Software Craftmanship

Emmanuel Chaffraix

25 avril 2018

Table des matières

1 Soutenance

Le 3 juillet.
Soutenance : 10min sur un sujet donné par le prof
10min de restitution de cours.
Par groupe de 3.

2 Introduction

2.1 Prof

Ex-GISTR 2009.

2.2 Iceberg

La tech n'est que la partie émergée de l'iceberg.
SLA : qualité de service. (uptime, etc)

3 Clean code

3.1 Qu'es-ce que du clean code ?

— Bugs
— Coding style
— Travailler avec les autres (50, 100, 1000 personnes)

3.2 La vie du développeur

Jamais remercié, toujours des plaintes !

Par code :
— Fail compilation
— Segfault
— Stack overflow

Client :
— ASAP
— Bad UX
— Not working.

3.3 ... est géniale

On créé des produits
On décrouvre des activités
On apprend !

3.4 Software Craftmanship Manifesto

Pas seulement du logiciel qui marche, mais aussi du logiciel bien conçu.
Pas seulement répondre à du changement mais ajouter constemment de la valeur
Pas seulement des individus et des interactions mais une communauté de professionnels
Pas seulement une collaboration avec des clients, mais aussi des partenariats productifs.

3.5 Exemple

Éviter les nombres magiques, la duplication de code, les abbréviations non-évidentes. Prendre des noms clairs.
Quelle unité ? € ? £ ? $?
C'est de la dette technique.

3.6 L'impact de la dette technique

Lisibilité du code.
Développement de nouvelles fonctionnalités.
Correction de bugs.

C'est finalement le calcul de son impact en tant que développeur sur un projet.

3.7 Quelques principes généraux

3.7.1 Be KISS

Keep It Simple, Stupid.
Exemple : Le bon coin. Très peu de fonctionnalités.

3.7.2 Be DRY

Don't repeat yourself.
Pas de code similaire.

3.7.3 Be focused

YAGNI : You Ain't Gonna Need It.
Ne pas développer des trucs qu'on ne va pas utiliser.

3.7.4 DIE

Duplication Is Evil.
Copy/Paste is easy, but hard to bug fix.

3.8 Quelques principes OO

Encapsulation
Héritage
Polymorphisme
— Overloading
— Templates
— Subtypings
Liste : souvent tableaux mais avec des éléments de liste chaînée.

3.8.1 SOLID

— SRP : Single responsibility principle
  Une classe ne doit avoir qu'une seule raison de changer.
  Ne vérifier un email qu'à un seul endroit, par exemple.
  Séparer les actions effectuées par une fonction.

— OCP : Open/closed principle
  Les entités logicielles doivent être ouvertes à l'extension mais fermées à la modification.
  Exemple : gros switch case à modifier.
  Solution : stocker le résultat dans chacune des classes.

— LSP : Liskov substitution
Si S est un sous-type de T, alors les objets de type T peuvent être remplacés par des objets de type S sans changer aucune des propriétés désirables du programme.
Si on manipule les types, on n'utilise pas bien son langage OO.

— ISP : Interface Segregation Principle
Many client-specific interfaces are better than one general-purpose interface.

— DIP : Dependency Injection Principle
One should "depend upon abstractions, not concretions".
On ne doit dépendre que des abstractions de classes, pas des classes elles-mêmes.
Exemple : Si on a un loggeur partout, faire une interface qui permet de tout changer facilement.

4 Soyez Agiles

4.1 V Cycle

Compléter.
Niveau fonctionnel : Concept development Transition Operation maintenance.
Niveau system : Requirements Engineering. Test evaluation
Niveau Subssytem
Bien pour les projets longs où on a besoin de s'engager sur un résultat.

4.2 Agile

Meilleure satisfaction client : il a quelque chose d'utilisable tout le long.
Voir https://www.youtube.com/watch?v=3wyd6J3yjcs.
Voir les diapos.
Idea $\underset{Build}{\longrightarrow}$ Product $\underset{Measure}{\longrightarrow}$ Data $\underset{Learn}{\longrightarrow}$ Idea.
Voir l'entreprise qui répondait à des questions ouvertes en 2007, rachetée par Google. Ils ont complètement faké le truc : Mail, gestion de ticket, moteur de recherche sur les questions et mail sinon. Ils n'ont pas eu le temps de finir leur algo.
Financement de Critéo avec de la vente de céréales.
Méthode : Existe depuis les années 60 avec Ford et Toyota.
Méthode lean.

4.3 SCRUM

— Product owner
Doit pouvoir lister les fonctionnalités qu'il souhaite et comprendre les besoins du client.

