Clean Code*

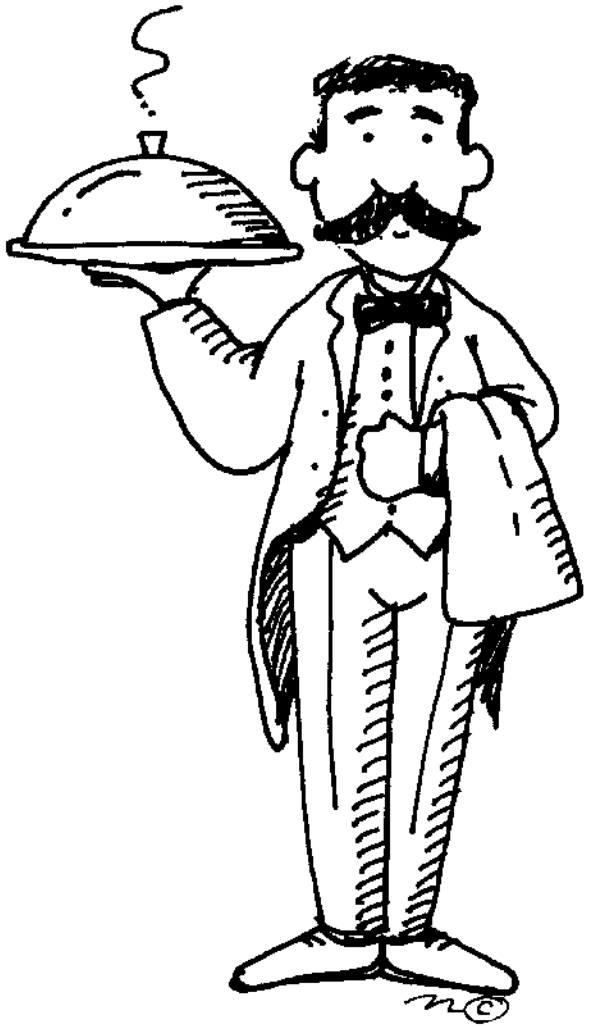
* sau de ce e mai important felul în care scriem cod decât ceea ce scriem

Asist. dr. Bogdan Iancu

Asist. dr. Alin Zamfiroiu

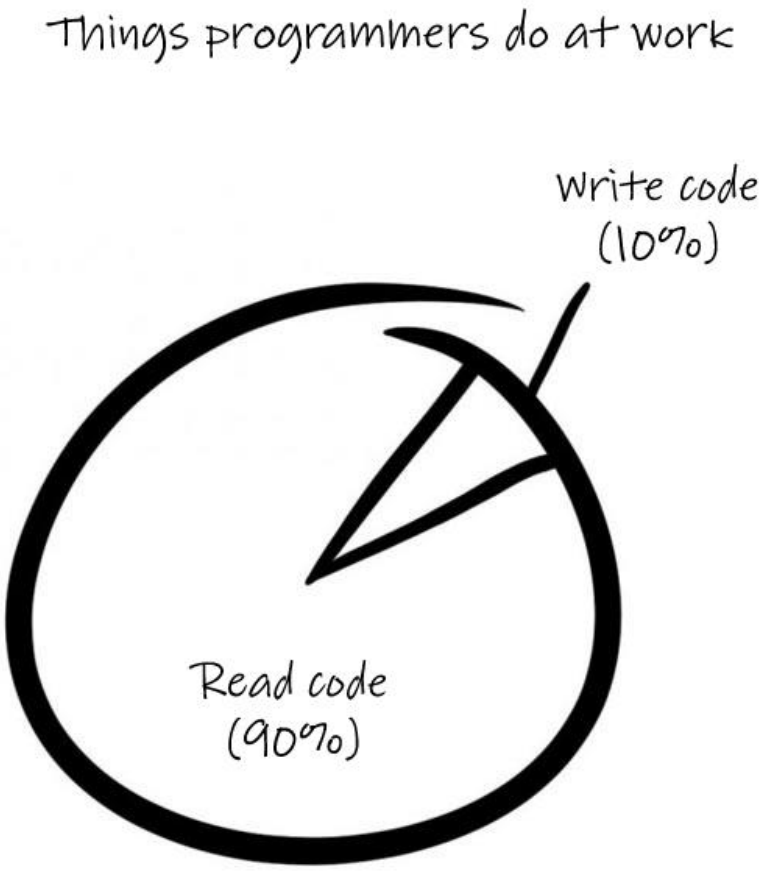
Despre ce vom discuta

- De ce clean code?
- Principii
- Convenții de nume
- Clean Code în practică
- Scurt dicționar
- Instrumente
- Bonus

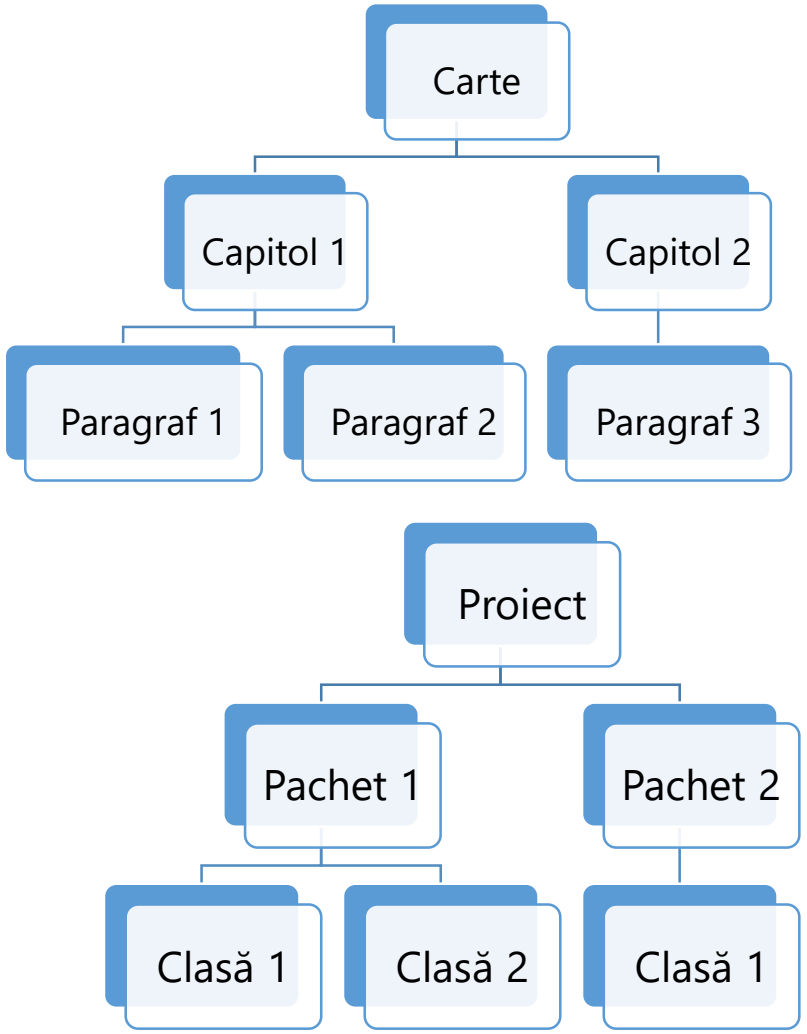


De ce Clean Code?

