SOBRIÉTÉ DU CODE
< CLEAN CODE >

ENSIMAG

12/03/2024

Florent RIGOT

Damien GAVLAK

AU PROGRAMME

Sommaire

1. Presentation
2. Clean Code ?
3. Best practices
4. SOLID Principle
5. Dans la pratique
6. Greensight
7. Workshop

Le support vous sera transmis à la fin du cours

Règles

- On se tutoie
- Mettez vos smartphones de côté, et tout autre canal de communication
- Participez, donnez votre avis, faites part de vos remarques et interrogations (soyons constructifs)
- Respectez les opinions de chacun (bienveillance)

1

PRESENTATION

QUI SOMMES-NOUS ?

Florent RIGOT

- *DUT GEII*
- *DUETI GEII / informatique*
- *Master informatique*
  Alternant Génie logiciel

Développeur Back-End
Responsable Cybersécurité

Damien GAVLAK

- *IUT Informatique*
- *DU Robotique*
- *Alternant ingénieur*
  Informatique et Réseaux

Tech lead
(pôle innovation)

Développeur full stack

Company Confidential © Capgemini 2024. All rights reserved |

QUI SOMMES-NOUS ?

Pauline Bragas

- *Prépa CPBx*
- *Elève-ingénieure en Cognitique*

Stagiaire cheffe de projet
(pôle innovation)

Tanguy Cauquil

- *Prépa PC*
- *Eleve-ingénieur Informatique*

Stagiaire
Dev Full Stack
(pôle innovation)

Company Confidential © Capgemini 2024. All rights reserved |

2

CLEAN CODE

BUZZWORD ?

QUE DISENT LES EXPERTS ?

Quiz d'ouverture

60%

du coût d’une application
c’est sa maintenance…

Le vrai coût d’une application c’est la maintenance *(plus de 60 % du coût de l’application d’après une étude effectuée par le cabinet de conseil Accenture et publiée dans “How Software Maintenance Fees Are Siphoning Away Your IT Budget – and How to Stop It”)*

Company Confidential © Capgemini 2024. All rights reserved |

95%

du temps du développeur
c’est la lecture du code…

On passe 10 fois plus de temps à lire le code qu’en écrire (Robert C. Martin dans son ouvrage : « *Clean Code: A Handbook of Agile Software Craftsmanship* »)

Company Confidential © Capgemini 2024. All rights reserved |

UN CLEAN CODE …

C’est un code … propre … c’est-à-dire

- **Simple** et **concis**
- **Clair** et **lisible**
- **Efficace, Robuste** et **performant**
- **Maintenable** et **évolutif** (sans trop coûter)
- N’a pas beaucoup de **dépendances**
- Ses **interfaces** sont bien définies
- **Testé unitairement**
- **Réponds à la specification**

Clean Code est un des ***piliers*** de la ***Craftsmanship***

Company Confidential © Capgemini 2024. All rights reserved |

UN CLEAN CODE …

- **Clair** et **Lisible**

```java
public Boolean isAdult(final Integer age) {

    // Bad
    if (age >= 18)
        return true;
    else
        return false;

    // Better
    if (age >= 18) {
        return true;
    } else {
        return false;
    }

    // Good
    if (age >= 18) {
        return true;
    }

    return false;

}
```


Company Confidential © Capgemini 2024. All rights reserved |

UN CLEAN CODE …

- **Robuste et Maintenable**

```java
public String getCountryCode(final String birthplace) {

    // Bad
    if (birthplace == "Italy") {
        return "IT";
    }

    // Better
    if (birthplace.equals("Italy")) {
        return "IT";
    }

    // Good
    if (Objetcs.equals(birthplace, "Italy")) {
        return "IT";
    }

    ...

}
```


Company Confidential © Capgemini 2024. All rights reserved |

Clean Code – C’est Obligatoire

Le bug de l’an 2000 :

