. integraciniai testai (kaip gabaliukai integruojasi tarpusavyje)
- Lengvai rašomi ir skaitomi

VISMA

Priklausomybė nuo laiko

```php
class DeliveryEstimator
{
    public function estimateDays(): int
    {
        $now = new DateTimeImmutable();

        $tuesday = 2;
        $saturday = 6;
        $dayOfWeek = (int)$now->format('N'); // 1 - Monday, 2 - Tuesday, ...

        if ($dayOfWeek == $saturday) {
            $days = 4;
        } elseif ($dayOfWeek > $tuesday) {
            $days = 5;
        } else {
            $days = 3;
        }
        return $days;
    }
}
```

```php
class DeliveryEstimatorWithParam
{
    public function estimateDays(DateTimeInterface $now): int
    {
        $tuesday = 2;
        $saturday = 6;
        $dayOfWeek = (int)$now->format('N'); // 1 - Monday, 2 - Tuesday, ...

        if ($dayOfWeek == $saturday) {
            $days = 4;
        } elseif ($dayOfWeek > $tuesday) {
            $days = 5;
        } else {
            $days = 3;
        }

        return $days;
    }
}
```

VISMA

Patarimai rašant testus

- Neįmanoma ištestuoti visų įmanomų įvedamų parametrų kombinacijų.
- Reiktų pagalvoti apie ribinius atvejus.
- Reikia pagalvoti apie klaidas ir kai perduodami tušti duomenys.
- Testuoti vieną dalyką viename testo metode.
- Kiekvienas testo metodas turi turėti kaip įmanoma mažiau *assert* veiksmų
- Vengti biznio logikos.

Rekomenduojama struktūra

- Arrange - paruošimas
- Act - vykdymas
- Assert - tikrinimas

Arba

- Given
- When
- Then

```php
class MallardDuckTest extends TestCase
{

    public function testDisplay()
    {
        //arrange
        $duck = new MallardDuck();

        //act
        $result = $duck->display();

        //assert
        $this->assertEquals( expected: 'I am Mallard Duck', $result);

    }
}
```

VISMA

Dažnai pasitaikančios ydos

- Testai rašomi vėliau negu kodas.
- Testus rašo ne kodo autorius.
- Testų rašymas perleidžiamas testuotojams (“nes jie gi testuotojai”).
- Testų kodui skiriamas nepakankamas dėmesys.
- Testai priklauso vieni nuo kitų
- Viename teste tikrinami keli skirtingi atvejai

Projektavimo įtaka testuojamumui
arba
Testavimo įtaka projektavimui

VISMA

Problemos

- Esminė problema - blogai suprojektuotos klasės apsunkina testavimą.

- Testavimą apsunkina
  - Testuojamos klasės sukūrimas
  - Priklausomybių izoliacija
  - Patikrinimas ar sąveikos su priklausomybėmis veikia korektiškai

Patarimai, kaip rašyti testuojamą kodą

- Naudoti priklausomybių injekcijas (Dependency injection).
  - Tai lengvai leidžia naudoti imitacijas vietoj tikrų objektų testuojant.
- Dirbti su interfeisais.
  - Projektavimo principas **Favor composition over inheritance**
  - Paveldėti metodai gali apsunkinti testavimą
- Vengti sudėtingų statinių metodų.
  - Statinių metodų naudojimas sukuria stiprias priklausomybes. Statinių metodų nepavyksta perrašyti ar pakeisti, todėl sunku juos imituoti testuojant.
  - Math::pow() - gerai
  - DatabaseWorker::selectData() - blogai

VISMA

Priklausomybių injekcijos (Dependency injection)

- Dependency injection - projektavimo šablonas, kuris įgyvendina “dependency inversion” principą (SOLI**D**).
- Objektas gauna visus reikalingus objektus (priklausomus objektus, “priklausomybes”) per konfigūraciją
  - vietoje to kad ieškotų ar susikurtų reikalingus objektus pats.

```
class MapLoader{
  Database db = new Database();

  Map load(){
    return db.select("...");
  }
}
```

```
class MapLoader{
  MapLoader(Database db){
    this.db = db;
  }

  Map load(){
    return db.select("...");
  }
}
```

VISMA

Priklausomybių injekcijų privalumai

- Įgalina rašyti testuojamą kodą
- Geresnis kodo perpanaudojamumas
- Lengvesnė konfigūracija
- Implementacijų pakeitimas
- Laikomasi “Single responsibility” (**S**OLID) ir “Dependency Inversion” (SOLI**D**) principų.

VISMA

Imitacija (Mocking)

VISMA

Kas yra Mocking?

Šios temos kontekste:

- Testuojami objektai dažnai turi priklausomybių nuo kitų sudėtingų objektų.
- Norint ištestuoti funkcionalumą izoliuotai - reikia “tikras” priklausomybes pakeisti imitacijomis.

