

# Initiation au Génie Logiciel

## Cours 6

### Introduction au « développement agile »

## Plan

- 1 Cycles de développement
- 2 Méthodes agiles, principes généraux
- 3 Comment se passe un Sprint ?
- 4 Principes de développement agiles : TDD, YAGNI, KISS

## Bibliographie

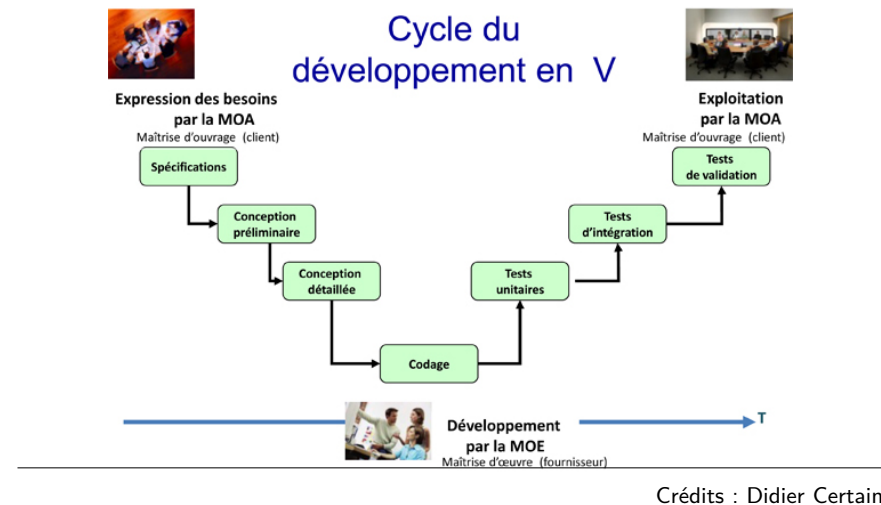
- *L'informatique agile, c'est quoi ?*. D. Certain. Interstices. 2013.  
[https://interstices.info/jcms/int\\_71726/linformatique-agile-cest-quoi](https://interstices.info/jcms/int_71726/linformatique-agile-cest-quoi)
- *XP : Extreme programming*. Wikipédia.
- *TDD : Test Driven Development*. Wikipédia.

## Plan

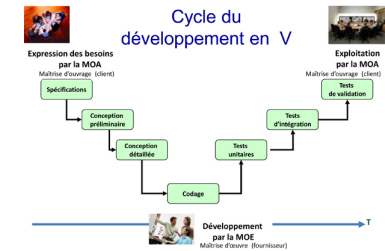
- 1 Cycles de développement
- 2 Méthodes agiles, principes généraux
- 3 Comment se passe un Sprint ?
- 4 Principes de développement agiles : TDD, YAGNI, KISS

