Module Conception d'Architectures Logicielles

Chapitre 1 – Présentation du Module, Pourquoi la Conception, Cas spécifiques

Véronique DESLANDRES - Licence professionnelle DEVOPS, IUT LYON1

Présentation du module

- 28h, une évaluation finale, du Contrôle Continu
- **Principes SOLID et Design Patterns**: 18h
  - SOLID = principes d'une Conception de Qualité
  - Design Patterns : introduction, implémentation de patterns
- **Qualité / Génie Logiciel / Clean Code** : 4h
  - SonarQube, GitLab ?
  - Cours / TD
- **Patterns d'Architecture** : 6h
- **Supports / planning des séances :**

https://perso.liris.cnrs.fr/vdesland/doku.php?id=start:ens:uml_dasi

L'évaluation

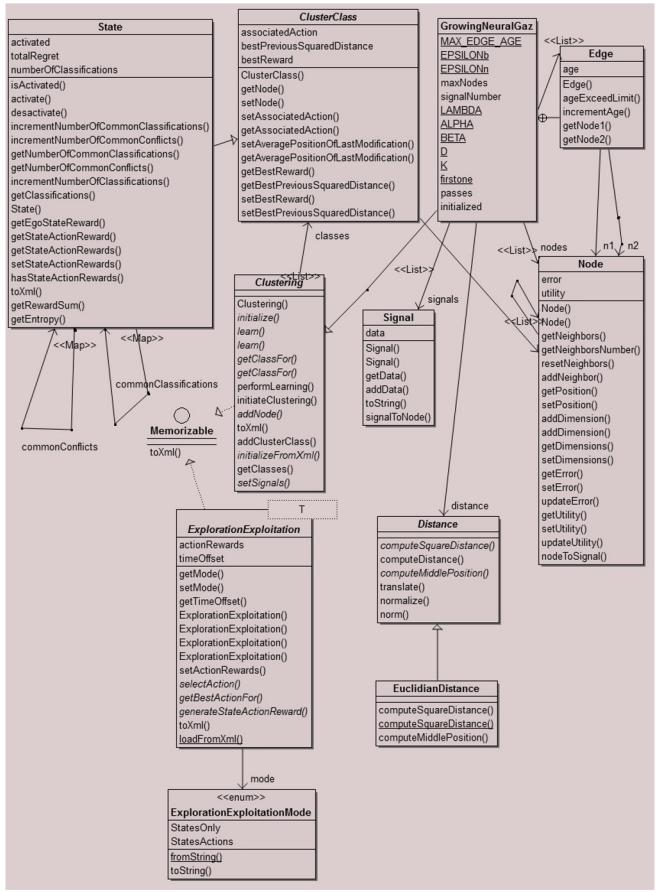
- **Examen final**
  - 60% de la note
  - Documents autorisés : corrigés de TD
- **Contrôle continu**
  - 40% de la note
  - Un QROC
  - Participation en cours
    - Bonus/Malus 2 pts

POSITIONNEMENT DU MODULE

- *Pré requis :* savoir utiliser UML (Diagramme de Classes) pour modéliser
  - Cf rappels si nécessaires sur mon site

- Compétences en Conception de niveau Bac+3
  - Concevoir **l'architecture** logicielle
  - **Découper en packages** à forte cohésion, faiblement couplés

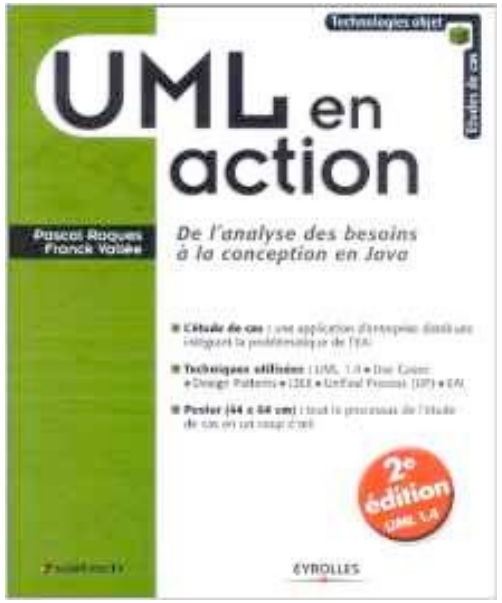
- Compétences vues dans **ce module** et en **Archi nTiers**
- Mises en pratique en .NET et JEE



Bibliographie UML

BU IUT Lyon !

- Pour **débuter** :
  - Introduction à UML, de Sinan Si Alhir, traduit par Alexandre Gachet, 221 p., 1ère édition, sept. 2005, Editions O'Reilly.
- **Référence** : mise en pratique d'UML dans des projets réels
  - UML en Action, Pascal Roques, Franck Vallée, Ed. Eyrolles
  - Modélisation Objet avec UML, Muller, Gaertner, Eyrolles
- Comprendre **UML sur des applications web** :
  - UML2 - Modéliser une application web, Pascal Roques, 2008, Editions Eyrolles



Vous avez dit Pédagogie ?

