

SD - R5.VCOD.06 - Développement logiciel

Chapitre 1 : Principes généraux

Enseignant : Stéphane JOYEUX, DSI Global Services (DSI-Group)

Ingénieur en Informatique et Responsable Technique



Présentation

Enseignant



**Responsable
Technique**

DSI - 2018

**Stéphane
JOYEUX**

51 ans / 4 enfants

**Ingénieur
Informatique**

EiCNAM - 2017

**Professeur
Vacataire**

IUT / Metz - 2009



/stéphane-joyeux-a071696



stephane.joyeux@dsi-globalservices.fr

stephane.joyeux@univ-lorraine.fr

Formation Initiale

• Brevet des collèges : Juin 1988

• BAC C (*Maths & Physiques*) : Juin 1992

• **D.U.T. Informatique** : **Juin 1995**



• D.E.S.T. Informatique : Juin 2005

• **Diplôme d'ingénieur** : **Juin 2017**

Option système et réseau
(*Cours du soir CNAM - EiCnam*)



Nom de Zeus ! 29 ans !



Le programme du module (26h).

Compétence ciblée :

– Développer un outil décisionnel.

SAÉ au sein desquelles la ressource peut être mobilisée et combinée :

– SAÉ 5.VCOD.01 | Analyse et conception d'un outil décisionnel.

Descriptif :

L'objectif de cette ressource est de présenter les approches et les outils qui, au-delà du langage de programmation même, permettent de mener à bien un projet informatique


Contenus :

– Cycle de vie d'un projet informatique (*conception, UML, AGL*)

– Gestion de projet informatique (*méthode agile*)

– Utilisation d'un outil de versionning (*git par exemple*).

La formation intègre des apports en matière d'ingénierie logicielle visant à faciliter l'intégration d'équipes de développement par nos étudiants en leur donnant les fondamentaux du domaine et en les sensibilisant aux bonnes pratiques et outils dédiés.

A man in a dark suit and light shirt stands against a background of a city skyline. The right side of his body is overlaid with a digital, pixelated cityscape. The background is a faded, high-angle view of a city with many skyscrapers, including the Empire State Building.

1. La qualité au coeur du développement.

Principe Général

« Build Quality In »

La Qualité est non-négociable !!

- Par exemple, chaque itération Sprint donne lieu à un :

« *Potentially shippable product increment* »

→ Produit incrémental potentiellement livrable ...



Coût de la non Qualité



<http://www.test-recette.fr/generalites/qualite-logicielle/cout-non-qualite.html>

Qualité Globale

- ✓ La qualité intrinsèque du produit (de son code).
- ✓ La qualité fonctionnelle (l'absence de bugs et le comportement attendu).
- ✓ La qualité du produit (répond au besoin réel).
- ✓ La qualité du processus (time-to-market, visibilité, productivité, efficacité ...).
- ✓ La qualité des relations humaines (reconnaissance, motivation, implication, satisfaction).

Pratiques d'ingénierie



S'adapter à l'évolution des exigences, des priorités, du planning, des pratiques mises en œuvre.



Simplicité. L'art de maximiser le travail non fait est essentiel.



Une attention et des moyens permanents pour garantir l'excellence technique.

A man in a dark suit and light shirt stands against a background of a dense city skyline, likely New York City. The skyline is rendered in a semi-transparent, digital style, appearing as if it's being projected or layered over the man's right side. The man is smiling and looking towards the right. A dark horizontal bar is positioned across the middle of the image, containing the text.

2. La rétroconception.

Définition

La rétro-conception (*aussi connue sous le nom d'ingénierie inversée ou de rétro-ingénierie*) est une méthode qui tente d'expliquer, par déduction et analyse systémique, comment un mécanisme, un dispositif, un système ou un programme existant, accomplit une tâche sans connaissance précise de la manière dont il fonctionne.