## Cycles de développement : le cycle en V

80% des développements logiciels sont organisés selon un cycle en V

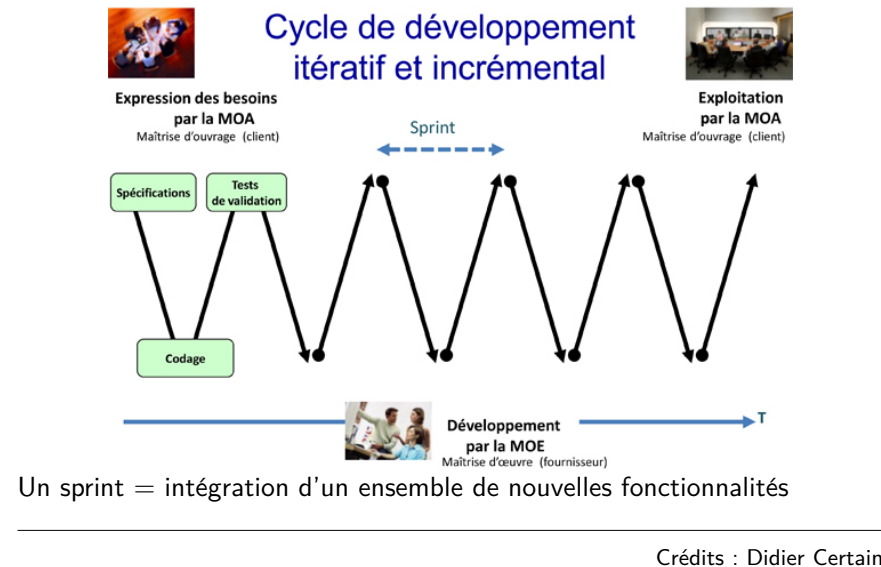


## Cycles de développement : le cycle en V (II)

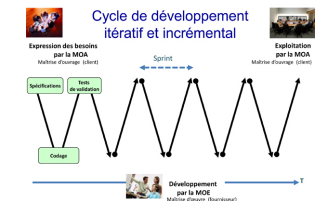


- + Découpage du développement rationnel, logique, stable
- + Identification claire des documents à produire, des rôles des équipes
- + Convient aux grosses équipes, développement de logiciels "critiques"
- Manque de souplesse : le client ne peut pas changer ses objectifs en cours de route !
- Découverte tardive des erreurs de spéc./conception par le client
- Intégration finale risquée : elle s'effectue à la fin du cycle (1 à 2 ans)

## Cycles de développement : le cycle itératif



## Cycles de développement : le cycle itératif (II)



- + Intégration continue : dispose en *permanence* d'un logiciel fonctionnel (cycles courts : 2 semaines)
- + Validation rapide et fréquente du logiciel par le client (au moins à chaque fin de sprint)
- + Le client peut changer ses objectifs (pas de spécification initiale figée)
- Inadapté au développement de logiciels critiques (besoin de spécifications détaillées et stables)
- Mise en place complexe pour les grosses équipes (> 12 p.) et équipes multi-localisées

## Plan

- 1 Cycles de développement
- 2 Méthodes agiles, principes généraux
- 3 Comment se passe un Sprint ?
- 4 Principes de développement agiles : TDD, YAGNI, KISS

## Méthodes agiles : une implémentation du cycle itératif

### Manifeste agile

« Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire. Ces expériences nous ont amenés à valoriser :

- Les individus et leurs interactions plus que les processus et les outils
- Des logiciels opérationnels plus qu'une documentation exhaustive
- La collaboration avec les clients plus que la négociation contractuelle
- L'adaptation au changement plus que le suivi d'un plan

Nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers. »

## Méthodes "agiles" : le bestiaire

- RAD, ASD, FDD, Kanban, Lean, ...
- **Scrum**
  - ▶ User Stories
  - ▶ Sprint
  - ▶ Sprint planning
  - ▶ Daily Scrum
  - ▶ Démo de fin de Sprint
- **XP : Extreme Programming**
  - ▶ Intégration continue
  - ▶ TDD : Test Driven Development
  - ▶ Pair programming
  - ▶ Principes YAGNI et KISS

Nous allons piocher des éléments dans **Scrum** et **XP** pour organiser le travail en TP

## Préliminaire au développement agile : les user-stories

### Définition 1 (User-story)

Une user-story (récit utilisateur) est une phrase simple décrivant **une** fonctionnalité du logiciel à développer. **Idéalement**, la phrase contient trois éléments descriptifs : **qui ?**, **quoi ?** et **pourquoi ?**.

En tant que <qui>, je veux <quoi> afin de <pourquoi>

### Exemple 2 (Une User-story pour le robotWeb)

En tant qu'utilisateur, je veux pouvoir taper un mot clé afin d'obtenir une liste de pages web contenant ce mot clé.

## Le développement agile commence : la version 0 du logiciel

- Les user-stories vont être intégrées au fur et à mesure dans le logiciel
- La version 0 (V0) est quasiment une coquille vide : elle doit être rapidement opérationnelle
- **Idéalement**, la V0 doit faire *intervenir* tous les composants du logiciel final : périphériques, interf. graphiques, syst. de fichiers, réseau, ...

### Exemple 3 (Version 0 du logiciel pour le robot web)

- Lit une chaîne de caractère au clavier
- Lit une URL sur internet
- Écrit un contenu quelconque dans un fichier

### Remarque 1

*Si la V0 fait bien intervenir tous les éléments de la chaîne, on pourra détecter très tôt tout problème technique majeur : par exemple des incompatibilités matérielles ou logicielles.*

