# SOFTWARE CRAFT

## TDD, Clean Code et autres pratiques essentielles

Cyrille Martraire
Arnaud Thiéfaine
Dorra Bartaguiz
Fabien Hiegel
Houssam Fakih

## Développer ?

*"Working code is a low bar."*

- **Un logiciel n'est jamais fini** — Il change en permanence
- **C'est lire et comprendre le code** — Au moins autant que l'écrire
- **Définir le besoin** — Aussi difficile que d'écrire le code correspondant
- **Pas de hacker** — Ni de virtuose

### Une communauté

Bienveillante, Aider, Transmission

Pas d'élitisme

### Boîte à outils

Test-Driven Development, Behavior-Driven Development, Clean Code, Domain-Driven Design, Legacy remediation, Pair / Mob Programming, OO, FP, SOLID

## Développement dirigé par les Tests (TDD)

Ecrire 1 test qui échoue

Améliorer le code — Meilleure lisibilité

Faire passer le test — le plus rapidement possible

Réduire les peurs — Décomposer le problème

Prendre le temps de rendre le code lisible — Refactoring

Aide à définir 1 objectif à atteindre

Ecrire le moins de code possible

### Nos tests

- Retranscript la règle de gestion
- Facile de déterminer cause de l'échec

***Should / When***
***Given / When / Then***

```
DivideShould.Throw_an_invalid_operation_when-denominator_is_zero
```

### Les 3 règles - Uncle Bob

- On doit écrire 1 test qui échoue avant d'écrire n'importe quel code de production
- On ne doit pas écrire plus de tests que ce qui est nécessaire pour échouer (ou ne pas compiler)
- On ne doit écrire que le code suffisant pour que le test actuellement en échec réussisse

Programmer **efficacement** des fonctionnalités **complexes**

## Techniques et principes de propreté de code

*"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."* – Martin Fowler

1 exercice de communication

Dégradation dans un environnement urbain

Si pas réparée : entraine d'autres dégradations

Propension au laisser-aller

Face à 1 environnement dégradé

Théorie de la vitre brisée

Règle du boy scout

KISS — Keep It Simple and Stupid

YAGNI — You Ain't Gonna Need It

Être **exemplaire**

### 4 règles pour 1 design simple - Kent Beck

Passer les tests — fonctionner comme prévu

Révéler l'intention — faire preuve de clarté

Eviter la duplication — DRY

Rester petit — ne conserver que ce qui est important

### Mettre l'accent sur le nommage

Exprimer l'intention
- Expliquer ce qu'on cherche à faire
- Pourquoi on veut le faire

s'appuyer sur le métier

Noms communs, Verbes, Renommage

### Commentaires avec parcimonie

Comment exprimer la même connaissance à travers le code ?

Souvent 1 code smell — Souci de propreté ?

*"Don't comment bad code - rewrite it." - Brian Kernighan*

Signaler une subtilité — Optimisation par exemple

Marquer des problèmes — TODO / FIXME

Mentions légales

### Découper de façon à faciliter découverte + navigation

S'applique également aux tests
- La qualité des test doit être exemplaire
- Si mettre autant d'effort voire plus

Storytelling
- Notre code doit raconter une histoire
- Devrait se parcourir comme une table des matières

Bien formater son code
- Respect du standard défini par l'équipe
- Niveau maximal d'indentation limité (2 max)
- Lignes de code pas très larges

Découper les fonctions
- Se focaliser sur le service rendu
- Ne doit faire qu'une seule chose
- Peu de paramètres
- 1 seul niveau d'abstraction

Simplicité contractuelle
- 1 seul paramètre en sortie
- Encapsuler in/out complexe dans 1 type dédié
- Considérer chaque paramètre comme immuable
- Définir les variables au plus près de leur utilisation

## Spécifications agiles avec le développement dirigé par le comportement (BDD)

### 3 Amigos

Discuter des fonctionnalités à construire

Représente le besoin — Recherche de valeur

Représente la mise en oeuvre — Faisabilité technique

Challenger les 2 autres — Identifier failles de raisonnement — Problèmes dans l'expression des besoins