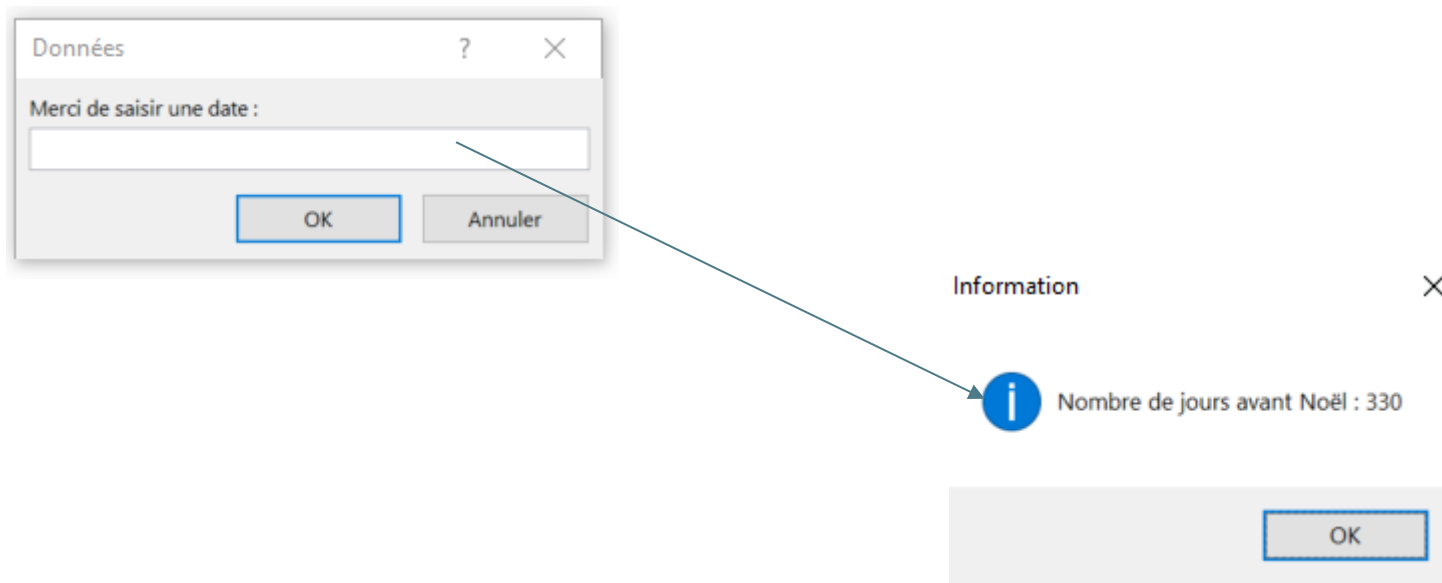
La rétro-ingénierie s'applique notamment dans les domaines de l'ingénierie mécanique, **informatique**, électronique, chimique, biologique et dans celui du design.

Le terme équivalent en anglais est reverse engineering (*ou retro-engineering*).

<https://fr.wikipedia.org/wiki/R%C3%A9tro-ing%C3%A9nierie>



Exemples en VBA



- ✓ En « déduire » le programme **VBA** correspondant en « *déroulant* » le programme.
- ✓ Ecrire le programme correspondant au scénario affiché en **VBA**.

Comment vérifier ?

- ✓ La première vérification consiste à exécuter le programme et constater qu'il se déroule selon le scénario établi.
- ✓ Comment l'exécuter à l'infini ?

Le programme Initial

Option Explicit

```
Public Sub AfficherNombreJoursAvantNoel()  
    ' 1. Déclaration et type des données I/O :  
    Dim donneeSaisieIN As String  
    Dim resultatOUT As Long  
  
    ' 2. Demande à l'utilisateur de saisir une date :  
    donneeSaisieIN = Application.InputBox _  
        ("Merci de saisir une date : ", "Saisie", Type:=2)  
  
    ' 3. Réalisation du calcul :  
    resultatOUT = DateDiff("d", CDate(donneeSaisieIN) _  
        , CDate("25/12/2022"))  
  
    ' 4. Affichage du résultat :  
    MsgBox "Nombre de dodo(s) avant Noël :" & resultatOUT  
  
End Sub
```


A man in a dark suit and white shirt stands against a background of a city skyline, with the Empire State Building prominent. A digital, pixelated cityscape is overlaid on the right side of the image, appearing to be part of the man's silhouette. A dark horizontal bar is positioned across the middle of the image, containing the text.