- Programarea nu constă în a spune computerului ce să facă
- Programarea constă în a spune altui om ce vrem să facă un computer
- Din păcate câteodată acel „alt om” suntem chiar noi
- Până la urmă suntem autori
- Nu avem timp să fim leneși
- Altfel putem ajunge un substantiv

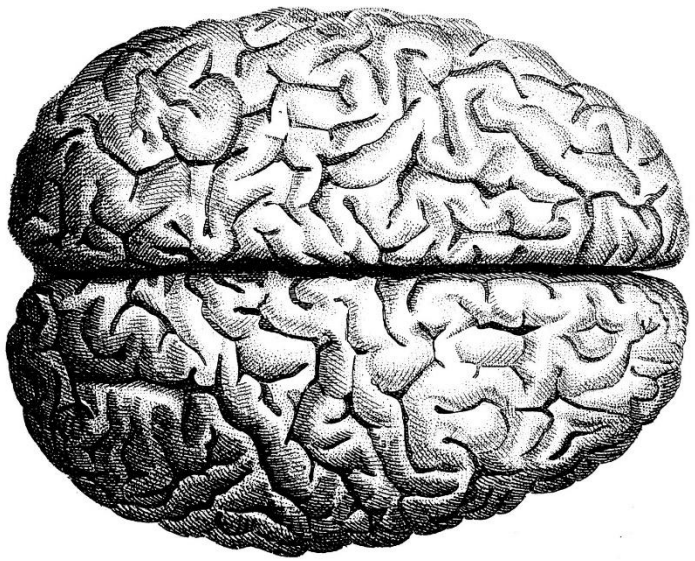


Programator = Scriitor



Încă ceva

- Când citim cod creierul nostru joacă rol de compilator
- Conform studiilor oamenii pot reține simultan doar 7 elemente (±2) în memorie
- Rubber Duck Programming (Debugging)



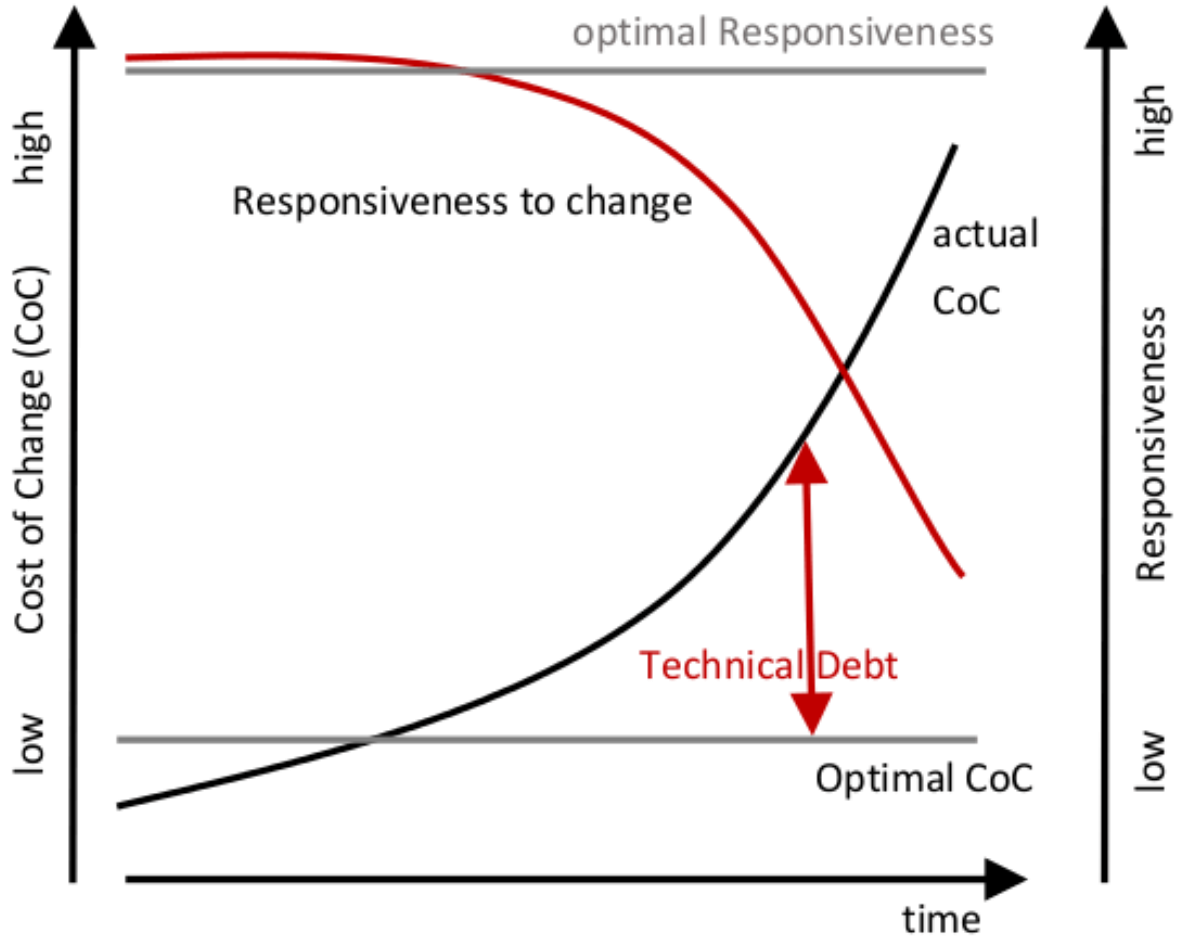
Ce înseamnă Clean Code

- Codul trebuie sa fie ușor de citit
- Codul trebuie sa fie ușor de înțeles
- Codul trebuie să fie ușor de modificat

... de către oricine

http://www.codeoclock.com/2015/03/03/clean-up/

Avantaje



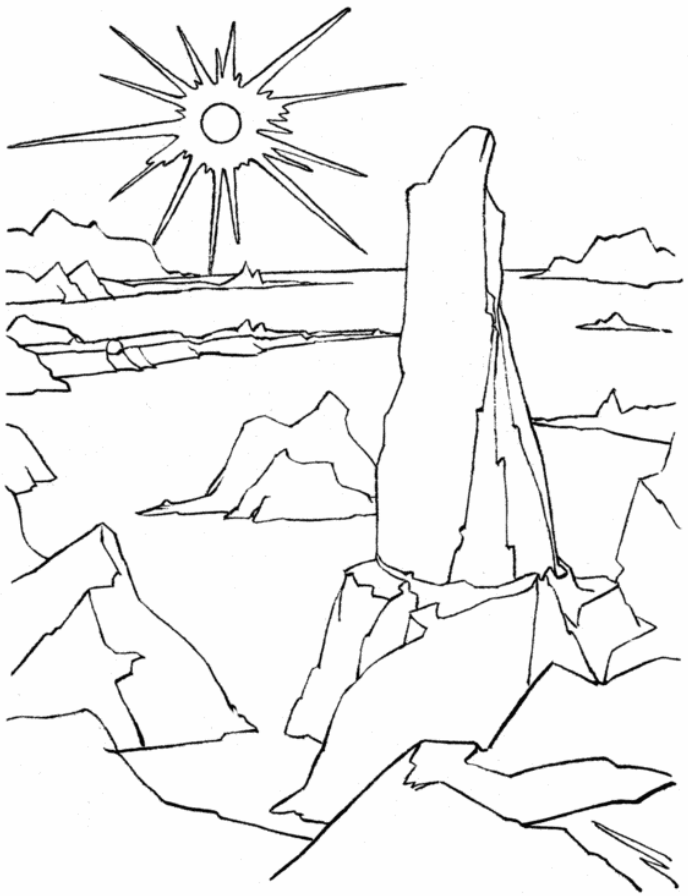
http://www.chudovo.com/Blog/CleanCode_General_Principles