### Exemples concrets

Dans le langage du métier

- Exprimer le besoin
- Détails concrets
- Auto-suffisants

Identifier scénarios-clés

LIVE STREAMING — D'incréments de spécifications — Ateliers brefs — 30 min / jour

### Automatisation

Syntaxe Gherkin

**Dev** — Critères d'acceptation

**Ultérieurement** — Tests de non régression

### Bénéfices

Critères d'acceptation — pour déterminer l'avancement des développements

Compréhension partagée entre tous

Tests de non régression — avec bonne couverture fonctionnelles

Documentation vivante — évolutive

```
Feature: Offrir des remises aux clients fidèles
  Afin d'encourager les clients à revenir acheter,
  En tant que vendeur de livres,
  Je souhaite offrir des remises de prix aux clients fidèles
Example: Aucune remise en deçà de 50 EUR d'achat
  Given un panier avec 2 articles pour un total de 49 EUR
  When le client consulte son panier,
  Then le total n'inclut aucune remise
```

*"Having conversations is more important than capturing conversations is more important than automating conversations" – Liz Keogh*

Améliorer **l'efficacité** de la **collaboration** entre les spécialistes impliqués afin de construire de meilleurs logiciels au meilleur coût.

## Collaborer efficacement avec le pair / mob programming

### Binômage (Pair Programming)

Alternance fréquente — Savoir ralentir

**Pilote (Driver)**
- Celui / Celle au volant
- Contrôle la machine

**Copilote (Navigator)**
- Agit pas directement sur la machine
- Prend du recul

### Pourquoi ?

Apprendre et progresser

Partager et transmettre — Réduire le Truck Factor

Assurer et se rassurer

Se motiver / s'entraider

### Plusieurs styles

**Ping-Pong**
- TDD + Pair
- Alterner à chaque phase rouge

**Strong Style**
- "J'ai une idée"
- "Voici le le clavier"

**Silent** — Seul le code pour communiquer

**Evil Twin**
- Piéger son partenaire
- Mettre en avant les cas limites

Respect du partenaire — Transparence — Objectif partagé — Rétrospective — Faire des pauses — Agenda — Savoir dire non

QUICK TIPS

Silence — Différentes machines — Rôles figés — Désengagement

### Ensemble / Mob programming

Collaboration simultanée de chaque membre

*"Tous les esprits brillants travaillent sur la même chose, en même temps, au même endroit et sur le même ordinateur" - Woody Zuill*

## Travailler avec du code legacy

"Un code ne disposant pas de tests"

Avoir le bon état d'esprit — Garder son calme, Rester focus

Réaliser état des lieux — Se faire une idée générale du code

Présence de tests ? — Oui → Enrichir les tests → Procéder au refactoring

Non → Code testable ?

Code des tests **se conforme** au comportement du **code de production**

Code testable ? — Oui → Ecrire les tests → Procéder au refactoring

Non → Rendre le code testable — Isolation des dépendances — **Seams**

Ecrire les tests → Approche par observation du code — Face à un code cryptique

Ecrire les tests → Approche par compréhension métier — Face à un code plutôt clair

Couverture, Mutation Testing, Fiabilité

Construire 1 **Golden Master**

*"Tester en premier les branches les moins profondes du code, refacturer en premier les branches les plus profondes" - Sandro Mancuso*

## Techniques de refactoring

*"Consiste à retravailler un code source sans en modifier le comportement, afin d'en améliorer la lisibilité et de le rendre beaucoup plus facile à maintenir / accueillir des évolutions"*

**Antidote de la dette technique**
- Ne pas la laisser s'accumuler
- Pratique régulière et continue

**Des freins**
- Pas de culture de la qualité du code dans l'équipe
- Peur de provoquer des régressions
- 1 poste de coût du point de vue client
- Manque de connaissance savoir-faire

### Comment ?

Transformer par petits pas

**Tests** = pré-requis

***Refactorer aussi le code de test***

Inversion de lignes, Renommage, Réécriture de boucles, Extraction, Changement de signature, Déplacement, Réécriture conditions, Inlining

## Principes et outils pour tester efficacement

### Doublures de tests

- Analogie au monde du cinéma
- Permettent de garantir la répétabilité

Test Double

Dummy Object — Objet fantoche

Test Stub — Imiter le comportement souhaité pour un test

Test Spy — S'assurer nombre d'appels, args, ...