3. Git – gestionnaire de versionning.

Avant de commencer

https://gitlab.univ-lorraine.fr/users/sign_in



GitLab Community Edition

LDAP Standard

Nom d'utilisateur

Mot de passe

☐ Se souvenir de moi

Connexion

<https://gitlab.univ-lorraine.fr/>

Systeme de gestion de version

Introduction

Le but des systèmes de gestion de version (*version control systems*) est principalement double :

1. Garder un historique des différentes versions d'un projet (au sens large)
2. Faciliter la collaboration entre plusieurs personnes travaillant sur le même projet

Leur rôle est d'automatiser ces tâches et de les rendre efficaces.

Historique des versions

L'historique des versions est un enregistrement des modifications apportées à un projet au fil du temps. Il permet de revenir à une version antérieure du projet, de comparer deux versions, de voir qui a fait quoi, etc.

Collaboration

La collaboration est facilitée par la possibilité de travailler sur le même projet en même temps, sans risque de conflit. Les modifications sont enregistrées dans un dépôt centralisé, et chaque personne peut récupérer les modifications des autres.

Git c'est quoi ?

Git

Git est un système de gestion de version distribué. Il permet de gérer des projets de toutes tailles, de collaborer avec des milliers de personnes, et de gérer des projets de manière efficace.

Nous allons maintenant présenter quelques notions de base de git.