- Votre travail :
  - **Ecouter** les cours en séance, **poser** des questions
    - Il n'y a pas de question bête
  - Transmettre en cours les points qui restent flous ou mal compris

- Mon travail :
  - Vous rendre **acteurs** de votre apprentissage
  - Répondre à **vos** besoins d'apprentissage

Supports

- Mes supports sont des **présentations de cours**, en séance, assez volumineux car faits pour l'interaction…
  - Mis à disposition sur ma page web du labo :

    https://perso.liris.cnrs.fr/vdesland/doku.php?id=start:ens:uml_dasi

    Ou chercher : DESLANDRES LIRIS → page Enseignement

  - Autres références fournies en fin de ce cours

Présentation réciproque

- **V. Deslandres**
  - Enseignante-chercheure à l'IUT de Lyon depuis 2003
  - 5 ans chez Renault - Direction de la Recherche
  - Laboratoire LIRIS, équipe IA (modélisation multi-agents)

Tour de salle

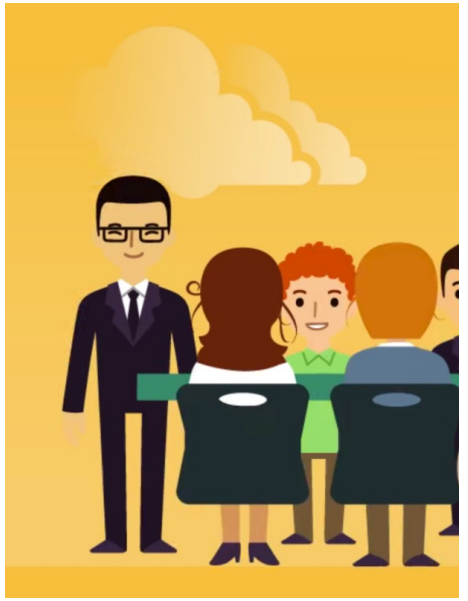
- Quelle formation antérieure ?
- Missions de Conception / Développement ?

Pourquoi bien Concevoir ?

En quoi c'est important pour le logiciel

Rappel : les parties prenantes du développement logiciel

- **Utilisateurs**
  - Ceux qui se servent du logiciel
- **Clients**
  - Ceux qui paient pour le logiciel
- **Développeurs** (les *Dev*)
  - Ceux qui conçoivent et implémentent le logiciel
- **Exploitants** (la Prod, les *Ops*)
  - Ceux qui exploitent le logiciel (mise en production, maintenance, évolution)
- **Managers**
  - Ceux qui supervisent la production du logiciel



Analyse & Conception : 2 phases distinctes

Conceptual Design

ANALYSE
Conception fonctionnelle

Technical Design

CONCEPTION technique

indique au client ce que le système va faire

client et les utilisateurs finaux

permet aux développeurs de savoir comment implémenter une solution conforme aux attentes du client

Analystes

Conception fonctionnelle vs. Technique

- **Conception fonctionnelle ou Analyse** : « dissection » du problème

QUOI

  - Étude du domaine du problème
  - Définition d'un comportement observable de l'extérieur
  - Langage compréhensible par le client
  - Ex. dans l'apps Blablacar : conducteur, passager, trajet, réservation, etc.

- **Conception technique** : fabrication d'une solution informatique

COMMENT

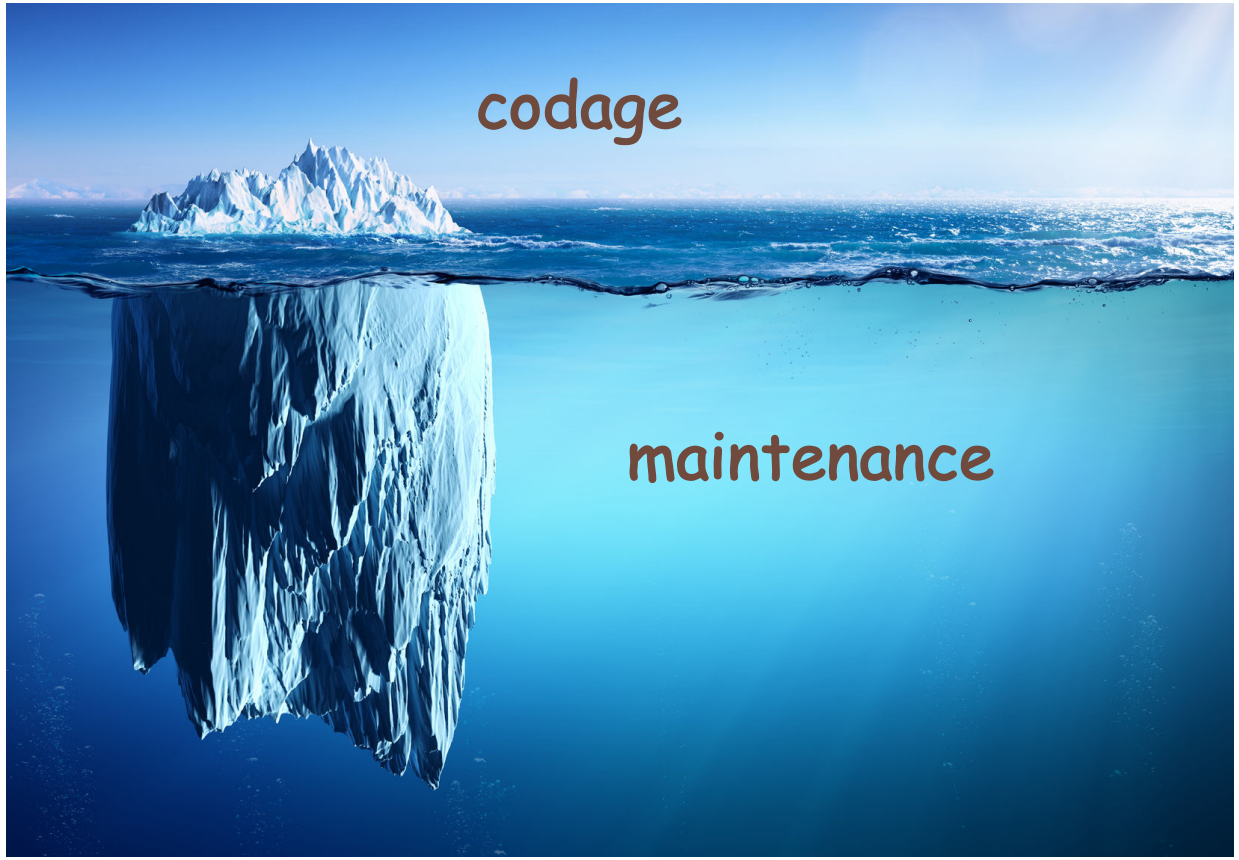
  - Détails d'implémentation réelles (IHM, stockage, accès, sécurité, fréquence/débits, performances)
  - Premiers livrables de code
  - Ex.: Découper le trajet en tronçons ; lister les trajets demandés en moins de 2 sec. ; identifier un Conducteur par son nom/prénom/dateNaissance obtenu de son permis ; archiver les entités sans interaction avec l'apps depuis plus de 3 ans