1 Janvier 1970 -> 01-01-70 ✓

1 Janvier 2000 -> 01-01-00 ✗

Company Confidential © Capgemini 2024. All rights reserved |

Clean Code – C’est Obligatoire

Le bug de l’an 2000 :

1 Janvier 1970 -> 01-01-1970 ✓

1 Janvier 2000 -> 01-01-2000 ✓

Company Confidential © Capgemini 2024. All rights reserved |

Clean Code - Avis de quelques experts

Bjarne Stroustrup

I like my code to be **elegant** and **efficient**. Logic must be **simple** so that bugs can hardly hide. It has **few dependencies** to ease maintenance. My code do **one thing**, and do it well.

Grady Booch

A clean code is **simple** and **direct**. It **can be read** as a good book. It does **not hide the intentions** of the designer. It has **clear abstractions** and **well-defined interfaces**.

Dave Thomas

A clean code **can be read** and **improved** by another developer that the original author. It has **unit tests** and **acceptance tests**. It has **few dependencies** and its **API is clear and minimal**.

(*) Answers copied from "Clean Code" by Robert C. Martin, aka Uncle Bob

Clean Code – Avis de Kent Beck

A Good Code …

- ✓ **Passes** all the **tests**
- ✓ Is **not redundant**
- ✓ Expresses the **intentions** of the developer
- ✓ **Reduces** the **number** of **classes** and **methods**

Author of XP (eXtreme Programming), co-author of Junit, contributor to the agile manifesto, promotor of TDD

Company Confidential © Capgemini 2024. All rights reserved |

3

CLEAN CODE

BEST PRACTICES

CLEAN CODE – KISS – KEEP IT SIMPLE & STUPID

On trouve aussi

- Keep It Short & Simple
- Keep It Simple & Straightforward

- Garder son code très simple
- Ne pas le complexifier pour la « *beauté* » du code
- S'il est simplement écrit, il est simple à comprendre

*"Everything should be made as simple as possible, but no simpler." -- **Albert Einstein***

*"Simplicity is the ultimate sophistication." -- **Leonardo da Vinci***

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – YAGNI – YOU AREN’T GONNA NEED IT

Coder **que** ce qui est **immédiatement nécessaire** (efficacité)

Ne **jamais** « prévoir » du code pour le **futur**

➔ But : Gagner du temps et réduire la complexité !

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – YAGNI – YOU AREN’T GONNA NEED IT

Evaluation de nouveaux frameworks

Décision sur le **design actuel** basée sur des **besoins futurs**

Abstraire les **dépendances externes**

Testing, **sécurité**, **scalabilité** et **exigences** fonctionnelles

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – YAGNI – YOU AREN’T GONNA NEED IT

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – DRY – DON’T REPEAT YOURSELF

Eviter au maximum les redondances / duplications de code (copier/coller)

Si un ensemble de code est répété à l’identique plusieurs fois dans le code, alors une fonction est nécessaire.

Once and Only Once (O3) peut être considéré comme un sous-principe de DRY.

La redondance n’est pas à bannir complètement. Parfois elle a du sens.

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – DRY – DON’T REPEAT YOURSELF

Exemple :

```java
public static int sum (int first, int second){
    if (first > 100 || first < 1 || second > 100 || second < 1){
        throw new IllegalArgumentException("Out of range");
    }

    return  first + second;
}

public static int multiply (int first, int second){
    if (first > 100 || first < 1 || second > 100 || second < 1){
        throw new IllegalArgumentException("Out of range");
    }

    return  first * second;
}
```

```java
public static int sum (int first, int second){
    validateRange(first, second);
    return  first + second;
}

public static int multiply (int first, int second){
    validateRange(first, second);
    return  first * second;
}

private static void validateRange(int first, int second) {
    if (first > 100 || first < 1 || second > 100 || second < 1) {
        throw new IllegalArgumentException("Out of range, must be between 1 and 100");
    }
}
```


Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SLAP – SINGLE LEVEL OF ABSTRACTION PRINCIPLE

Quoi : Qu’est ce fait le code ≠ Comment : Comment il le fait

Objectif : Cacher la complexité

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SLAP – SINGLE LEVEL OF ABSTRACTION PRINCIPLE

Exemple :

```
class CreateTask(
    private val clock: Clock,
    private val localStorage: LocalStorage,
    private val observers: List<TaskObserver>
) {

    fun invoke(description: String) {
        // normalize description
        val normalizedDescription = if (description.length > MAX_DESCRIPTION_LENGTH)
            description.substring(0, MAX_DESCRIPTION_LENGTH - 1) else
            description

        // create task
        val currentTime = clock.now()
        val initialStatus = Status.NotStarted
        val newTask = Task(normalizedDescription, initialStatus, currentTime)

        // save task
        localStorage.save(newTask)

        // notify
        val observingNotStarting = mutableListOf<TaskObserver>()
        for (i in 0..observers.size) {
            val taskObserver = observers[i]
            if (taskObserver.observedStatus == Status.NotStarted) {
                observingNotStarting.add(taskObserver)
            }
        }
        for (i in 0..observingNotStarting.size) {
            observingNotStarting[i].notify(newTask)
        }
    }
}
```

```
class CreateTask(
    private val clock: Clock,
    private val localStorage: LocalStorage,
    private val observers: List<TaskObserver>
) {

    fun invoke(description: String) {
        val newTask = createNewTask(description)
        localStorage.save(newTask)
        notifyAnyObservers(newTask)
    }

    private fun createNewTask(description: String): Task {
        val normalizedDescription = normalize(description)
        return Task(normalizedDescription, Status.NotStarted, clock.now())
    }

    private fun normalize(description: String): String {
        return if (description.length > MAX_DESCRIPTION_LENGTH)
            description.substring(0, MAX_DESCRIPTION_LENGTH - 1) else
            description
    }

    private fun notifyAnyObservers(newTask: Task) {
        observers
            .filter { taskObserver -> taskObserver.observedStatus == Status.NotStarted }
            .forEach { taskObserver -> taskObserver.notify(newTask) }
    }
}
```


Company Confidential © Capgemini 2024. All rights reserved |

4

CLEAN CODE

SOLID PRINCIPLES

CLEAN CODE – SOLID

Meta Acronyme composé de 5 principes orientés objet :

- SRP : Single Responsibility Principle
- OCP : Open-Closed Principle
- LSP : Liskov Substitution Principle
- ISP : Interface Segregation Principle
- DIP : Dependency Inversion Principle

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – SRP – SINGLE RESPONSIBILITY PRINCIPLE

Répondre toujours à un besoin à la fois

Exemple :

```
private static void calculateCellsWhereICanSpawnARobot() {
    myCellsWhereICanSpawnRobots = grid.stream()
        .filter(Cell::isOwnedByMe)
        .filter(Cell::isACellWhereICanSpawnARobot)
        .filter(Cell::isNotInARangeOfRecycler)
        .collect(Collectors.toList());
}

private static void calculateDistancesForTheOpCellsThatICanAttack() {
    opCellsThatCanBeAttacked = grid.stream()
        .filter(Cell::isOwnedByOp)
        .filter(Cell::doesNotContainUnits)
        .filter(Cell::isNotInARangeOfRecycler)
        .collect(Collectors.toList());
}

private static void calculateDistancesForTheNewCellsThatICanOwn() {
    xxCellsThatCanBeOwned = grid.stream()
        .filter(Cell::isActive)
        .filter(Cell::isNotOwnedByAnyone)
        .filter(Cell::isNotInARangeOfRecycler)
        .collect(Collectors.toList());
}
```


Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – OCP – OPEN-CLOSED PRINCIPLE

Code doit être compatible/ouvert à du code futur sans modifier le code existant

Architecture du code

Ce n’est pas incompatible avec YAGNI

Fonctionne bien quand DRY est respecté

Objectifs :