Principii

- DRY
- KISS
- YAGNI
- SOLID

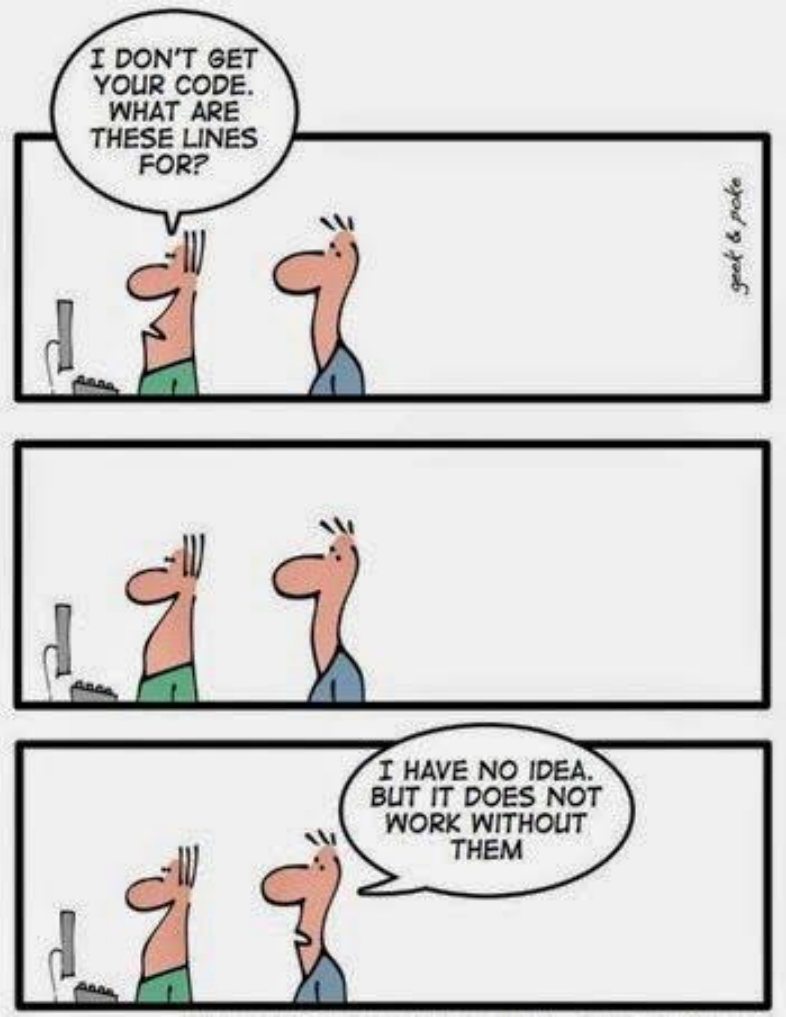
D.R.Y.

- **D**on’t **R**epeat **Y**ourself
- Aplicabil ori de câte ori dăm Copy/Paste unei bucăți de cod



K.I.S.S.

- **K**eep **I**t **S**imple and **S**tupid
- Ori de câte ori vrem ca o metodă să facă de toate

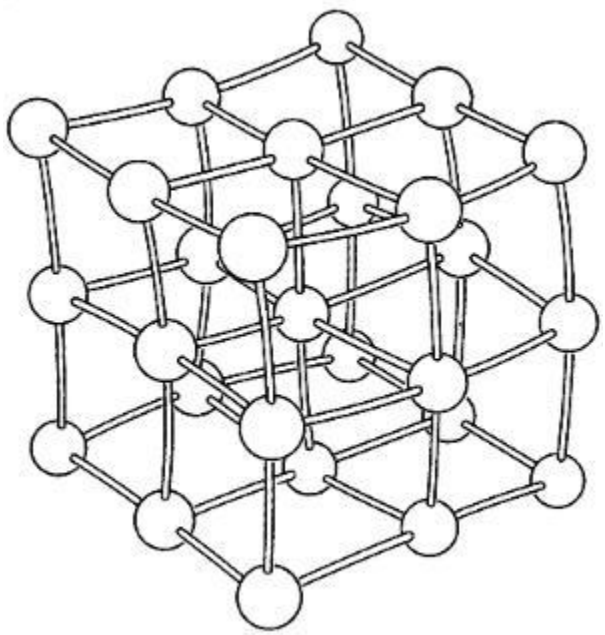


Y.A.G.N.I.

- **Y**ou **A**in’t **G**onna **N**eed **I**t
- Nu scriem metode ce nu sunt necesare încă (poate nu vor fi necesare niciodată)
- Oarecum derivat din KISS

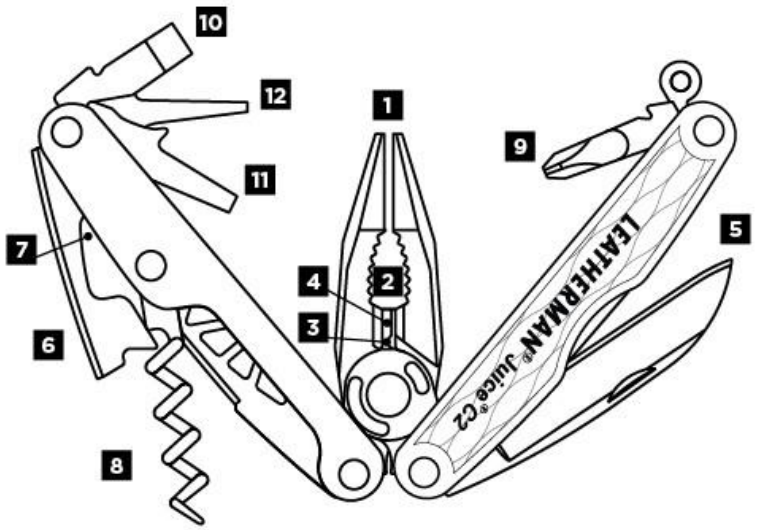
S.O.L.I.D.

- **S**ingle responsibility (SRP)
- **O**pen-closed (OCP)
- **L**iskov substitution (LSP)
- **I**nterface segregation (ISP)
- **D**ependency inversion



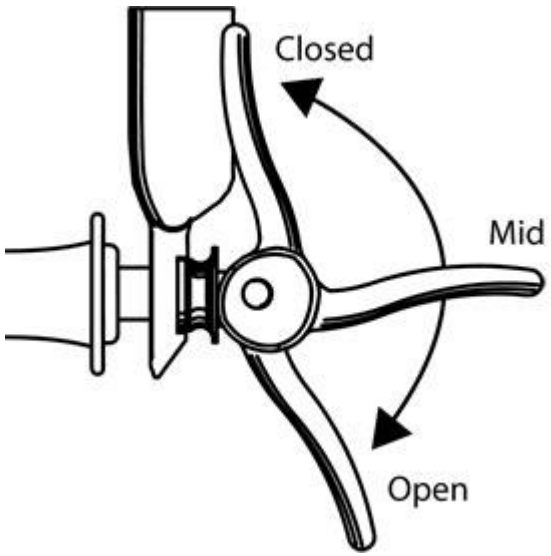
Single Responsability Principle

- O clasă trebuie să aibă întotdeauna o singură responsabilitate și numai una
- În caz contrar orice schimbare de specificații va duce la inutilitatea ei și rescrierea întregului cod



Open-Close Principle

- Clasele trebuie să fie deschise (open) pentru extensii
- Dar totuși închise (closed) pentru modificări