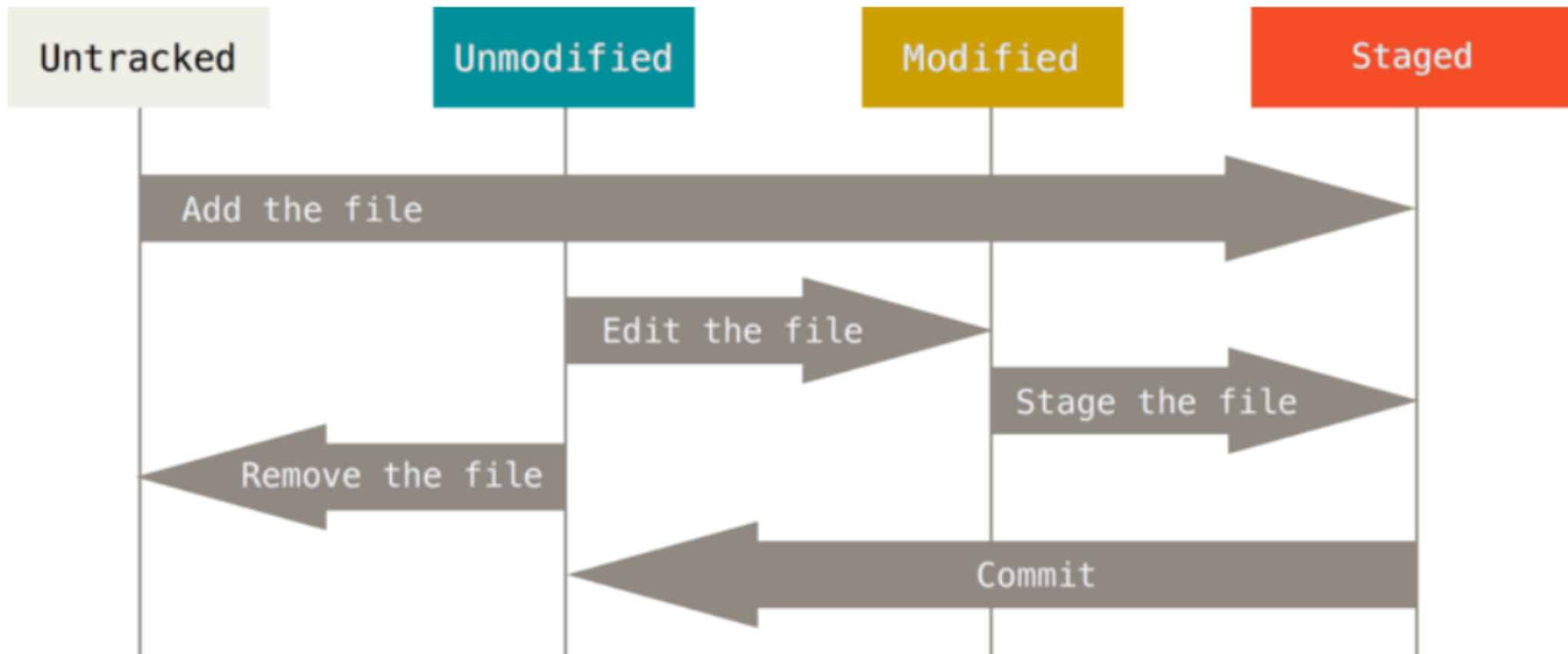
Dépôt

Un dépôt est un ensemble de fichiers et de répertoires, ainsi que l'historique des modifications apportées à ces fichiers. Il est possible de créer un dépôt à partir d'un dossier existant, ou de cloner un dépôt existant.

Localement, un dépôt est un dossier contenant un dossier `.git` qui contient l'historique des modifications apportées au dépôt.

Remarque : l'accès et la modification du dépôt doit se faire (sauf rares exceptions) via les commandes git.

Cycle de vie



Commit

Commit

Un **commit** est une modification apportée à un dépôt. Il est possible de faire plusieurs commits dans un dépôt, et de revenir à une version antérieure du dépôt en annulant les commits.

Un commit est composé de :

- Un message de commit
- Un auteur
- La date de création
- Un identifiant unique
- Les modifications apportées au dépôt

Index et verbes

Index

L'index est un espace temporaire contenant les modifications prêtes à être *commitées*. Ces modifications peuvent être :

- création de fichier
- modification de fichier
- suppression de fichier

Verbes

Les verbes de git sont les commandes permettant de manipuler le dépôt. Les verbes les plus utilisés sont :

- `git init` : créer un dépôt
- `git clone` : cloner un dépôt
- `git add` : ajouter des modifications à l'index
- `git commit` : créer un commit
- `git push` : envoyer les commits sur le dépôt distant
- `git pull` : récupérer les commits du dépôt distant
- `git status` : afficher l'état du dépôt
- `git log` : afficher l'historique des commits

Initier un dépôt GIT

Identifiants (à faire une seule fois)

Pour pouvoir utiliser git, il faut configurer son nom et son email. Cela permet de savoir qui a fait quoi.

```
git config --global user.name "John Doe"  
git config --global user.email "john.doe@email.com"
```

Aide

Pour obtenir de l'aide sur une commande, il suffit d'ajouter `--help` à la fin de la commande.

```
git add --help
```

Création d'un dépôt

Création d'un dépôt git

Pour mettre en place un dépôt, il y a principalement deux manières :

1. Faire d'un répertoire existant un dépôt Git (ce qui est expliqué en cours)
2. **Cloner** un dépôt existant (*quelque part*) (ce qui sera fait en TD)

Création d'un dépôt git local

Pour créer un dépôt git, il faut se placer dans le répertoire à versionner et exécuter la commande `git init`.

```
% cd /User/njozefow/TypeScript
% git init
Initialized empty Git repository in /Users/njozefow/TypeScript/.git/
% ls -la
total 0
drwxr-xr-x   3 njozefow  staff   96 Nov 15 07:47 .
drwxr-xr-x+ 113 njozefow  staff 3616 Nov 15 07:46 ..
drwxr-xr-x   9 njozefow  staff  288 Nov 15 07:47 .git
```

La commande crée un répertoire `.git`. Le répertoire `.git` contient l'historique des modifications apportées au dépôt.