Pourquoi bien concevoir, est-ce important ?

Concevoir un logiciel non pas pour son utilisation IMMEDIATE, mais qui sera MODIFIÉ, qui pourra ETRE ÉTENDU.

Conception = définir un logiciel facile à faire évoluer

Formaliser les choses de façon générique. Toujours anticiper leur possible évolution.

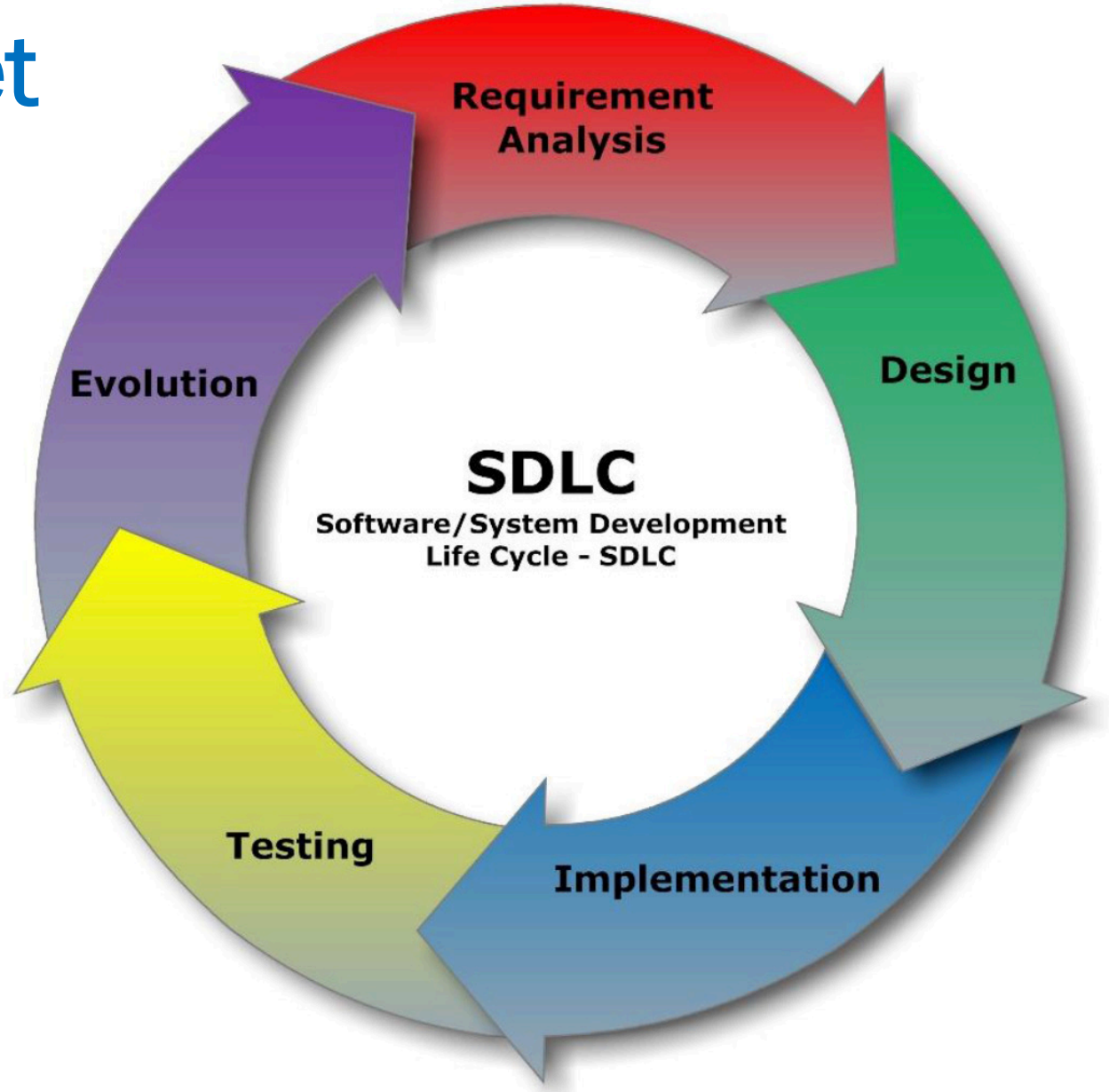


Cycle de vie d'un projet

Durée de vie moyenne : 7 ans

À votre avis ?
Quel pourcentage du temps représente la création d'un logiciel ? (test compris)

Celui consacré à sa maintenance et son évolution ?