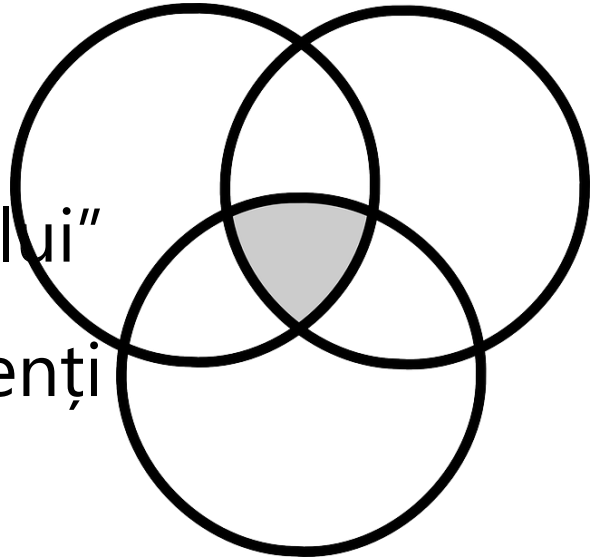


Liskov Substitution Principle

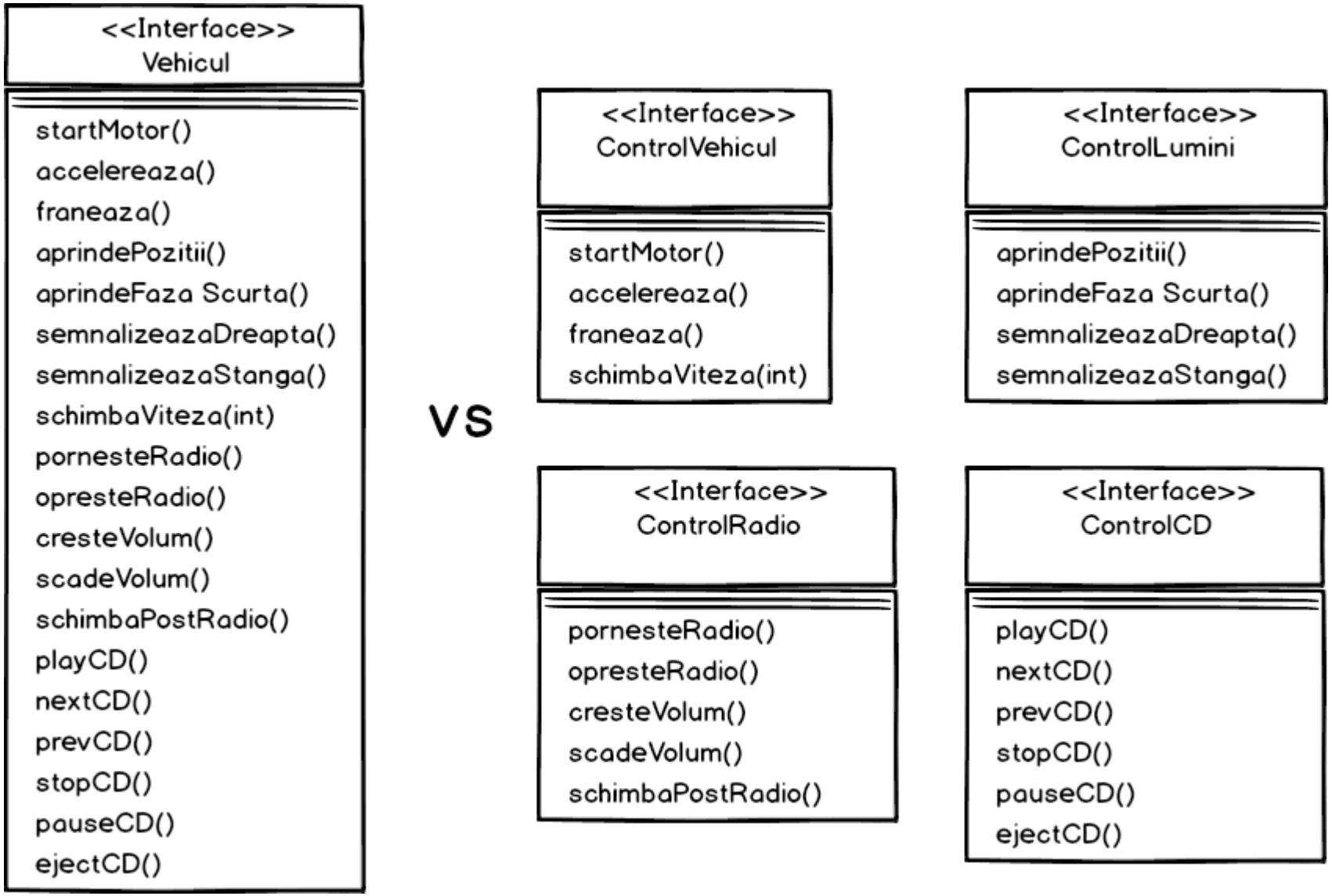
- Obiectele pot fi înlocuite oricând cu instanțe ale claselor derivate fără ca acest lucru să afecteze funcționalitatea
- Întâlnită și sub denumirea de „Design by Contract”

Interface Segregation Principle

- Mai multe interfețe specializabile sunt oricând de preferat unei singure interfețe generale
- Nu riscăm astfel ca prin modificarea „contractului” unui client să modificăm și contractele altor clienți
- Obiectele nu trebuie obligate sa implementeze metode care nu sunt utile



Interface Segregation Principle

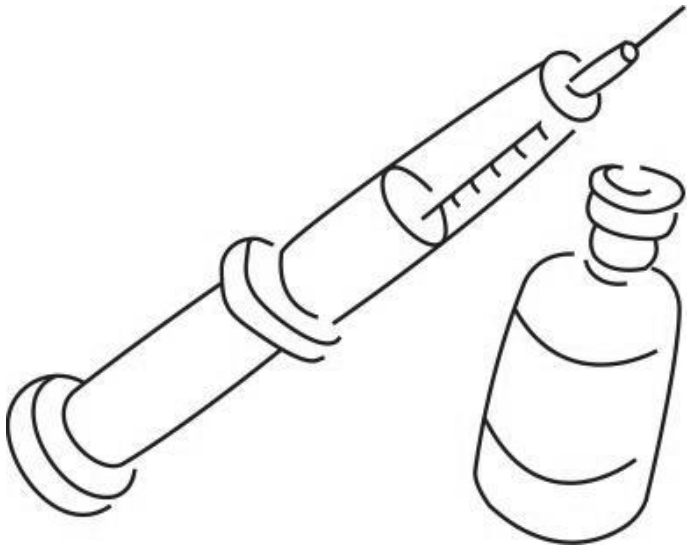


Dependency Inversion Principle

- O clasă trebuie să depindă de abstractizări, niciodată de obiecte concrete
- O aplicabilitate a acestui principiu este ***Dependency Injection*** ce permite cuplarea slabă a serviciilor de clienți

Program to interfaces, not implementations

Depend on abstractions. Do not depend on concrete classes



Ce părere aveți despre codul de mai jos?

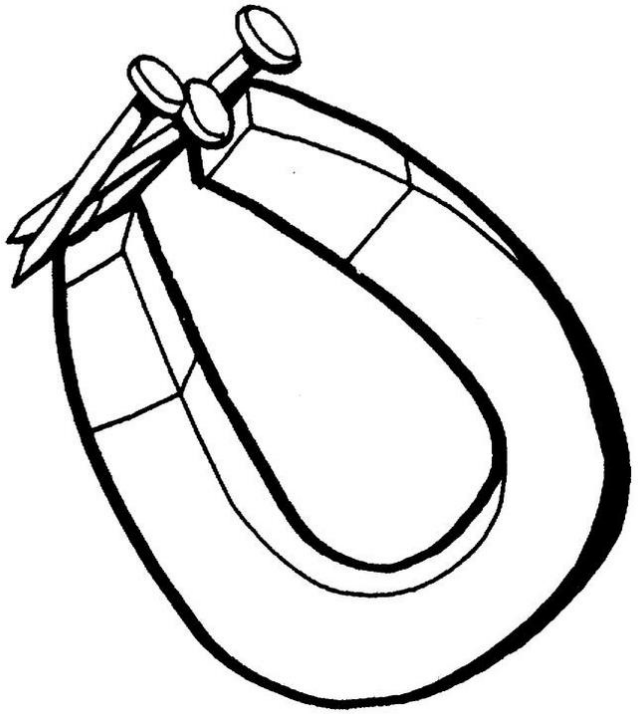
```
public static int Calculeaza() {
    int x = 5;
    ArrayList l = new ArrayList();
    l.add(x);
    int y = 10;
    l.add(y);
    l.add(15);
    int m = 0;
    for(Object k : l) {
        m+= (int)k;
    }
    return m;
}
```

Convenții de nume

- UpperCamelCase
- lowerCamelCase
- System Hungarian Notation
- Apps Hungarian Notation

Convenții de nume - Clase

- Atenție la denumirile alese
- Numele prost alese sunt un magnet pentru programatorii leneși
- Compuse dintr-un substantiv specific, fără prefixe și sufixe inutile
- Nu trebuie să uităm de Single Responsability Principle