Remarque : si le répertoire n'est pas vide, il faut d'abord ajouter les fichiers au dépôt avec `git add` (cf ci-dessous).

Modifier un dépôt GIT

Modification d'un dépôt git local

Une modification peut être :

- l'ajout d'un nouveau fichier
- la suppression d'un fichier
- renommer / déplacer un fichier
- la modification du contenu d'un fichier

Ces états sont établis par rapport au dernier commit.

Remarque : le terme *fichier* est utilisé pour désigner un **fichier** ou un **répertoire**.

Tester l'état d'un fichier

Tester l'état des fichiers

Pour voir l'état des fichiers, il faut exécuter la commande `git status`.

```
% git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Ajout d'un nouveau fichier

Ajout d'un nouveau fichier sous suivi de version

On crée un nouveau fichier dans le projet :

```
echo 'console.log("Hello, world!");' > main.ts
```

On peut voir que le fichier n'est pas suivi par git :

```
% git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  main.ts

nothing added to commit but untracked files present (use "git add" to track)
```

Le fichier est marqué comme non suivi (*untracked*). Cela signifie qu'il ne sera pas pris en compte lors des commits et que ses modifications ne sont pas suivies.

Pour ajouter le fichier à l'index, il faut exécuter la commande `git add`.

Ajout d'un nouveau fichier – part 2

```
% git add main.ts
% git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   main.ts
```

Suppression

Suppression d'un fichier

On peut supprimer un fichier. Par exemple, on peut supprimer le fichier `main.ts`.

```
% rm main.ts
% git status
On branch main
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    main.ts

no changes added to commit (use "git add" and/or "git commit -a")
```

On peut voir que le fichier `main.ts` a été supprimé mais n'est pas ajouté à l'index. Pour supprimer le fichier à l'index, il faut exécuter la commande `git rm`.