— Scrum master
Est responsable de l'équipe (souvent un dev). Change régulièrement. Pas forcément le chef.

— Developer team

— Backlog
Do to list sur le projet. Ne fait que grossir. C'est tout ce que souhaite le client.

Poker planning : on met un prix / complexité sur chacune des fonctionnalités et on discute de quand on les implémente.

4.3.1 SCRUM Workflow

— Sprint meeting planning
Remplir le backlog. On chosit ce qu'on va implémenter.

— Daily Scrum meeting
Une fois par jour, état de l'avancement du projet, debout. On dit ce qu'on a fait, ce qu'on va faire et là où on bloque.

— Sprint retrospective
Voilà ce qu'on a fait par rapport à ce qu'on avait prévu.

4.3.2 Scrum diagram

Voir le diagramme sur les déiapos du prof.

4.4 Kanban

Voir le diagramme.
Chacun a une couleur de ticket.
Intérêt : on limite le nombre de tickets par personne et par colonne.

5 Think test

5.1 Intro

TDD : Test driven approach
BDD : Behaviour driven approach

5.2 Why do we test ?

Tests help to understand requirements
Tests pretoects future developements
Tests are a fall protection
We are human after all.

5.3 How do we test ?

We create code to test our code

— Unit test : Focus on ONE function usage, mock dependencies. Ex : base de données : on en prend une fausse.
Developer.
— Integration test : crosses the boudnary between components.
Developer.
— Behaviour test : An example of the user using the system. Test a usecase
Scrum master + product owner.

5.4 TDD : Test Driven Development

No code without a test.

1. Write a test that fails
2. Write the code that fulfills the test.
3. Refactor source code (feature and test !)

Compliqué à mettre en place pour les personnes qui ne sont pas habituées à faire du test.
Ralentit le développement.

5.5 BDD : Behaviour Driven Development

A simple test : Given... When... Then...
Given a user with an account
When he tries to create a new account with existing account credentials
Then it must be logged in with the existing account
A feature contains tests.
Voir https://cucumber.io/

5.6 Think test

Write a failing feature test

5.7 The double loop

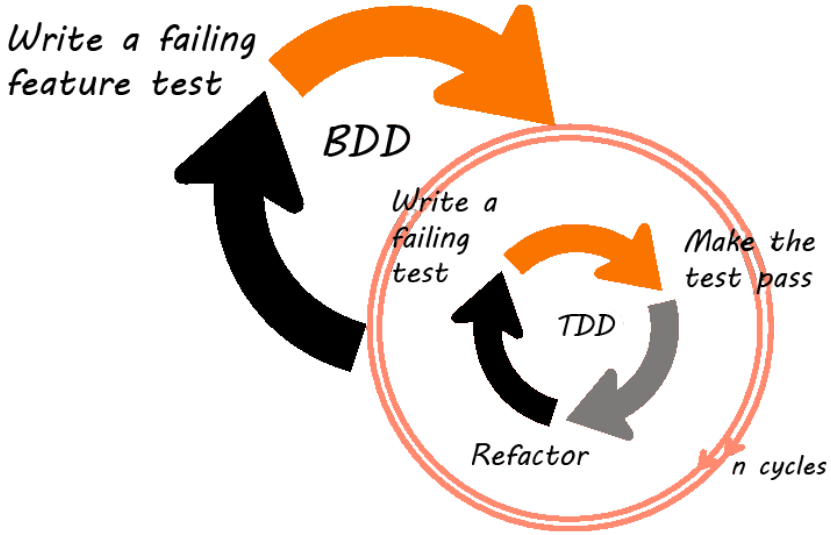

FIGURE 1 – The double loop

5.8 Test pyramid

UI > Service > Unit.
In time and money.

5.9 What is a good test ?

Easy to understand, clear when fails.
Determinist
Focused
AAA : Arrange, Act, Assert.

5.10 Test coverage

A 100% coverage is not a bug-free code.

6 From dev to prod

6.1 Working with others

Avoid :
— Missing code files
— Ugly code naming
— Code deletion
Anticipate problems
So rules are required !
— Coding style
— Code reviewal
— Automated process
Infinite loop : Code, Build, Test, Release, Deploy, Operate, Measure, Plan.

7 Software architectures

7.1 Monolyth

Fine for simple projects.

7.2 Multi-tier

7.2.1 3-tier

Presentation
Logic (for the job ex : age calculation).
Data.

Typically for local software.

7.3 SOA : Service Oriented Architecture

7.4 CRUD

7.5 REST

Standard API : swagger.

7.6 The current state problem

7.7 Event sourcing

Cucurrent environment (multithreading, processing, etc) : try to work with immutable objects (perf, corruption).