Convenții de nume - Metode

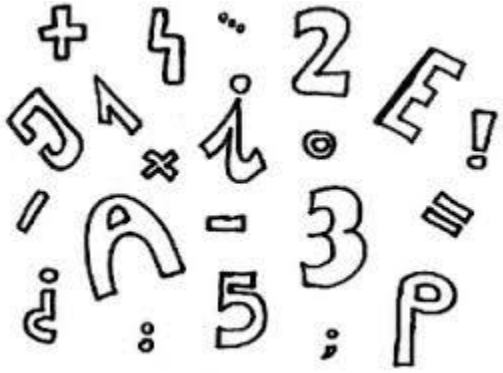
- Trebuie să descrie clar ce fac
- Unde denumirile prost alese nu pot fi evitate (sunt generate automat de mediu) e indicat ca în interiorul lor să fie doar apeluri de alte metode
- Dacă denumirea unei metode conține o conjuncție („și”, „sau”, „ori”) cel mai probabil vorbim de două metode
- Nu abr den met

Convenții de nume - Variabile

- Nu e indicat ca variabilele de tip șiruri de caractere să conțină cod din alte limbaje (SQL, CSS)

- Variabilele booleane trebuie să sune ca o întrebare la care se poate răspunde cu adevărat/fals

  boolean isTerminated = false;



- Când există variabile complementare, numele trebuie să fie simetrice

Reguli de scriere a codului sursă

```
protected String nume; protected String prenume;protected int varsta;
```

Declararea unei variabile

pe fiecare linie.

```
public class Persoana {
    protected String nume;
    protected String prenume;
    protected int varsta;
```

Reguli de scriere a codului sursă

Blocurile de cod încep cu

{ și se termină cu }.

```
public double getMedie() {
    return medie;
}
```

Chiar dacă avem o

singură instrucțiune.

Reguli de scriere a codului sursă

Blocurile cu instrucțiuni sunt marcate și prin identare.

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

Reguli de scriere a codului sursă

Între headerul funcției și acolada de deschidere a blocului funcției se pune un spațiu

```
public void setMedie(double medie) {
    this.medie = medie;
}
```

Reguli de scriere a codului sursă

Acolada de închidere a unui corp de instrucțiuni este singură pe linie.

```
public void setVarsta(int varsta) {
    if (varsta > 0) {
        this.varsta = varsta;
    } else {
        this.varsta = 1;
    }
}
```

Excepție fac situațiile când avem *if-else* sau *try-catch*.

Reguli de scriere a codului sursă

Metodele sunt separate printr-o singură linie goală.

```
public double getMedie() {
    return medie;
}

public void setMedie(double medie) {
    this.medie = medie;
}
```

Reguli de scriere a codului sursă

Parametrii sunt separați prin virgulă și spațiu.

```
public Persoana(String nume, String prenume, int varsta) {
```

Reguli de scriere a codului sursă

Operatorii sunt separați de operanzi printr-un spațiu.

```
return nume + " " + prenume;
```

Excepția de la această regulă fac operatorii unari.

Reguli de Clean Code în structuri condiționale

- Evitați comparațiile cu true și false

```
if(raspunsCorect == true) {
    //...
}
```

- Variabilele booleane pot fi instanțiate direct

```
boolean raspunsCorect;

if(punctaj == 0) {
    raspunsCorect = false;
}
else {
    raspunsCorect  = true;
}
```

- Nu fiți negativiști!

Reguli de Clean Code în structuri condiționale

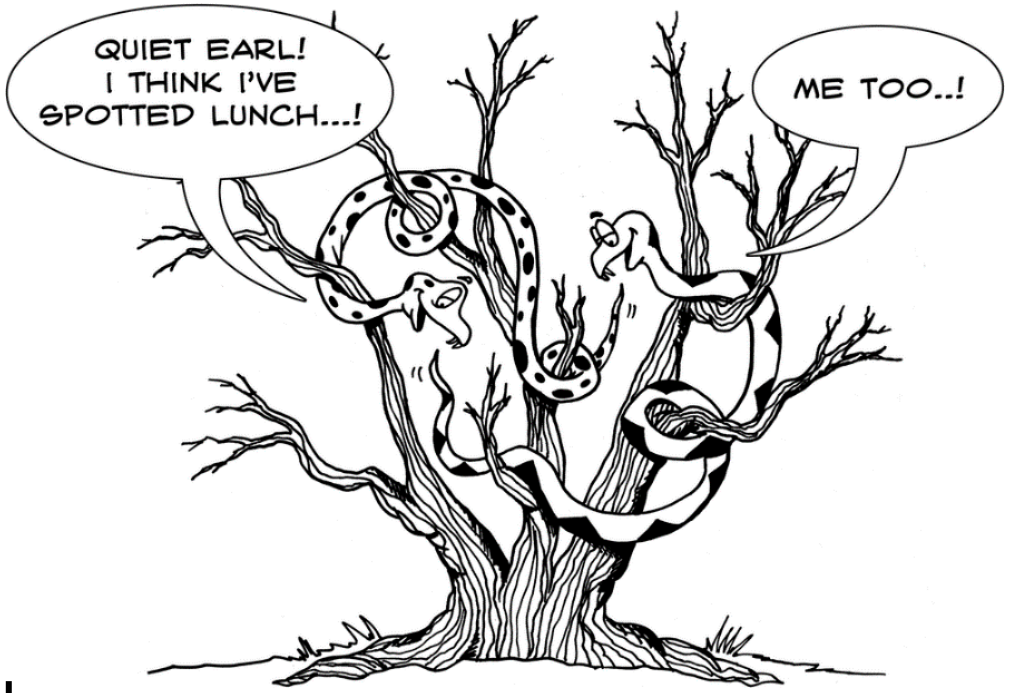
- Folosiți operatorul ternar ori de câte ori este posibil

int max = a > b ? a : b;

- Nu comparați direct cu stringuri, folosiți enum pentru situații de genul
- Constantele trebuie indentificate și denumite (de obicei la începutul claselor)

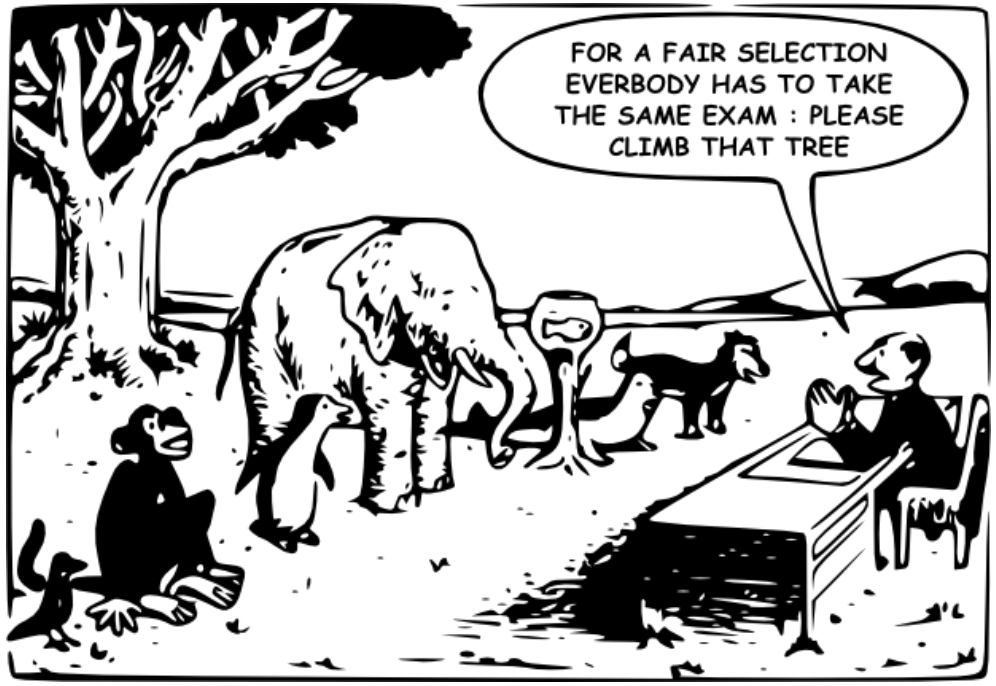
Reguli de Clean Code în structuri condiționale