```
% git rm main.ts
rm 'main.ts'
% git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    main.ts
```

Remarque : on peut utiliser directement la commande `git rm` pour supprimer un fichier et l'ajouter à l'index.

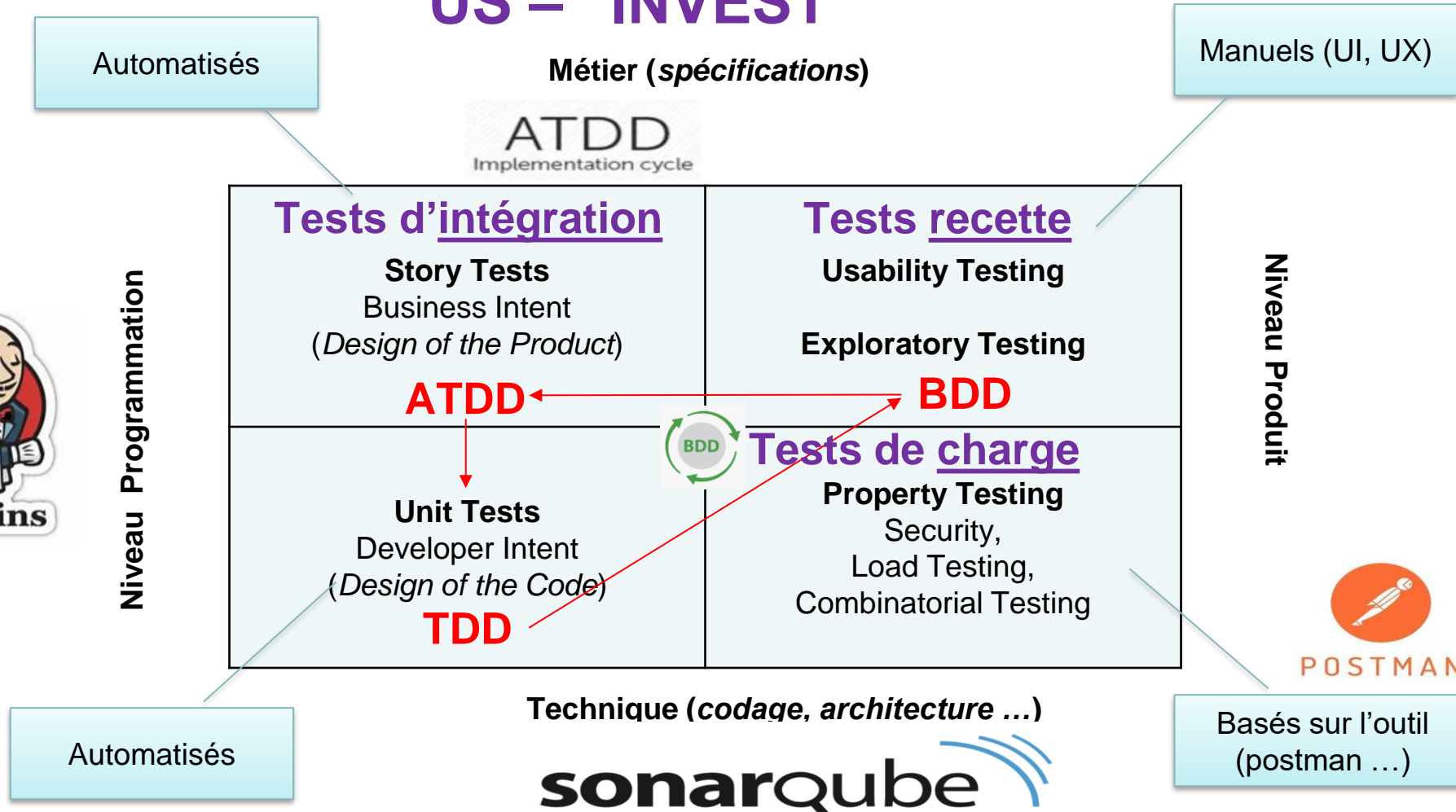
A man in a dark suit and white shirt stands against a background of a city skyline, with the Empire State Building prominent. A digital, pixelated cityscape is overlaid on the right side of his body. A dark horizontal bar contains the text.

4. Technique avancée par les tests.

Rappel des 4 grands types de tests

US – “INVEST”

Taches “SMART”



Du code “SOLID”

Principales pratiques

❑ Tests :

- Unitaires.
- Acceptances (recette) ...

❑ Intégration continue et « automatisation » :

- Serveur de 'build', repository (git)

❑ Design patterns (*Patron de conception*) :

- On ne « réinvente pas la roue » ... On utilise le bon pneu.

❑ Architecture ouverte :

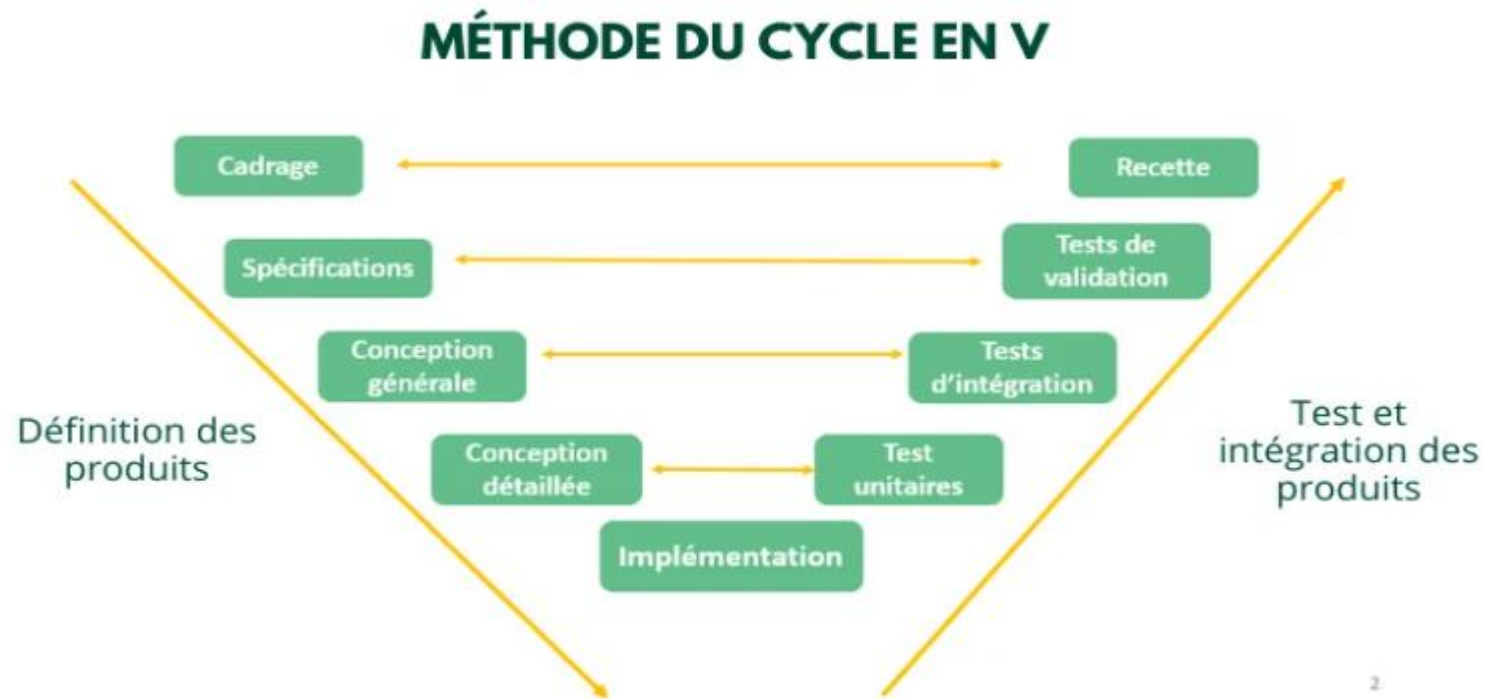
- Adaptabilité.

❑ Pair-programming et code review...

- 'Refactoring', amélioration continue ..

Méthode cycle en V

- ❑ Le cycle en V est une méthode **de gestion de projet** réputée pour son organisation des activités en deux flux parallèles :
- ✓ Un flux **descendant** : qui détaille le produit depuis le cadrage jusqu'à son implémentation en passant par le recueil des expressions et les spécifications.
- ✓ Et un flux **ascendant** : qui vérifie la qualité du produit à chacune des phases du projet en partant de l'implémentation pour remonter jusqu'au cadrage.



5. Automatisierung

Qu'est ce que l'automatisation des tests ?

- ❑ Les tests automatisés sont **une forme de test logiciel**.
- ✓ Ils consistent à utiliser des outils et des logiciels pour exécuter des cas de test et vérifier automatiquement les résultats par rapport aux attentes prédéfinies.
- ✓ Ceci dans le but de mettre en place **une stratégie de test efficace**.