➔ Architecture du code compatible avec du code futur

➔ Pas de code tant que le besoin n’est pas immédiat et définit

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – OCP – OPEN-CLOSED PRINCIPLE

Exemple :

```java
package com.ezoqc.blog.solid.ocp;

public class UserAgeValidator {
  public boolean isOldEnoughToDrinkAlcohol(int age) {
    return age >= 18;
  }
}
```

```java
package com.ezoqc.blog.solid.ocp;

public class UserAgeValidator {
  public boolean isOldEnoughToDrinkAlcohol(int age, String provinceCode) {
    return provinceCode.equalsIgnoreCase('qc') && age >= 18
      || provinceCode.equalsIgnoreCase('on') && age >= 21
  }
}
```

```java
package com.ezoqc.blog.solid.ocp;

public interface UserAgeValidator {
  boolean isOldEnoughToDrinkAlcohol(int age);
}

public class QuebecAgeValidator implements UserAgeValidator {
  public boolean isOldEnoughToDrinkAlcohol(int age) {
    return age >= 18;
  }
}

public class OntarioAgeValidator implements UserAgeValidator {
  public boolean isOldEnoughToDrinkAlcohol(int age) {
    return age >= 21;
  }
}
```


Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – LSP – LISKOV SUBSTITUTION PRINCIPLE

Nom basé par sa créatrice Barbara Liskov

Pouvoir substituer dans une classe les propriétés de celles-ci

Principe de programmation orienté objet :

- Classes
- Interfaces
- Types et sous-types

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – LSP – LISKOV SUBSTITUTION PRINCIPLE

Exemple :

```java
public class User {
  private String emailAddress;

  public String getEmailAddress() {
    return this.emailAddress;
  }
}

public class AnonymousUser extends User {
  public String getEmailAddress() {
    throw new RuntimeException("Anonymous users don't have an email address.");
  }
}
```

```java
package com.ezoqc.blog.solid.lsp;

public abstract class User {

}

public class RegisteredUser extends User{
  private String emailAddress;

  public String getEmailAddress() {
    return this.emailAddress;
  }
}

public class AnonymousUser extends User {

}
```


Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – ISP – INTERFACE SEGREGATION PRINCIPLE

Un client utilisant une interface ne doit pas embarquer obligatoirement des méthodes inutiles

Objectif :

➔ Découper fonctionnellement les interfaces (et les méthodes qu'elles ont) selon la pertinence d'emploi par les clients

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – ISP – INTERFACE SEGREGATION PRINCIPLE

Exemple :

```java
interface Animal {

    void eat();

    void fly();

    void swim();
}

class Dog implements Animal {

    @Override
    public void eat() {
        System.out.println("Dog eats !");
    }

    @Override
    public void fly() {
        // Throws ERROR - dogs cannot fly
    }

    @Override
    public void swim() {
        // Throws ERROR - dogs cannot swim
    }
}

class Eagle implements Animal {

    @Override
    public void eat() {
        System.out.println("Eagle eats !");
    }

    @Override
    public void fly() {
        System.out.println("Eagle flies !");
    }

    @Override
    public void swim() {
        // Throws ERROR - dogs cannot swim
    }
}
```

```java
interface Animal {

    void eat();
}

interface AnimalWhichFlies extends Animal {

    void fly();
}

interface AnimalWhichSwims extends Animal {

    void swim();
}

class Dog implements Animal {

    @Override
    public void eat() {
        System.out.println("Dog eats !");
    }
}

class Eagle implements AnimalWhichFlies {

    @Override
    public void eat() {
        System.out.println("Eagle eats !");
    }

    @Override
    public void fly() {
        System.out.println("Eagle flies !");
    }
}
```


Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – DIP – DEPENDENCY INVERSION PRINCIPLE

Les abstractions ne doivent pas dépendre des détails

Mais, plutôt, les détails qui doivent dépendre des abstractions

Company Confidential © Capgemini 2024. All rights reserved |

CLEAN CODE – SOLID – DIP – DEPENDENCY INVERSION PRINCIPLE

Exemple :

```
const dbConn = //....

export function saveUser(user) {
    return dbConn.save(user)
}
```

```
export function saveUser(user, dbConn) {
    return dbConn.save(user)
}
```


Company Confidential © Capgemini 2024. All rights reserved |

FAIRE DU CLEAN CODE : COMMENT Y PARVENIR ?