Importance des tests de régression

Les tests de régression sont **obligatoires** si votre code :

- Doit être utilisé longtemps ;
- Sera utilisé dans des environnements différents : compilateur, processeur, version de langage, langue de l'utilisateur… ;
- Risque d'évoluer.

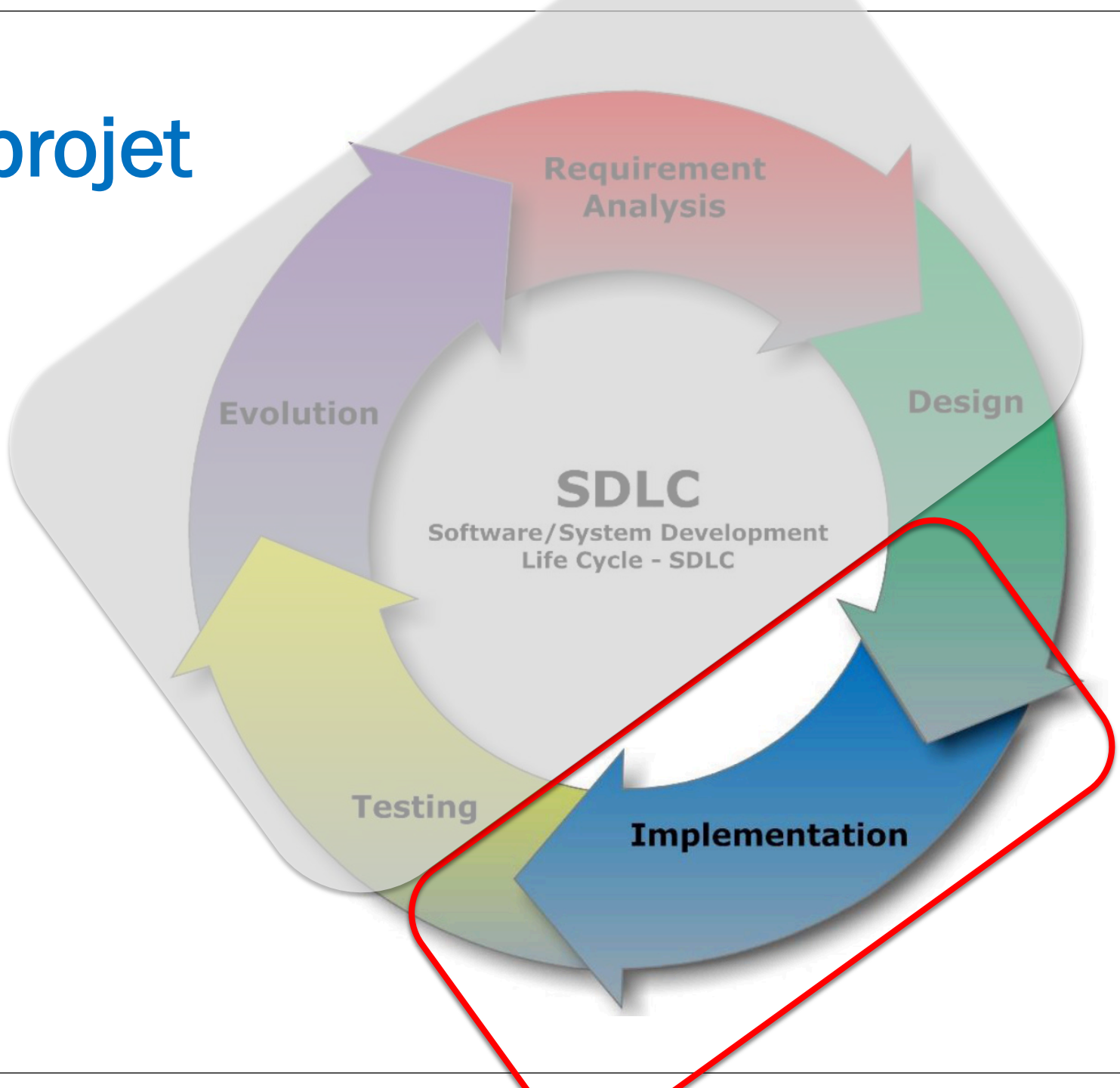
CONSEILS

- Faire des tests **courts en temps d'exécution** ;
- **Indépendants** les uns des autres, et donc **exécutables dans n'importe quel ordre** ;
- Vérifiant chacun le **minimum** de choses ;
- Documentant ce qu'ils font et pourquoi c'est le résultat attendu ;
- Stockés chacun **dans un fichier séparé** afin de pouvoir facilement utiliser *git bisect** pour trouver automatiquement quand un bug a été introduit.