- Ori de câte ori condițiile devin prea mari sunt indicate variabilele intermediare
- De obicei folosirea *enum* denotă un design greșit al claselor
- Multe constante indică o nevoie de înglobare a lor într-o tabelă din baza de date



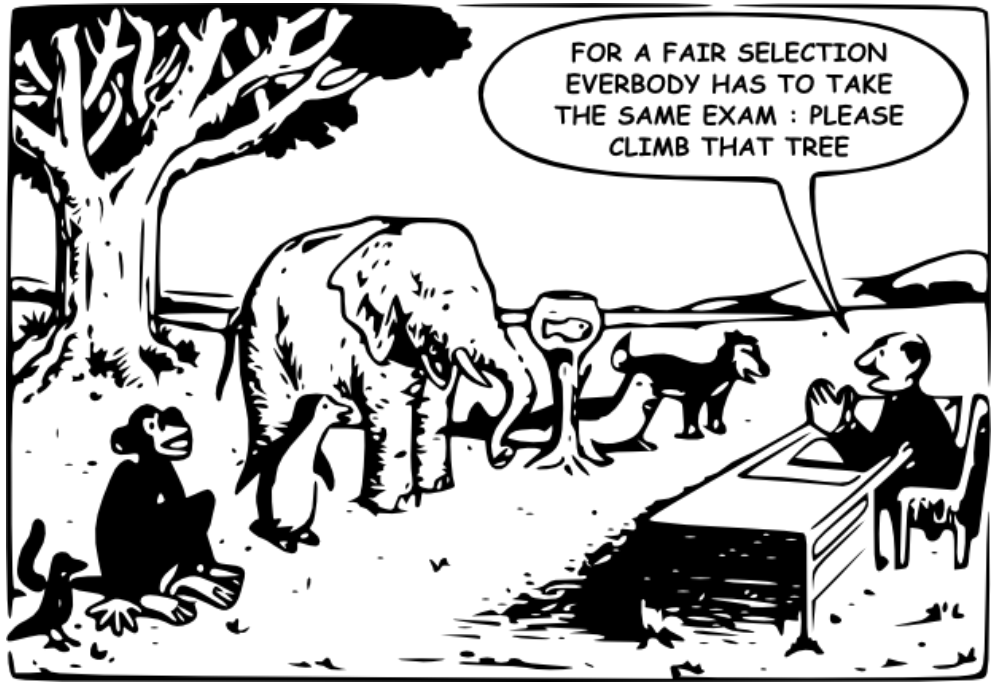
Reguli de Clean Code în metode

- Orice metodă e indicat să aibă cel mult trei niveluri de structuri imbricate (arrow code)
- Întotdeauna se va încerca ieșirea din funcție cât mai repede posibil (prin return sau excepție)
- Variabilele vor fi declarate cât mai aproape de utilizarea lor
- Încercați pe cât posibil folosirea *this* și stabilirea unei convenții de nume pentru parametrii constructorului



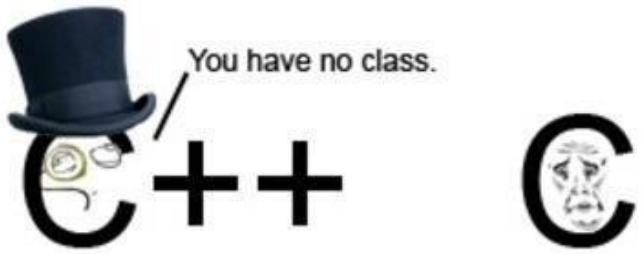
Reguli de Clean Code în metode

- Evitați metodele cu mai mult de doi parametri
- Evitați metodele foarte lungi (peste 20 de linii de cod)
- Complexitatea trebuie să fie invers proporțională cu numărul de linii de cod
- Atenție la ordinea în care tratați excepțiile



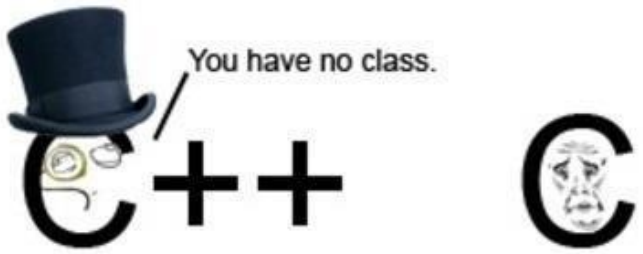
Reguli de Clean Code în clase

- Toate metodele dintr-o clasă trebuie să aibă legătură cu acea clasă
- Evitați folosirea claselor generale și mutați prelucrările respective ca metode statice în clasele aferente
- Evitați primitivele ca parametri și folosiți obiecte (clase Wrapper in Java) ori de câte ori acest lucru este posibil



Reguli de Clean Code în clase

- Atenție la primitive și în cazul prelucrărilor în mai multe fire de execuție
- Folosiți fișiere de resurse pentru șirurile de caractere din GUI
- Clasele ce conlucrează vor fi așezate una lângă alta pe cât posibil
- Folosiți-vă de *design patterns* acolo unde situația o cere



Reguli de Clean code în comentarii

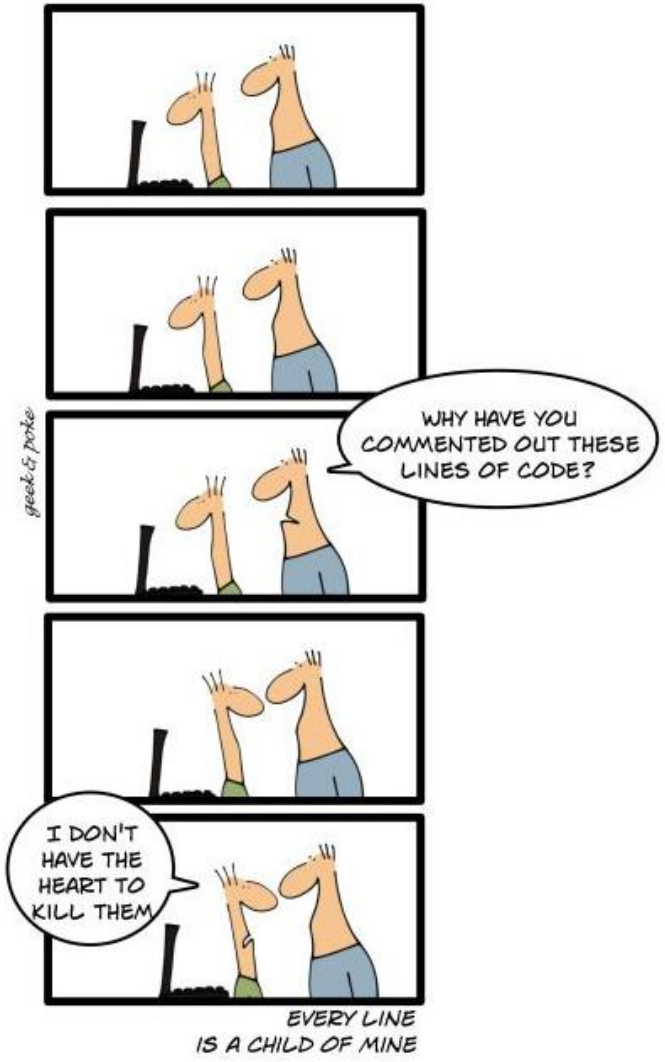
- De cele mai multe ori acestea nu își au deloc locul
- Codul bine scris este auto-explicativ
- Nu folosiți comentarii pentru a vă cere scuze