- Best practices
  - KISS (Keep It Simple Stupid)
  - YAGNI (You Aren't Gonna Need It)
  - DRY (Don't Repeat Yourself)
  - SLAP (Single Level of Abstraction Principle)

- Des méthodologies
  - SOLID : 5 principes de design

- Bon sens
  - Cause racine (lors de la résolution des problèmes trouver toujours la cause racine)
  - Boy Scout Rule (laisser le camp plus propre que l'état dans lequel vous l'avez trouvé)
  - Hollywood Principle ("Don't Call Us, We'll Call You", le code répond à des évènements externes)
  - Tell don't ask (plutôt que de demander à un objet les données et agir dessus, déplacer la logique métier dedans)
  - Bien nommer son code (champs, méthodes, variables, paramètres, interfaces, classes, annotations, etc…)
  - Bien gérer les erreurs (exceptions au lieu des codes retour)
  - Commenter le code (juste ce qu'il faut ☺)
  - Ecrire les tests

Company Confidential © Capgemini 2024. All rights reserved |

FAIRE DU CLEAN CODE : COMMENT Y PARVENIR ?

Company Confidential © Capgemini 2024. All rights reserved |

5

DANS LA PRATIQUE

UN BON DEV ?

Capgemini

UN BON DEV

PRAGMATIQUE

SAIT ARITCULER LES CONCEPTS

CAPACITÉ DE RÉFLEXION

EXPRIMER SA PENSÉE

A DU BON SENS

S'ADAPTE À LA SITUATION

DANS LA PRATIQUE …

Code Nouveau suivant les bons principes

- Clean Code : écrire du bon code
- Clean Design : bien concevoir
- Clean Architecture : bien choisir sa(ses) structure(s) logicielle(s)

➔ Bannir la sur-ingénierie et la sur-spécification

Code Legacy : intervention suivant Boy Scout Rule

- On laisse le code plus propre que son état initial

➔ Rachat de la Dette Technique

Company Confidential © Capgemini 2024. All rights reserved |

DANS LA PRATIQUE …

Company Confidential © Capgemini 2024. All rights reserved |

100%

des développeurs ont l’obligation de faire du

- ✓ Clean Code
- ✓ Clean Design
- ✓ Green Code

Company Confidential © Capgemini 2024. All rights reserved |

CONCLUSION

Plus rapide pour corriger des bugs

Code plus performant

Company Confidential © Capgemini 2024. All rights reserved |

CONCLUSION

Company Confidential © Capgemini 2024. All rights reserved |

RÉFÉRENCES

Books

Clean Code: A Handbook of Agile Software Craftsmanship - Robert C-Martin

The Pragmatic Programmer — David Thomas and Andrew Hunt

Software Architect's Handbook — Joseph Ingeno

Extreme Programming Installed — Ron Jeffries, Ann Anderson, Chet Hendrickson

Agile Software Development, Principles, Patterns, and Practices

Green pattern manuel d'éco-conception des logiciels

https://www.lulu.com/shop/green-code-lab/green-patterns-manuel-d%C3%A9co-conception-des-logiciels/ebook/product-18848920.html?page=1&pageSize=4