**https://runebook.dev/fr/docs/git/git-bisect-lk2009*

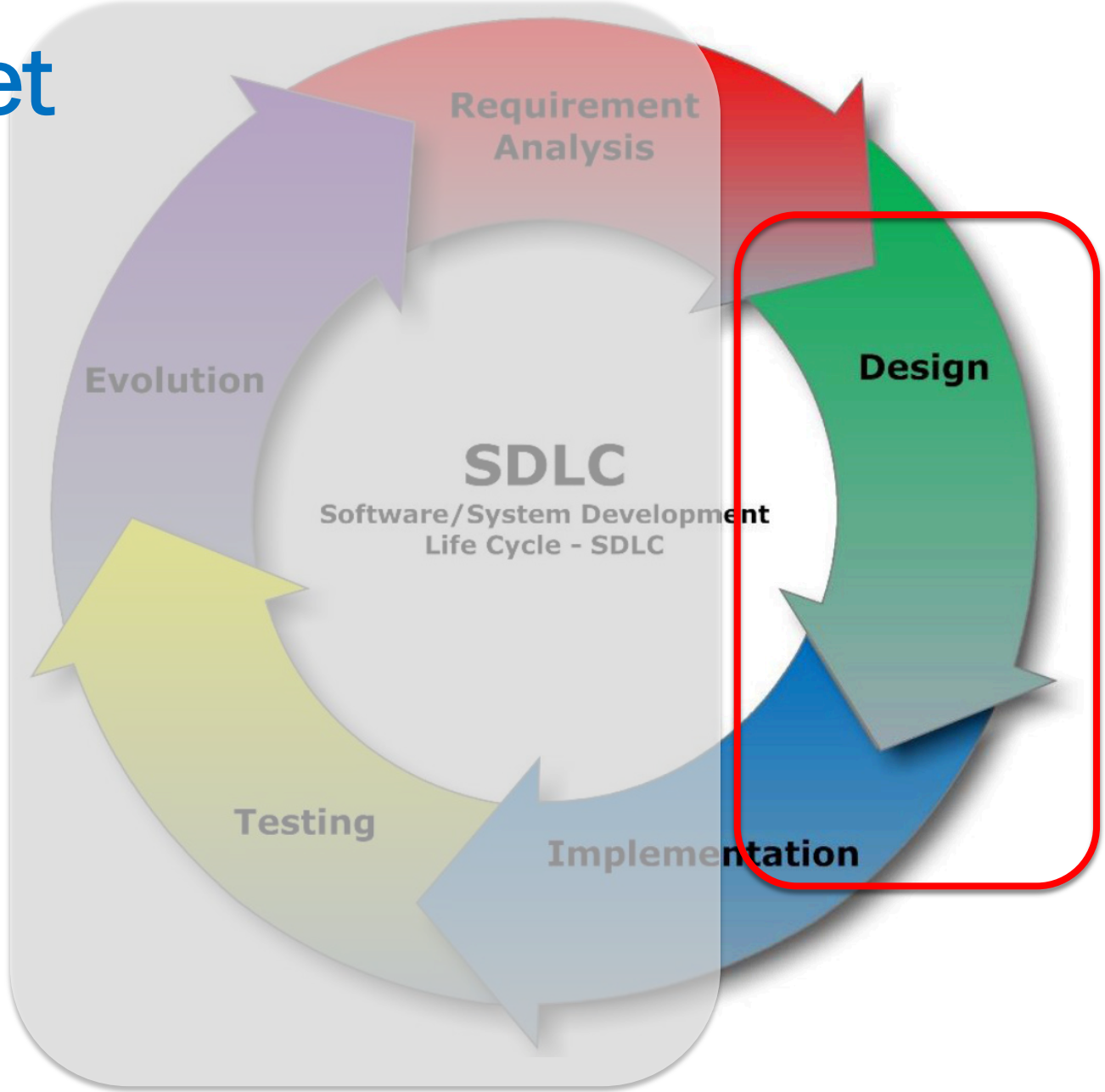
Cycle de vie d'un projet

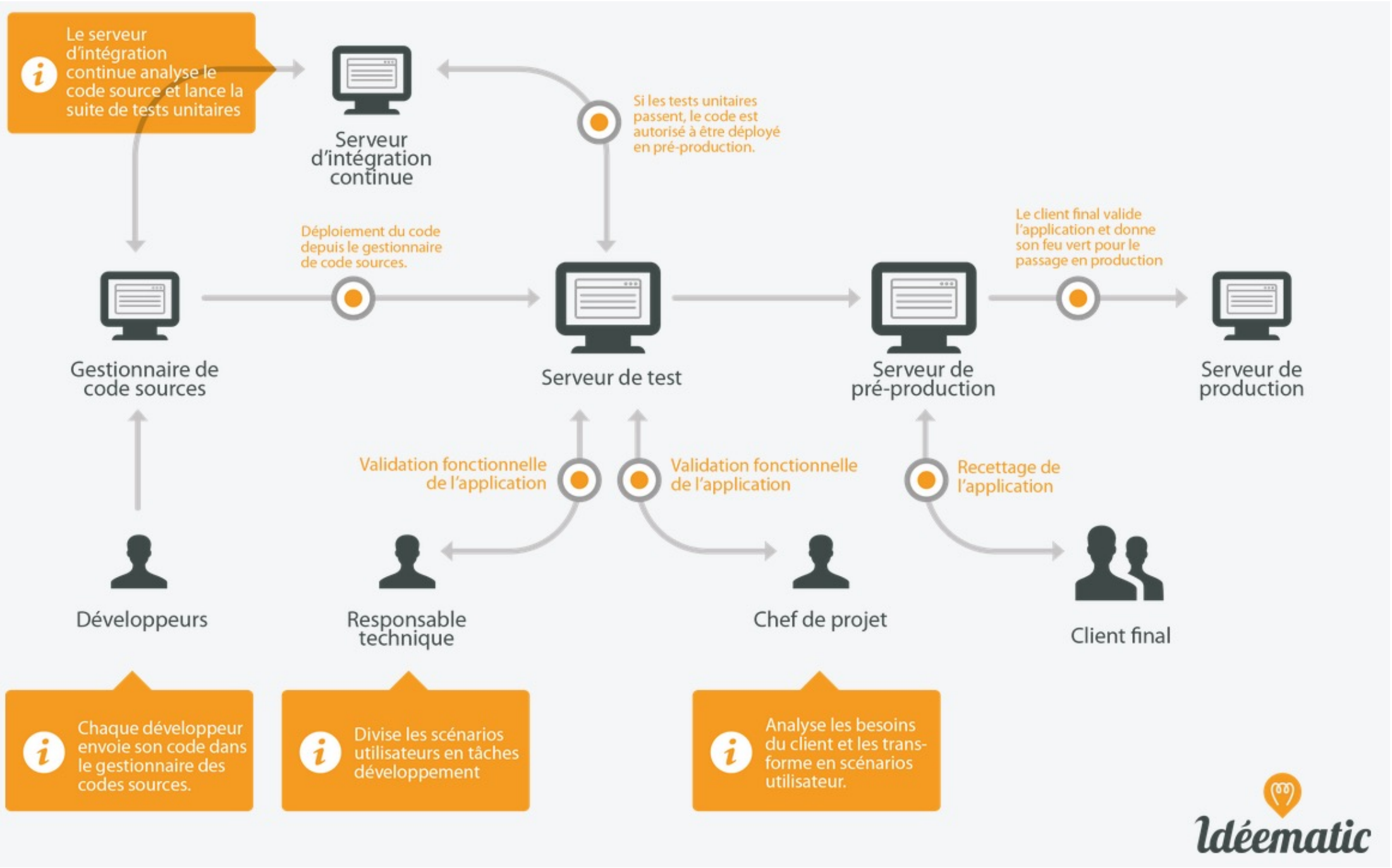
Partie vue à l'IUT



Cycle de vie d'un projet

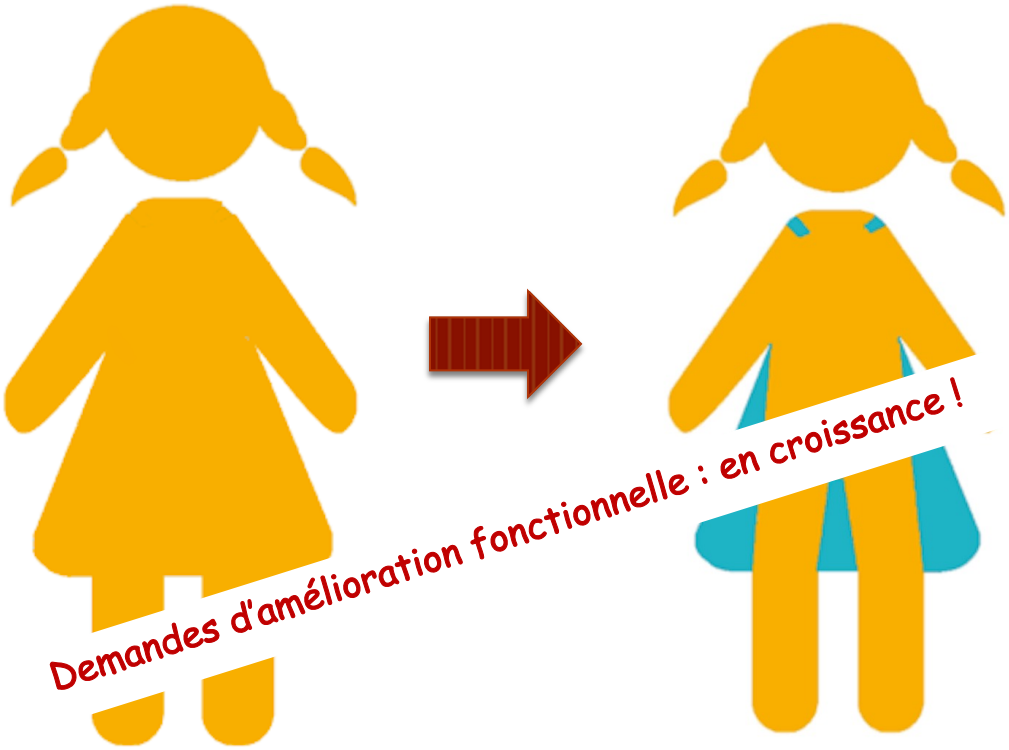
Partie concernée par ce module




Le serveur d'intégration continue analyse le code source et lance la suite de tests unitaires
Serveur d'intégration continue
Si les tests unitaires passent, le code est autorisé à être déployé en pré-production.
Déploiement du code depuis le gestionnaire de code sources.
Le client final valide l'application et donne son feu vert pour le passage en production
Gestionnaire de code sources
Serveur de test
Serveur de pré-production
Serveur de production
Validation fonctionnelle de l'application
Validation fonctionnelle de l'application
Recettage de l'application
Développeurs
Responsable technique
Chef de projet
Client final
Chaque développeur envoie son code dans le gestionnaire des codes sources.
Divise les scénarios utilisateurs en tâches développement
Analyse les besoins du client et les transforme en scénarios utilisateur.
Idéematic

Evolution : 2 types

- Evolutions fonctionnelles



- Evolutions technologiques

Les plus fréquentes

Ex de besoins technologiques : déploiement sur différents OS, migration d'OS

78% des répondants à leur enquêtes utilisent de 1 à 4 OS différents

(Enquête PUPPET 2018, 300(professionnels de l'IT, enquête annuelle internationale)

# of OSes Selected	# of Responses	% of Total
0	25	1%
1	1,035	22%
2	1,038	23%
3	883	19%
4	636	14%
5	420	9%
6	234	5%
7	139	3%
8	86	2%

78%

78% of respondents are widely deployed on 1-4 different operating systems.

Ex. de fonctionnalités techniques

- Mise à jour automatique du DNS
- Empêcher la duplication des noms de déploiement
- Échec rapide si opération non autorisée
- Mise à jour automatique des variables d'environnement
- Retour à la version précédente (rollback) manuelle
- Auto-rollback en cas d'échec d'une nouvelle livraison
- Améliorer les performances de copie de BD
- Etc.