//When I wrote this, only God and I understood what I was doing

//Now, God only knows

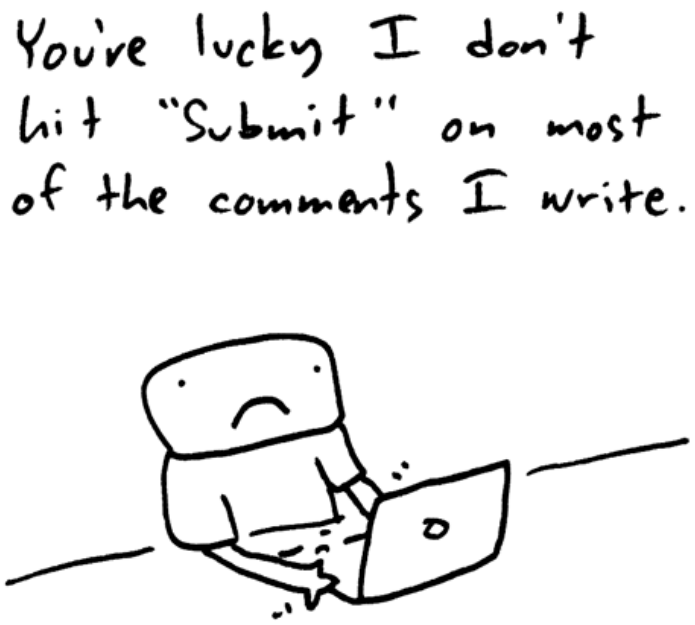
Reguli de Clean code în comentarii

- Nu comentați codul nefolosit – devine *zombie*
- Există soluții de versionare pentru recuperarea codului modificat
- Atunci când simțiți nevoia de a folosi comentarii pentru a face o metodă lizibilă, cel mai probabil acea funcție trebuie separată în două funcții



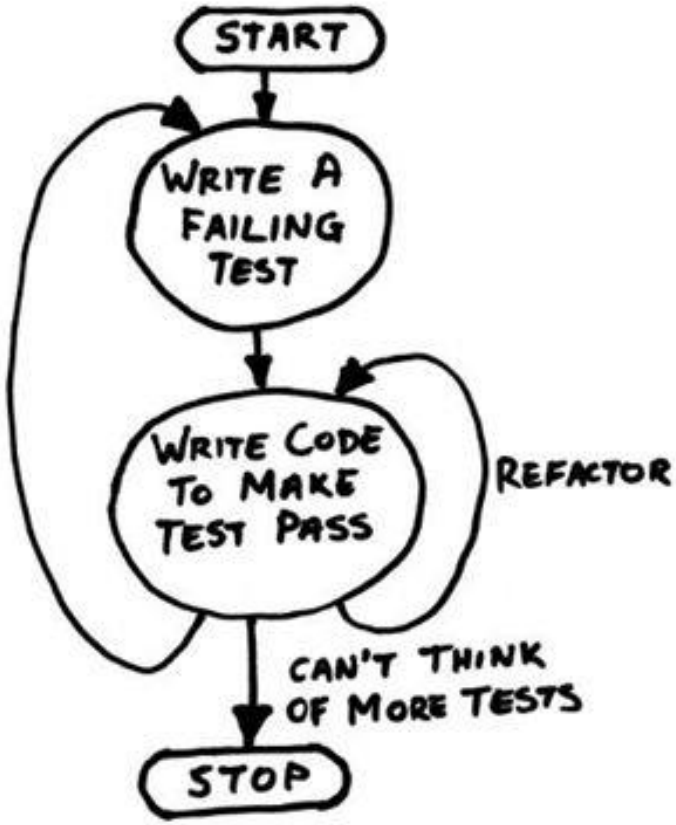
Reguli de Clean code în comentarii

- Evitați blocurile de comentarii introductive
- Toate detaliile de acolo se vor găsi în soluția de versionare
- Sunt indicate doar pentru
  - biblioteci ce vor fi refolosite de alți programatori (doc comments) - http://www.oracle.com/technetwork/articles/java/index-137868.html
  - TODO comments

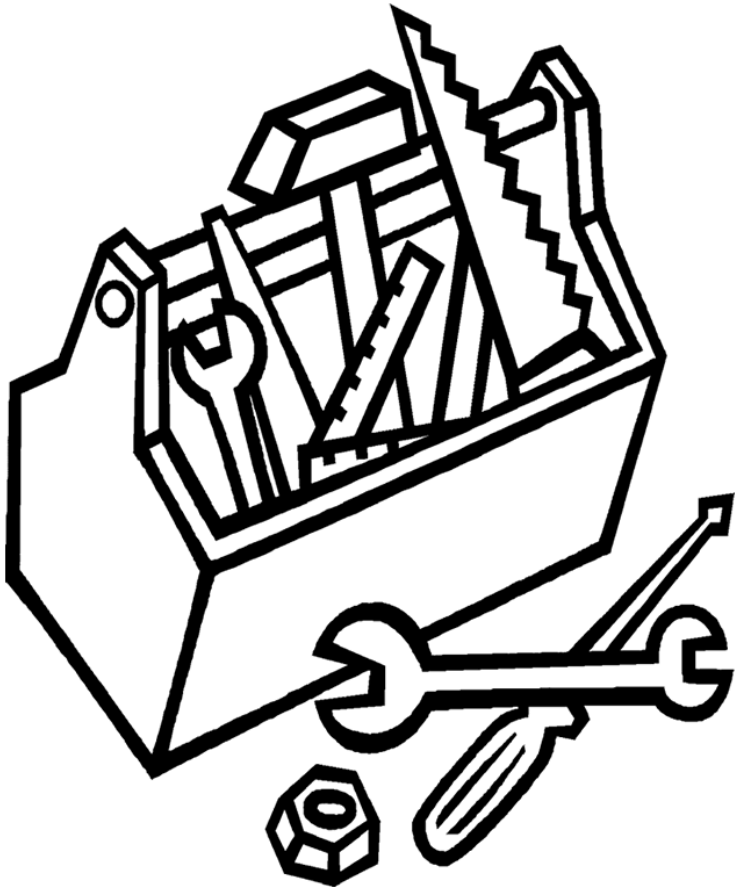


Scurt dicționar

- **Test Driven Development (TDD)** – Dezvoltare bazată pe cazuri de utilizare
- **Refactoring** – rescrierea codului într-o manieră ce se adaptează mai bine noilor specificații
- **Automatic Testing (Unit Testing)** – Testarea automată a codului pe baza unor cazuri de utilizare. Foarte utilă în refactoring pentru că putem verifica dacă am păstrat toate funcționalitățile sau nu (regression)
- **Code review** – Procedură întâlnită în special în AGILE (XP, SCRUM) ce presupune ca orice bucată de cod scrisă să fie revizuită și de un alt programator
- **Pair programming** – Tehnică specifică AGILE prin care programatorii lucrează pe perechi pentru task-uri complexe, pentru a învăța sau pentru a evita code review

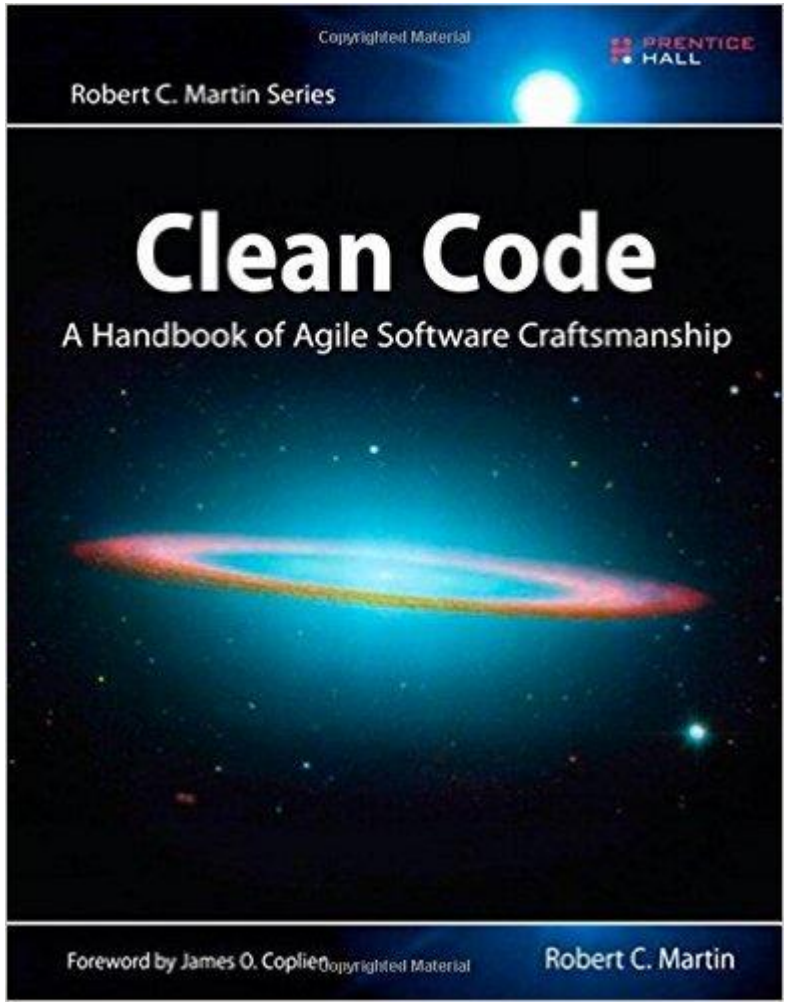


Instrumente



De citit

Robert C. Martin (Uncle Bob) – *Clean Code: A Handbook of Agile Software Craftsmanship*