- Objektai, simuliuojantys “tikrų objektų” elgesį.

Kam reikalingi netikri objektai (Mock objects)

Priklausomybių eliminavimas/imitavimas:

```
public void testOrderLookup() {
        Database db= new MockDatabase();
        db.expectQuery("select order_no from Order where cust_no is 123");
        db.returnResult(new String[] {"Order 2" ,"Order 3"});
        …
}
```

VISMA

DEMO (mock)

Imitavimo privalumai

- Sparta
- Patikimumas
- Skaitomumas

Testavimo įtaka projektavimui

“Unit testing forces you to actually use the class you are creating and punishes you if the class is too big and contains more than one responsibility.”

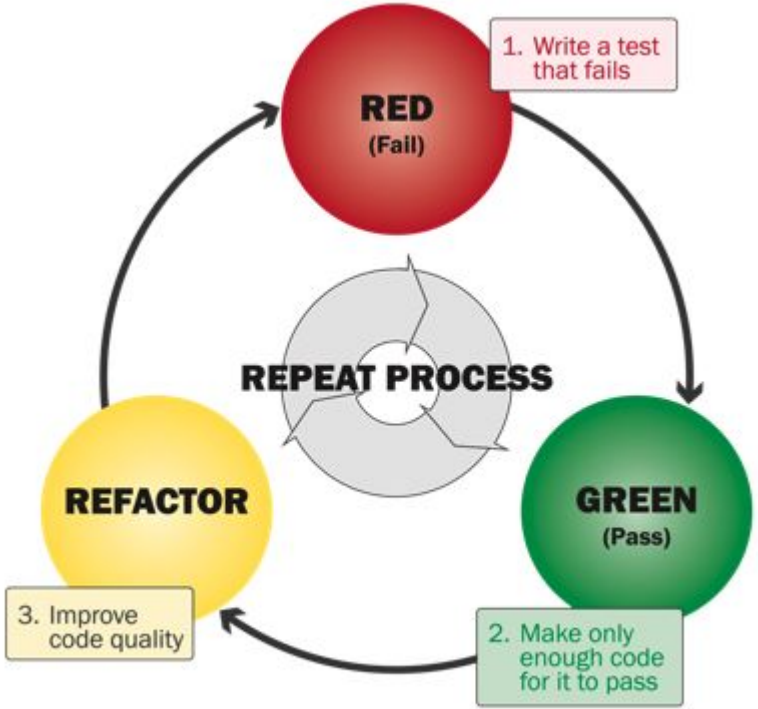
“By that pain, you change your design to be more cohesive and loosely coupled.”

VISMA

Test Driven Development

TDD eiga

- Parašyti ir paleisti neveikiantį test'ą
- Implementuoti tiek, kad testai veiktų
- Gerinti kodo kokybę (refactor)



VISMA

Testas kaip specifikacija

VISMA

Įvykdomos specifikacijos

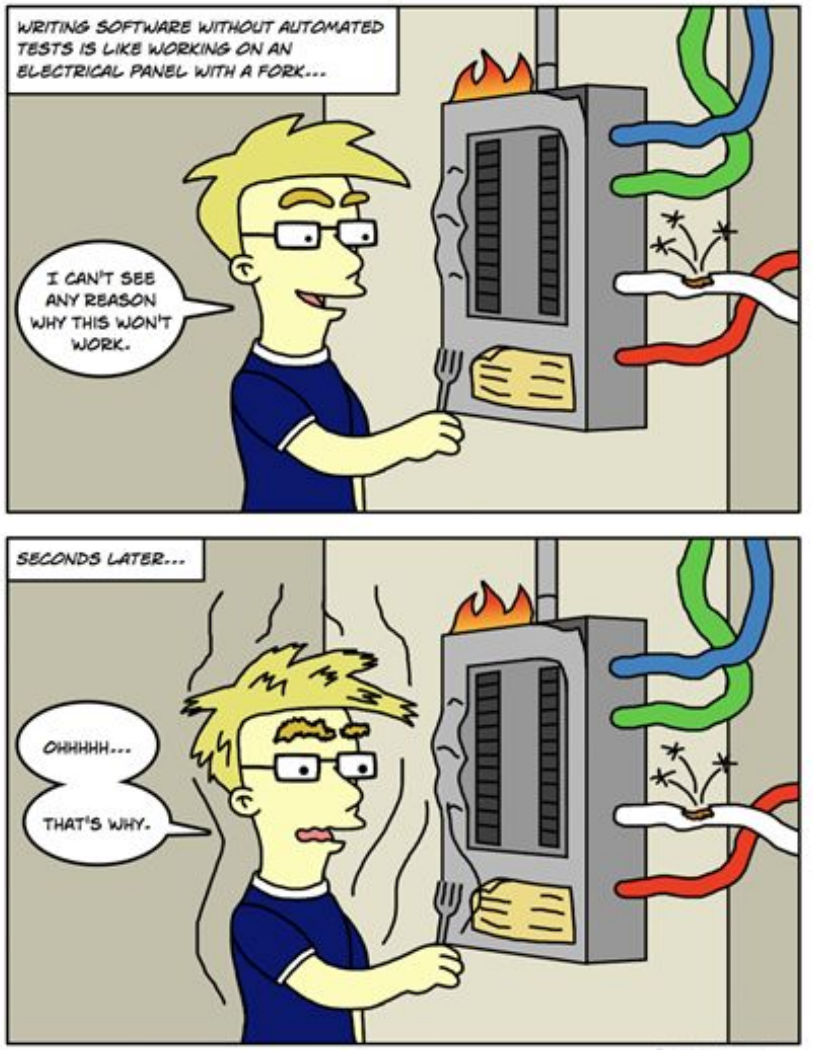
- Teisingai taikant TDD praktikas unit testai tampa detaliomis specifikacijomis, kurios sukurtos kaip tik laiku.
- Programuotojai tipiškai yra linkę dirbti su kodu, o ne su dokumentacija.
- Bandant suprasti klasę ar metodą pirmiausiai programuotojai žiūrės į kodo pavyzdžius, kurie kviečia metodus.
- Testai sukurti kaip specifikacijos daro būtent tai!