Website
https://refactoring.guru/fr/design-patterns
https://www.pluralsight.com/courses/principles-oo-design
https://en.wikipedia.org/wiki/Single-responsibility_principle
https://blog.adimeo.com/forum-php-2019-clean-architecture
https://deviq.com/
https://blog.ippon.fr/2018/05/14/mon-avis-sur-le-livre-clean-architecture-par-robert-c-martin/
http://principles-wiki.net/principles:start
https://thevaluable.dev/dry-principle-cost-benefit-example/
https://dev.to/codemouse92/clean-dry-solid-spaghetti-1lgm
https://martinfowler.com/bliki/TellDontAsk.html
https://medium.com/javarevisited/slap-that-ugly-code-6ec276d3a4bc
https://le0nidas.gr/2020/12/26/slap-single-level-of-abstraction-principle/
http://blog.ezoqc.com/5-exemples-faciles-pour-comprendre-les-principes-solid/
https://www.javatpoint.com/solid-principles-java

Videos
Oncle Bob

https://www.youtube.com/watch?v=7EmboKQH8lM

Company Confidential © Capgemini 2024. All rights reserved |

6

GREENSIGHT

Impacts liés à 2 facteurs :

- **Multiplicité (34 Milliard)**
- **Renouvellement fréquent** *(encore fonctionnels)*

https://bit.ly/2ZloMzE - Paragraphe 2.4.3

© Capgemini 2020. All rights reserved

MARCHÉ

Trois besoins identifiés

Sensibiliser

Utiliser les référentiels publics existants pour sensibiliser les équipes aux bonnes pratiques d'éco-conception des logiciels et **identifier les règles les plus pertinentes** pour leur projet.

Mesurer

Rendre l'impact visible en permettant aux utilisateurs de le mesurer directement pendant la phase de développement du projet, afin d'anticiper et d'**agir le plus tôt possible**.

Industrialiser

Intégrer et **automatiser la vérification** des bonnes pratiques d'éco-conception des logiciels directement dans les outils de développement afin de faciliter et systématiser leur application par les équipes.

GREENSIGHT

Sonar

- 24 bonnes pratiques d'éco-conception Java
  Basées sur les référentiels du marché (INR, GreenIT, ADEME)
- Compatible avec 20+ langages sur demande
  Java, PHP, Python, Javascript…
- Interface utilisateur dédiée intégrée à Sonarqube
- Compatible à partir de Sonarqube 8.9.9 LTS
  Version adaptable sur demande
- Disponible sur la Production Line Capgemini
- Plug'n play avec guide utilisateur
  Support à l'installation disponible sur demande

Utilisable gratuitement

Afficher 10 entrées Recherche:

N°	Rubrique	Description
1	1. Spécification	Ne pas retenir les fonctionnalités non essentielles
2	1. Spécification	Quantifier précisément le besoin
3	1. Spécification	Supprimer les fonctionnalités non utilisées
4	1. Spécification	Privilégier une approche mobile first
5	2. Conception	Optimiser le parcours utilisateur
6	2. Conception	Valider le parcours utilisateur
7	2. Conception	Proposer un traitement asynchrone lorsque c'est possible
8	2. Conception	Respecter le principe de navigation rapide dans l'historique
9	2. Conception	Éviter les animations Javascript / CSS
10	2. Conception	Limiter le recours aux carrousels

Afficher 1 à 10 de 115 entrées Précédent 1 2 3 4 5 …

https://collectif.greenit.fr/ecoconception-web/115-bonnes-pratiques-eco-conception_web.html

Dashboard

Company Confidential © Capgemini 2024. All rights reserved |

Dashboard

Company Confidential © Capgemini 2024. All rights reserved |

Dashboard

Company Confidential © Capgemini 2024. All rights reserved |

7

WORKSHOP

Workshop Greensight – Allez sur Greensight

- Allez sur Sonarqube
- Sélectionnez un projet
- Cliquez en haut sur **More** puis **Greensight**

Company Confidential © Capgemini 2024. All rights reserved |

Workshop Greensight - Améliorez le projet

- Consultez l'analyse de sonarqube
- Repérez quelques erreurs et corrigez les
- Scannez à nouveau votre projet
- Vérifiez que l'erreur à bien disparue

Company Confidential © Capgemini 2024. All rights reserved |

Capgemini

GET THE FUTURE YOU WANT

capgemini.com