Bonus

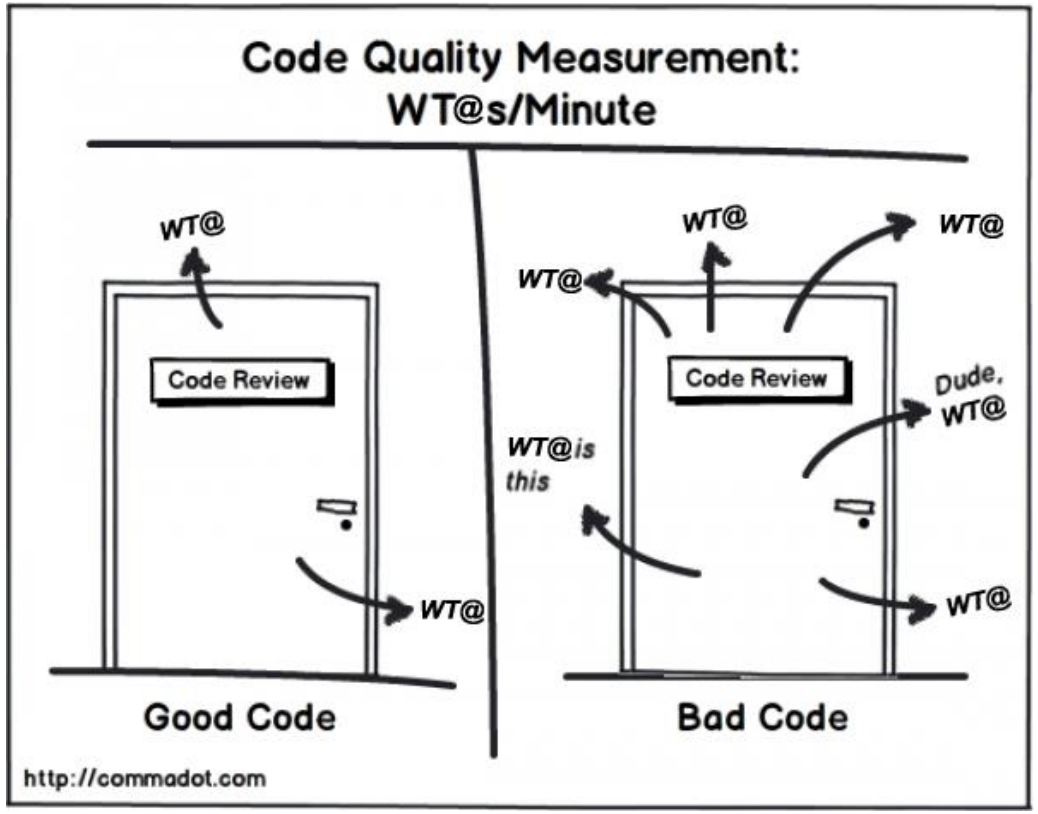
- The *Broken Window Principle*: clădirile cu ferestre sparte sunt mult mai vulnerabile la vandalism, care va duce la mai multe ferestre sparte;
- The *Boy Scout Rule*: lasați codul puțin mai curat decât l-ați găsit.
- Resurse suplimentare:

1. Robert C. Martin (Uncle Bob) – Clean Code: A Handbook of Agile Software Craftsmanship
2. Clean Code: Writing Code for Humans – Pluralsight series
3. Design Principles and Design Patterns”, Robert C. Martin

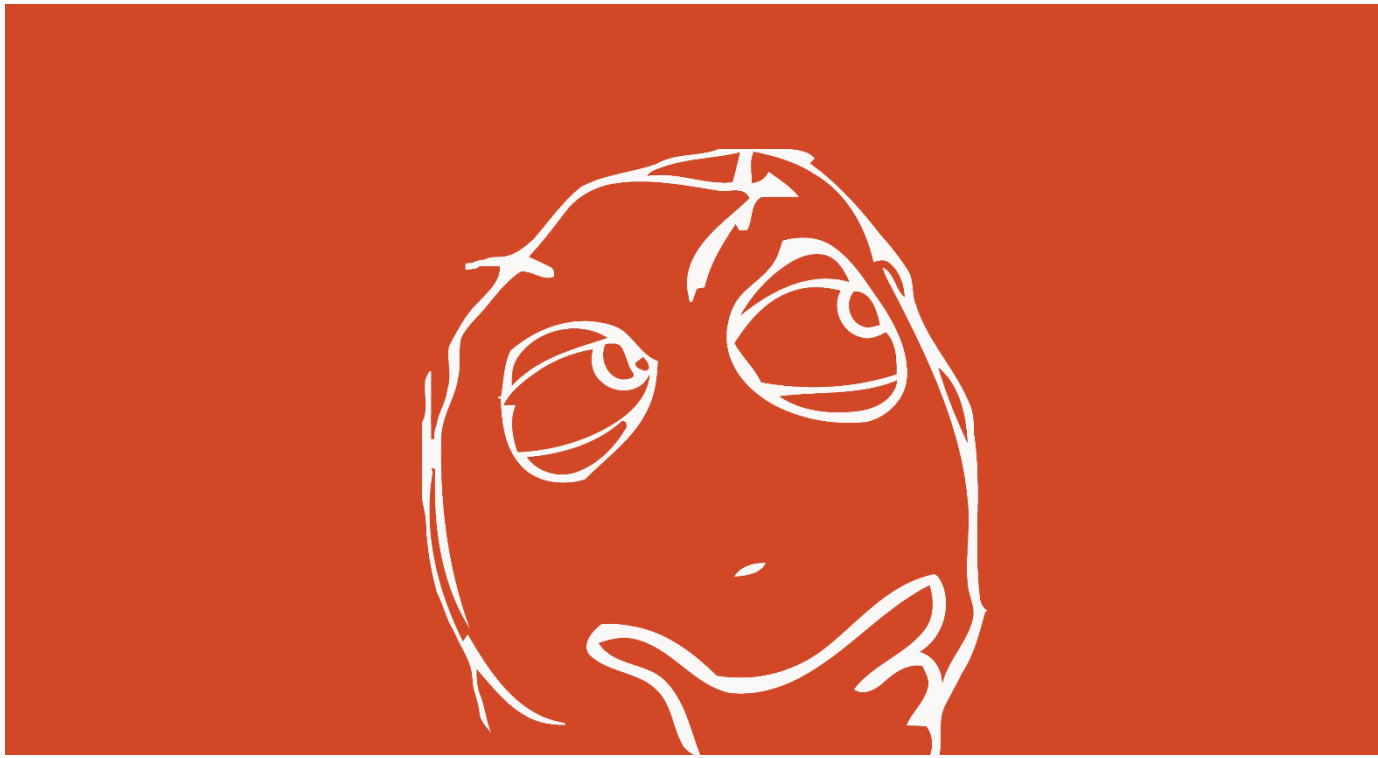
Încă ceva

- *"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."*

  Martin Golding



Întrebări?



Vă mulțumesc!