## Le point de départ : la version 0 du logiciel (II)

### Exercice 1

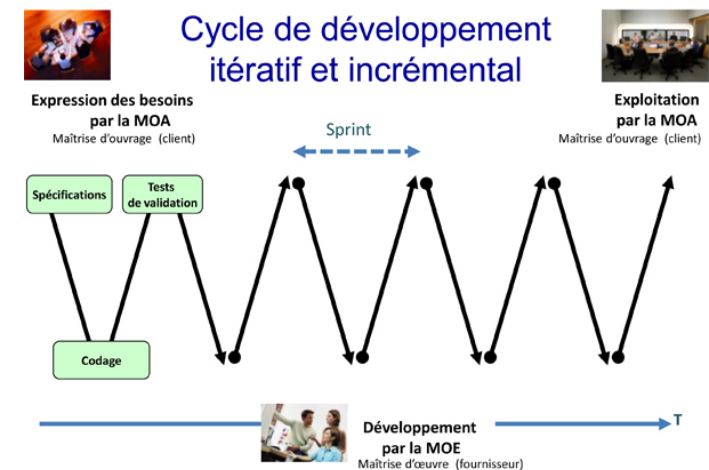
Quelle seraient de bonnes V0 pour les logiciels suivants :

1. Navigateur web
2. Eclipse
3. SVN

## Plan

1. Cycles de développement
2. Méthodes agiles, principes généraux
3. Comment se passe un Sprint ?
4. Principes de développement agiles : TDD, YAGNI, KISS

## Cycles de développement : le cycle itératif



Un **sprint** = intégration d'un ensemble de nouvelles fonctionnalités

## Comment se passe un Sprint ?

- \_\_\_\_\_ Client + Développeurs \_\_\_\_\_
- 1 Le client définit la liste de user-stories à intégrer pendant le Sprint
- \_\_\_\_\_ Développeurs \_\_\_\_\_
- 2 Affecter les user-stories à des équipes (Scrum Planning)
  - 3 **Développer, intégrer, tester le code du Sprint**
  - 4 Si nécessaire, pendant le Sprint, les équipes se réorganisent
- \_\_\_\_\_ Client + Développeurs \_\_\_\_\_
- 5 A la fin du Sprint : démo avec le client qui vérifie que les user-stories sont couvertes par la version courante du logiciel

## Comment développer du code pendant un Sprint

- Méthode de dével. du code : **TDD + KISS + YAGNI**
- Méthode de validation du code : Intégration continue (SVN + JUnit)
  - ▶ Le code correspondant à la "user-story" est intégré à la version courante du logiciel
  - ▶ On teste que la modification ne fait pas régresser le logiciel !
  - ▶ On teste que la modification satisfait bien la "user-story"
- Méthode de travail Pair programming (programmation en binômes)
  - ▶ Le pilote a le clavier. C'est lui qui code.
  - ▶ Le co-pilote, l'aide en suggérant de nouvelles solutions ou en décelant d'éventuels problèmes
  - ▶ Ils **échangent leurs rôles régulièrement**
  - ▶ Idéalement, les binômes **changent d'une séance à l'autre** pour améliorer la communication et la connaissance collective de l'application

## Plan

- 1 Cycles de développement
- 2 Méthodes agiles, principes généraux
- 3 Comment se passe un Sprint ?
- 4 **Principes de développement agiles : TDD, YAGNI, KISS**

## Principe KISS (Keep It Simple, Stupid)

Eviter toute complexité non nécessaire

- Rechercher la simplicité dans la conception
- Simplifier le code de la version courante si nécessaire (Refactoring)

### Exemple 4 (On veut représenter un ensemble de villes)

Pas KISS	<pre>class Ville(n:String){   var nom=n } class MutableVilleSet {   def ajouterVille(v:Ville):Unit={ ... } } val s= new MutableVilleSet s.ajouterVille(new Ville("Rennes")) s.ajouterVille(new Ville("Milan"))</pre>
KISS	<pre>case class Ville(n:String) val s= Set(Ville("Rennes"),Ville("Milan"))</pre>

## Principe YAGNI (You Ain't Gonna Need It)

Ne pas prévoir ce qui, de toute façon, ne sera jamais utilisé par la suite!

### Exemple 5 (On veut représenter des utilisateurs UNIX)

Pas YAGNI	<pre>case class Utilisateur(n:String,p:String){   // génère automatiquement le nom de login   // de 8 caractères à partir du nom et de la   // première lettre du prénom   var login= n.subSequence(0,7)+              p.subSequence(0,1) }</pre>
YAGNI	<pre>case class Utilisateur(login:String)</pre>

- Raccourcit le temps de développement
- Evite l'alourdissement de l'architecture
- Elimine des bugs potentiels... dans du code inutile!