21

Structures de données après *n* modifications, extensions et évolutions technologiques…

Découper son logiciel : idées générales

- *Segmenter* son application en **modules le plus indépendants possibles**
  - En termes de code, de fonctionnalités, pas forcément en termes de données

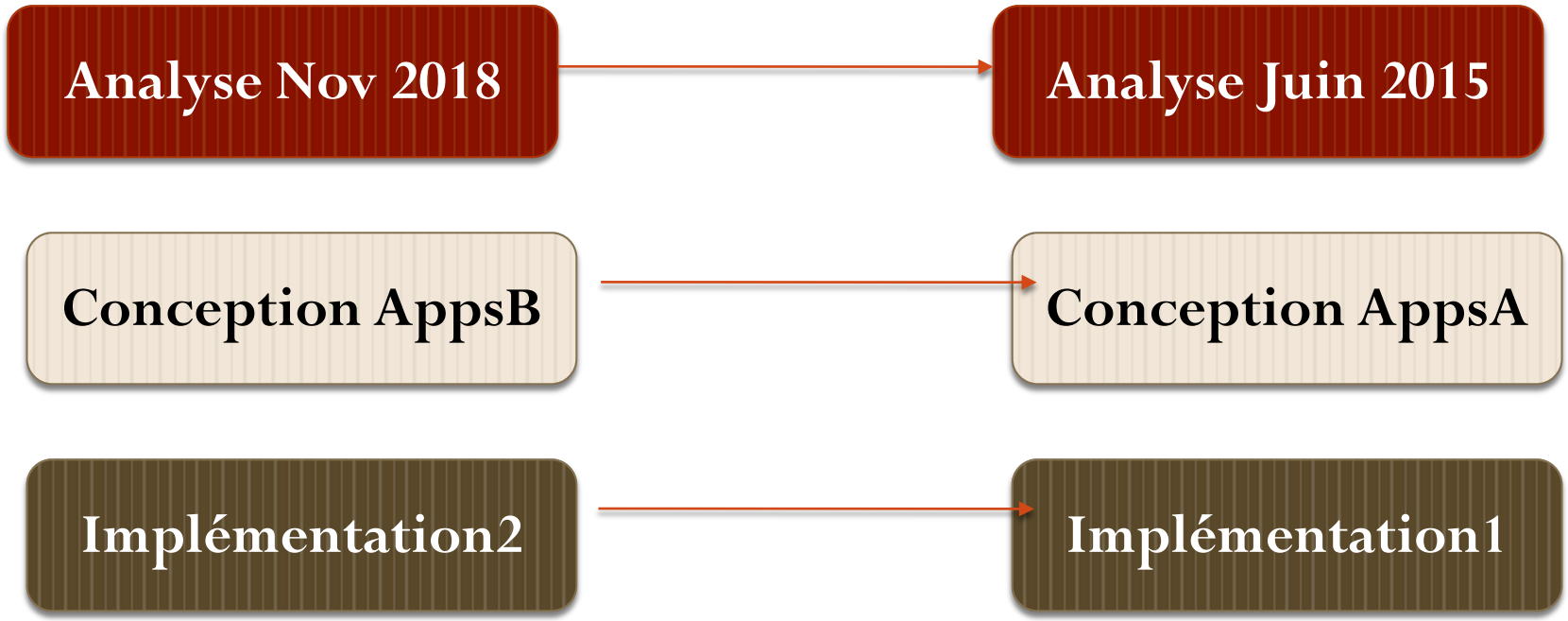
- On réduit les problèmes pour arriver à des composants **plus locaux, plus ciblés**
- Les classes qui changent **au même rythme** seront placées dans un même module.

- *Segmenter* diminue aussi le **périmètre** des tests
  - Modifier des lignes de code → génération de bugs
  - En cas d'évolution, comme on a découpé une appli en gros modules, on ne vérifie / teste que le module où a eu lieu la modification
  - (Et éventuellement toutes les applis Clientes qui utilisent ce module)

- Parvenir à **séparer son application en modules** n'est pas une chose facile
  - Les diagrammes des cas d'utilisations constituent une bonne approche

Autre objectif d'une conception générique : la réutilisation

- On peut « réutiliser » à différents niveaux et de différentes sources :



- **A chaque nouvelle expérience, on peut réutiliser.**
- **Pas d'outil spécifique : capitalisation de l'expérience**

Coder avec les principes LIM, KISS, DRY

Ce sont des objectifs fondamentaux à garder en mémoire quand on code :

- **LIM – Less is more :** moins il y a de code, le mieux c'est.
  - Code minimaliste ➔ meilleure factorisation ➔ meilleure évolutivité
- **KISS – Keep it simple and stupid** : le plus simple est le mieux.
  - Code simple ➔ facile à appréhender ➔ facile à maintenir dans le temps
- **DRY – Dont Repeat Yourself**
  - Dupliquer le code est source de problèmes à l'avenir (modification partielle, oubli)

Comment choisir : code Procédural ou Objet ?

- Robert Martin : « Le tout objet est un mythe »
- Constat : pour la maintenance de code (ajout de fonctions, de données), ce qui est facile pour l'Objet est complexe pour le procédural (structures de données et procédures), et réciproquement.

- Ainsi : si on a besoin de souplesse pour l'ajout de **nouveaux types de données** : choisir une **implémentation fondée sur les objets**
  - C'est facile d'en ajouter sans modifier ce qui existe
- Si on a besoin de souplesse pour l'ajout de **nouveaux types de comportements / fonctions** : choisir une **implémentation avec du procédural**

Conception de logiciels concurrents

- La conception d'un système concurrent peut être très différente de celle d'un système *monothread*. Le découplage du **quoi** et du **quand** a généralement un impact très important sur la structure du système.