*"Only mock code you own"*

Mock Object — Objet préprogrammé — Vérifie attentes bien réalisées

Fake Object — Vraie implémentation

*Créer couches intermédiaires*

### F.I.R.S.T

**Fast**
- Rapide
- Inciter à lancer les tests

**Répétable**
- Exécutable à l'infini
- Sur différents environnements

**Isolated/Independent** — Ne pas dépendre les uns des autres

**Self verifying** — Succès ou échec

**Timely/Thorough**
- Opportun et précis
- Couvre qu'un 1 seul cas d'utilisation

### Integration Tests

**Narrow**
- Limiter à quelques composants
- Substitue certaines dépendances

**Broad**
- Utilisent des instances réelles
- DB par exemple

## Outils et techniques avancées de TDD

Bottom-up — Chicago school — **Inside-out** — Développer par les fonctionnalités feuilles

Top-down — London school — Mockist approach — **Outside-in**
- Accent sur le besoin utilisateur
- Dépendances substituées

### ATDD et double boucle

Failing Acceptance Test → Failing Unit Test → TDD Loop → Green → Refactor → ATDD Loop

Acceptance Test Driven Development

## Techniques de conception

Principes SOLID

Tell don't ask

Principe d'Hollywood — Don't call us, we'll call you — Hooks / Callbacks

Loi de Demeter — Ne parlez qu'à vos amis immédiats

Approche fonctionnelle

...

Composition over inheritance

Faible couplage forte cohésion

Dépendance du spécifique vers le générique

API fluente

Builder

- 1 seul niveau d'indentation par méthode
- Ne pas utiliser le mot clé **else**
- Encapsuler tous les types primitifs et chaines de caractères
- Créer des classes encapsulant les collections
- Un seul point par ligne
- Ne pas abréger
- Garder toutes les entités petites
- Pas de classes avec plus de 2 attributs
- Pas de getter ni de setter

**Object calisthenics**

## Au delà des pratiques, un état d'esprit

### Développer son savoir-faire

- Progression par la répétition : kata
- Organiser sa veille
- Transmission / Compagnonnage
  - Accompagner les moins expérimentés

### Cultiver son savoir-être

- Valeurs : humilité, bienveillance, partage
- Egoless programming

### Faire preuve de professionnalisme

- Pragmatisme
- Amour du travail bien fait
- Négociation / compromis

## Introduire le craft dans votre contexte

Echanger avec autres

Participer à des meetups

Vidéos / Podcasts

Progresser kata par kata — FizzBuzz, Tennis, RomanNumerals, Gilded Rose, TripService, ...

**Individuellement**

### Dans l'organisation

Initier une dynamique → Porter par des sponsors

1 atelier par semaine
- Code kata
- Visionner vidéo
- Personnes motivées

Nécessite pas de soutien

Intensifier / généraliser — Atteindre personnes à l'écart

Exemples d'action
- Compétences craft dans fiches de poste
- Recruter dévs enthousiastes sur le craft
- Réserver du temps pour pratiquer
- Recruter / faire intervenir des coachs craft
- ...

### Convaincre les sponsors

Economique
- Temps de cycle
- Réduction du nombre de défauts
- Fréquence de livraisons
- Vitesse de remédiation

Evolution culturelle — Faire évoluer les équipes vers des enjeux d'actualité

Benchmarking — Se comparer aux autres sur le craft

Marque employeur
- Aider à recruter
- Conserver des compétences