- ❑ L'automatisation des tests suit un processus comprenant plusieurs étapes d'évaluation constante, détaillées comme suit :
 - Evaluation risques et des gains inhérents au processus d'automatisation des tests.
 - Evaluation de l'effort pour réaliser ce gain ou supprimer ce risque en déterminant les ressources, le temps, les coûts ...
 - Décision de réaliser ou non l'automatisation.
 - Evaluation des gains effectivement réalisés ou les risques qui demeurent malgré l'automatisation (*risques résiduels*).
 - Evaluation du ROI du cycle d'automatisation en comparant les gains obtenus à l'investissement initial.

Etapes de mise en place des tests automatisés

Etape 1 - Création des tests « unitaires » :


Les tests unitaires serviront de brique de base pour la mise en place des autres types de tests. Ce sont donc des « fonctions » qui seront appelées plus tard.

Etape 2 – Implémentation des tests de bout en bout :

Ces tests reprennent des étapes unitaires afin de construire des scénarios proches de ceux réalisés par un utilisateur réel.

Etape 3 - Implémentation des tests de performance :

Les tests de performance, parfois associés aux tests de charge, ont pour but de vérifier que dans ces cas d'utilisation normale ou intensive de l'application, celle-ci donne bien les résultats attendus dans des délais acceptables.

A man in a dark suit and white shirt stands against a background of a city skyline, with the Empire State Building prominent. A digital, pixelated cityscape is overlaid on the right side of the image, appearing to be part of the man's profile. A dark horizontal bar is positioned across the middle of the image, containing white text.

6. Remaniement de code et maintenance applicative.

Définition

Le remaniement de code est la méthode permettant de réécrire le code afin de le rendre plus lisible, plus maintenable, plus rapide ...