DEMO (TDD)

VISMA

Kodo testavimo svarba

- Problemos randamos anksti
- Įgalinami kodo pakeitimai
- Paprastesnė integracija
- Dokumentacija
- Įtakoja QA sėkmę
- Programinės įrangos patikimumas
- Lengviau ir pigiau aptikti klaidas



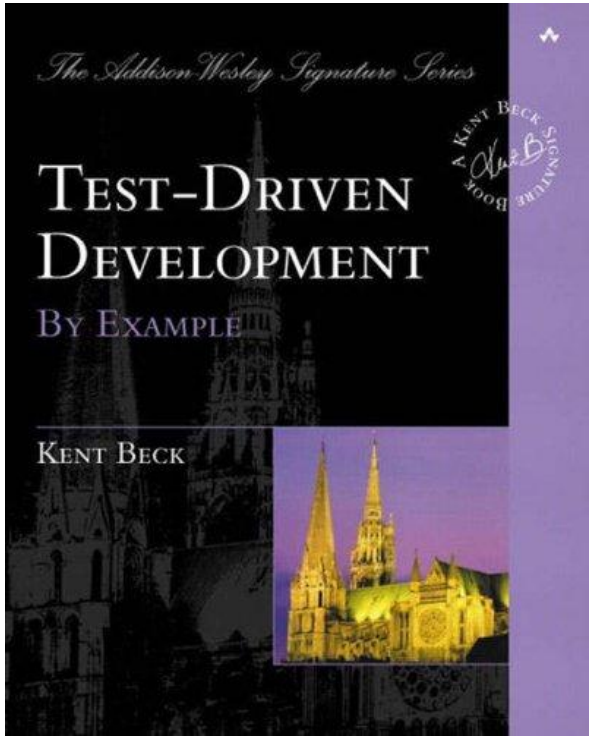
Naudoti kodo pavyzdžiai

https://github.com/chris1312/unittesting

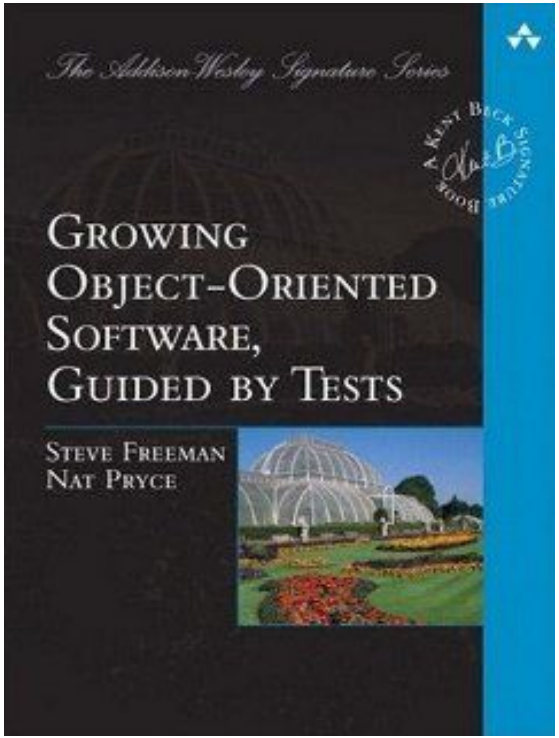
VISMA

Literatūra

Kent Beck, 2002



Steve Freeman, 2009



VISMA

Paskutinė skaidrė

Yra greičiau parašyti kodą su testais negu be jų

..arba kitais žodžiais tariant..

Yra greičiau parašyti kodą be testų,
nebent tikrai reikia, kad kodas veiktų korektiškai.

VISMA