- Garder le code lié à la concurrence séparé de tout autre code.

- Ne pas chasser à la fois les bogues du code normal et ceux du code *multithread*. Vérifier que le code fonctionne en dehors des threads.

Conception de logiciels embarqués

- **Caractéristiques** : temps-réel, concurrence et ressources limitées en charge de calcul et en mémoire.
- Ce type de logiciel est souvent couplé à un HardWare (HW) et un OperatingSystem (OS) spécifiques. Parfois même, le HW et l'OS sont développés en parallèle du logiciel et ne sont donc disponibles que tard dans le projet.
  - Avec donc une intégration tardive avec le HW et l'OS, tests difficiles.

- Les projets embarqués ont plutôt la réputation d'être marqués par les processus, les outils, la documentation et la recherche de la **conformité aux plans**. Bref, bien loin du développement logiciel agile.

Conception de logiciels embarqués (suite)

- **Séparer les problèmes** : le pilotage par les tests (TDD) va permettre de découpler le code. Càd. isoler et réduire les dépendances vers le HW et l'OS.

1. Jouer des suites de tests sur une **machine de développement** (et non sur la cible) avec des mocks et stubs (simulations des E/S) du HW et de l'OS

   Ici on teste le **code métier** : les algorithmes métier complexes sont joués sur une machine de développement.

   ➔ Confiance dans le code Métier

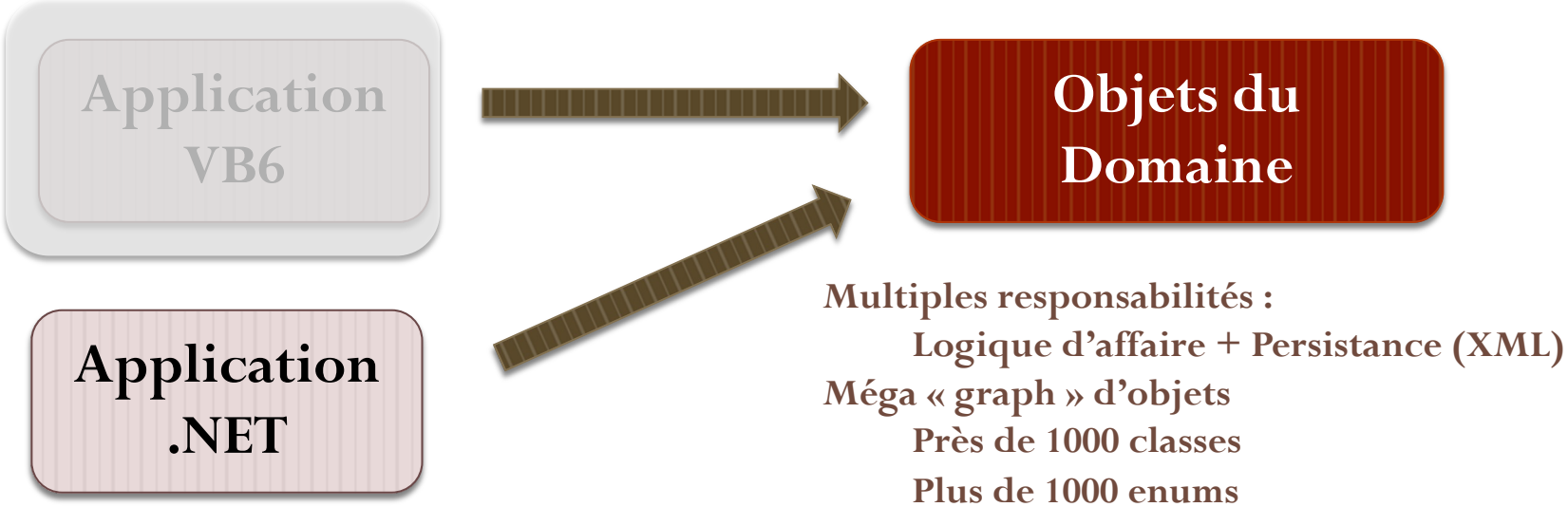
2. Lorsque la cible est enfin disponible, les suites de tests sont rejouées sur celle-ci.

   Les problèmes ne concerneront plus que les interfaces avec le HW et l'OS, le temps-réel, la concurrence et les ressources limitées en mémoire et charge de calcul.

   ➔ Confiance dans le système et ses interfaces

Refactorer ou tout recommencer ?

- Il faut briser le cycle « Développement ➔ Refonte totale »
- Faire du neuf avec du vieux : attitude RESPONSABLE
  - Une itération à la fois, refactoring après refactoring



- NOTA : faire les tests après coup n'encourage pas la réflexion sur les tests : restent immatures

Références, liens utiles

Martin FOWLER

- https://martinfowler.com/bliki/
- Objet du Domaine : https://martinfowler.com/eaaCatalog/domainModel.html
- Découpler Domain Objets du Domaine et Persistance : https://martinfowler.com/bliki/LocalDTO.html

Robert C. MARTIN (Uncle Bob)

- Livres « Clean Architecture », Robert C. Martin et http://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
- Code source exemple : https://github.com/karoldeland/CleanArchitectureLegacyCode
- Blog Clean Code : https://blog.cleancoder.com/uncle-bob/

- Livre « Working Effectively With Legacy Code » de **Michael Feathers**
- Architecture en oignons : https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/