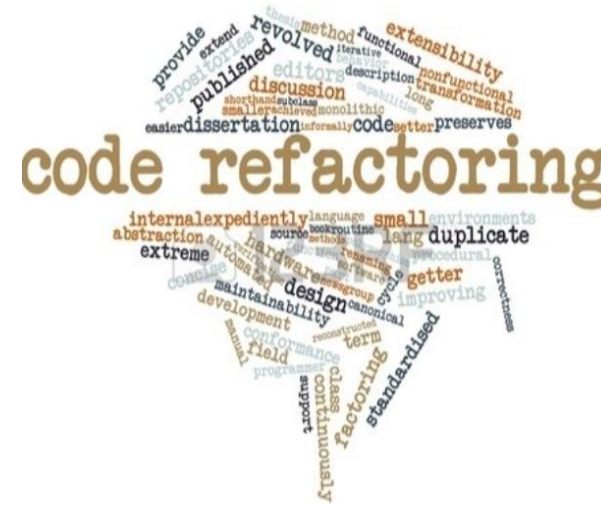
On parle également de REFACTORING.

❑ Le principe de base du refactoring est le suivant :

- Ajouter de nouvelles fonctions uniquement si et seulement si **celles-ci ne modifient** pas le comportement initial du code source.
- On peut apporter des modifications au code source (*Refactoring*) uniquement si et seulement si **celles-ci ne créent** pas de nouvelle fonction.
- Refactorer sans tests unitaires TU qui garantissent la **non-regréssion** est une mission « *quasi-impossible* ».

Pourquoi on refactor ?

- ☐ Apporter de la clarté / lisibilité au code.
- ☐ Eviter le code « *spaghetti* ».
- ☐ Faciliter sa relecture et sa compréhension.
- ☐ Faciliter le partage de connaissance.
- ☐ Faciliter son débogage et l'ajout de code.
- ☐ Nettoyer le code « *sale / dirty* » ou code « *mort / dead* ».
- ☐ Enlever les codes en « doublons ».
- ☐ Apporter de la rapidité d'exécution.



Simplifier l'analyse des erreurs et la maintenabilité du code.

Exemple de refactoring Java

```
@Override  
public int multiplication(int a, int b) {  
    return a * b;  
}
```

Refactoring de la
méthode
« *multiplication* »
qui réalise une
multiplication en
utilisant des
« *additions* ».

```
@Override  
public int multiplication(int a, int b) {  
    int resultat = 0;  
    if (a == 0 || b == 0) {  
        return resultat;  
    }  
    boolean isNegatif = (b < 0);  
    for (int i = 1; i <= abs(b); i++) {  
        resultat = addition(resultat, a);  
    }  
    if (isNegatif) {  
        return resultat * -1;  
    }  
    return resultat;  
}
```

A man in a dark suit and light shirt is smiling and looking to his right. The right side of his body is merged with a digital, pixelated overlay of a city skyline, including the Empire State Building. The background is a hazy, high-angle view of a city.

A vous de jouer 😊