## TDD : Test Driven Development

Pour ajouter une fonctionnalité X à un logiciel :

- 1 Ecrire les tests T montrant que X fonctionne
- 2 Vérifier que le test échoue (X n'est pas encore développée)  
⇒ Permet de vérifier que le test est valide!
- 3 Ecrire le code nécessaire pour passer T (et pas plus)
- 4 Vérifier que les tests T passent, sinon retourner en 3
- 5 Si nécessaire, remanier le code pour le simplifier (KISS)

Quiz 1 (f existe, on ajoute un test pour vérifier qu'elle est commutative)

```
@Test
def test1{
  assertEquals(f(11,1),f(11,1))
}
def f(x:Int,y:Int)= x
```

Quelle est l'étape du TDD qui a été négligée  1  2  3  4

## TDD : Test Driven Development (II)

### Exemple 6

On développe un logiciel dans lequel on peut cliquer sur un bouton pour modifier une liste d'éléments.

- L'équipe A développe l'interface graphique
- L'équipe B développe le traitement des listes
- L'équipe C développe l'affichage graphique des listes

La première "user-story" à traiter est : « L'utilisateur clique sur un bouton, l'application supprime 2 dans la liste [1, 2, 3] et affiche le résultat. »

## TDD : un exemple de code de l'équipe B

(1) On écrit le test

```
import org.junit.Assert._
import org.junit.Test
import ListTools._

class TestListTools {

  // delete supprime un élément
  // dans une liste
  @Test
  def test_supplement{
    assertEquals(
      List(1,3),
      delete(2,List(1,2,3)))
  }
}
```

```
object ListTools {
  def delete(x:Int,l:List[Int])=
    ???
}
```

(2) on vérifie que le test échoue!

## TDD : un exemple de code de l'équipe B

<pre>import org.junit.Assert._ import org.junit.Test import ListTools._  class TestListTools {    @Test   def test_supplement{     assertEquals(       List(1,3),       delete(2,List(1,2,3)))   } }</pre>	<p>(3) On écrit le code passant le test</p> <pre>object ListTools{   def delete(x:Int,l:List[Int])=     List(1,3) } // KISS + YAGNI :-)</pre> <p>(4) on vérifie que le test passe ... et c'est tout, pour ce test !</p>
--	---

B peut déjà envoyer son code qui valide la première "user-story" !!

⇒ L'avancement de A et C n'est pas bloqué par l'attente du code de B

## TDD : l'exemple, seconde "user-story"

### Exemple 7 (B étudie la seconde "user-story" à intégrer)

L'utilisateur clique sur un bouton cela supprime 1 dans la liste [1, 2, 1, 1].

<p>(1) On écrit le test</p> <pre>[...] @Test def test_supplement{   assertEquals(     List(1,3),     delete(2,List(1,2,3))) }  @Test def test_suppToutes_occ{   assertEquals(     List(2),     delete(1,List(1,2,1,1))) }</pre>	<pre>object ListTools {   def delete(x:Int,l:List[Int])=     List(2,3) }</pre> <p>(2) on vérifie que le test échoue !</p>
---	---

## TDD : l'exemple, seconde "user-story"

<pre>[...] @Test def test_supplement{   assertEquals(     List(1,3),     delete(2,List(1,2,3))) }  @Test def test_suppToutes_occ{   assertEquals(     List(2),     delete(1,List(1,2,1,1))) }</pre>	<p>(3) On écrit le code passant le test</p> <pre>object ListTools{   def delete(x:Int,l:List[Int])=     if (l==List(1,2,3))       List(1,3)     else List(2) } // Pas KISS :-)</pre> <p>(4) on vérifie que le test passe (5) On remanie le code</p> <pre>object ListTools{   def delete(x:Int,l:List[Int])=     l.filter(_!=x) } // KISS :-)</pre>
---	--

## Déroulement des séances de TPs : Daily Scrum

Chaque séance de TP (en mode projet) commencera par : un Daily Scrum

### Définition 8 (Daily Scrum – Méléé journalière)

C'est une réunion courte (max 10 minutes) où chaque membre du groupe dit, à son tour :

- 1 Ce qu'il a fait à la dernière séance
- 2 Ce qu'il doit faire à cette séance
- 3 Les problèmes qu'il a pour atteindre cet objectif

Quelques règles :

- Tout le monde est debout
- Le but est de signaler les problèmes, pas de les résoudre  
Ils seront résolus plus tard, après le daily scrum.