Génie Logiciel : Processus de développement : Méthodologies non Agiles

Dr. Selma Belgacem Ben Mansour

ISSAT de Sousse

2021-2022

Plan du cours

Plan du cours

Cycle de vie d'un logiciel

- **Étude de faisabilité** : budget, compétences, ressources...
- **Spécification des besoins** : Déterminer les besoins de l'utilisateur et par la suite les fonctionnalités du logiciel.
- **Conception** : Déterminer l'architecture globale et détaillée du logiciel.
- **Codage** : implémentation des différentes fonctionnalités du logiciel.
- **Tests** : Essayer le logiciel sur des données d'exemple pour s'assurer qu'il fonctionne correctement.
- **Installation** chez le client.
- **Maintenance** : modification si nécessaire.

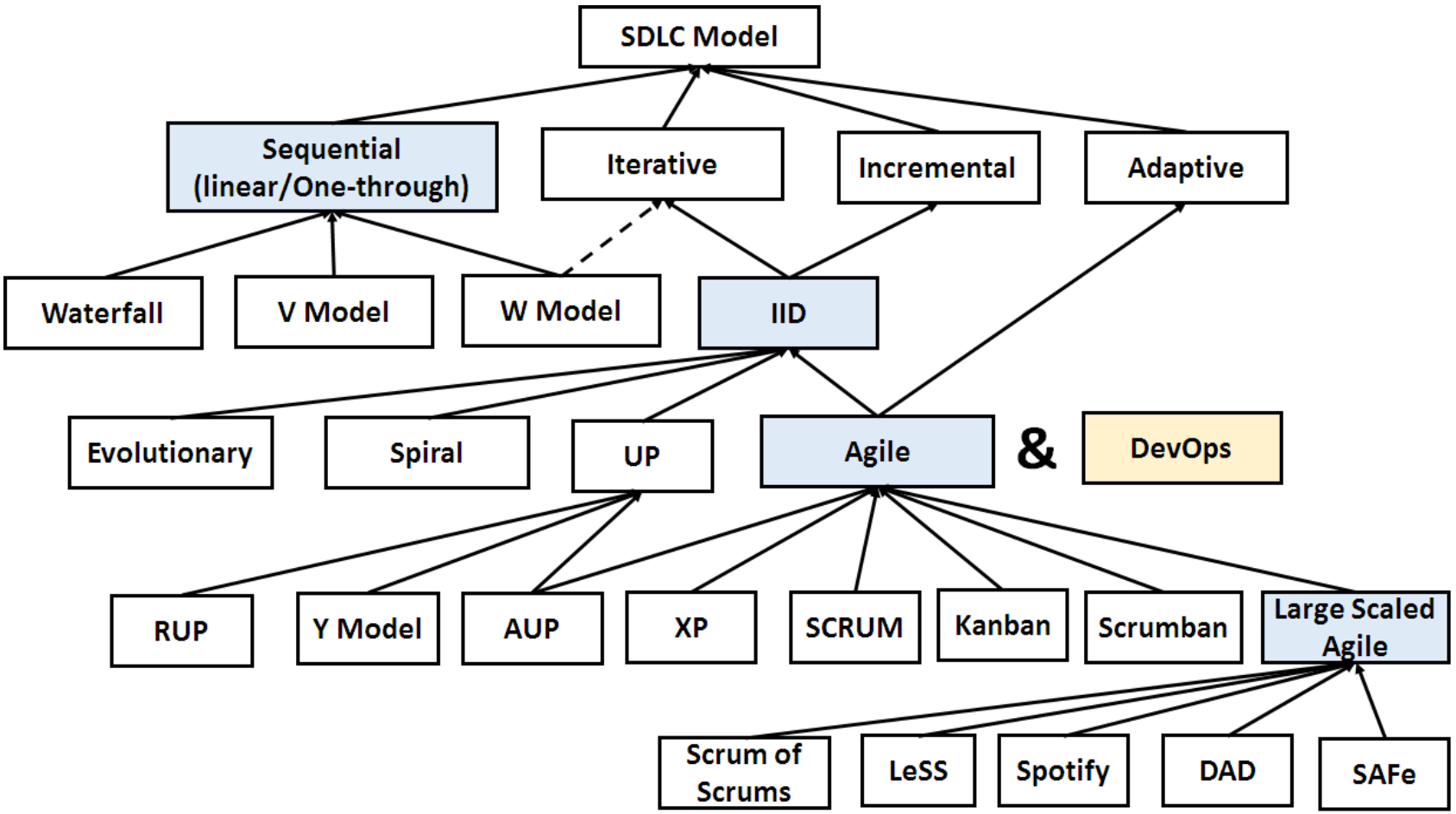
Types de maintenance

- **maintenance corrective** : correction des **erreurs**.
- **maintenance préventive** : évolution du logiciel selon l'évolution des **technologies**.
- **maintenance perfective** : modification du logiciel pour répondre aux **besoins** supplémentaires de l'utilisateur,
- **maintenance adaptative** : évolution du logiciel selon l'évolution de l'**environnement** extérieur.

→ Il existe différents **processus de développement** d'un logiciel qui décrivent l'organisation des étapes du cycle de vie d'un logiciel.

Classification des processus de développement d'un logiciel

- **SDLC** : *Software Development Life Cycle* = processus de développement d'un logiciel.



Selma Belgacem

Prototypage

- **Prototype** : version réduite du futur logiciel partiellement réalisée ne présentant pas toutes les fonctionnalités.
- **Maquette** : ensemble d'interfaces utilisateur, fonctionnelles ou non fonctionnelles, du futur logiciel.
- Un prototype ou une maquette est utile pour une **validation par le client** et pouvant être jetable sauf le prototype évolutif.
- **Prototype jetable** : construit rapidement par un environnement de développement léger. *Exemple* : une maquette (interfaces) d'un futur logiciel mobile peut être construite avec l'éditeur *PowerPoint* contrairement aux interfaces réelles du logiciel dont la construction est prévues avec *Android Studio*.

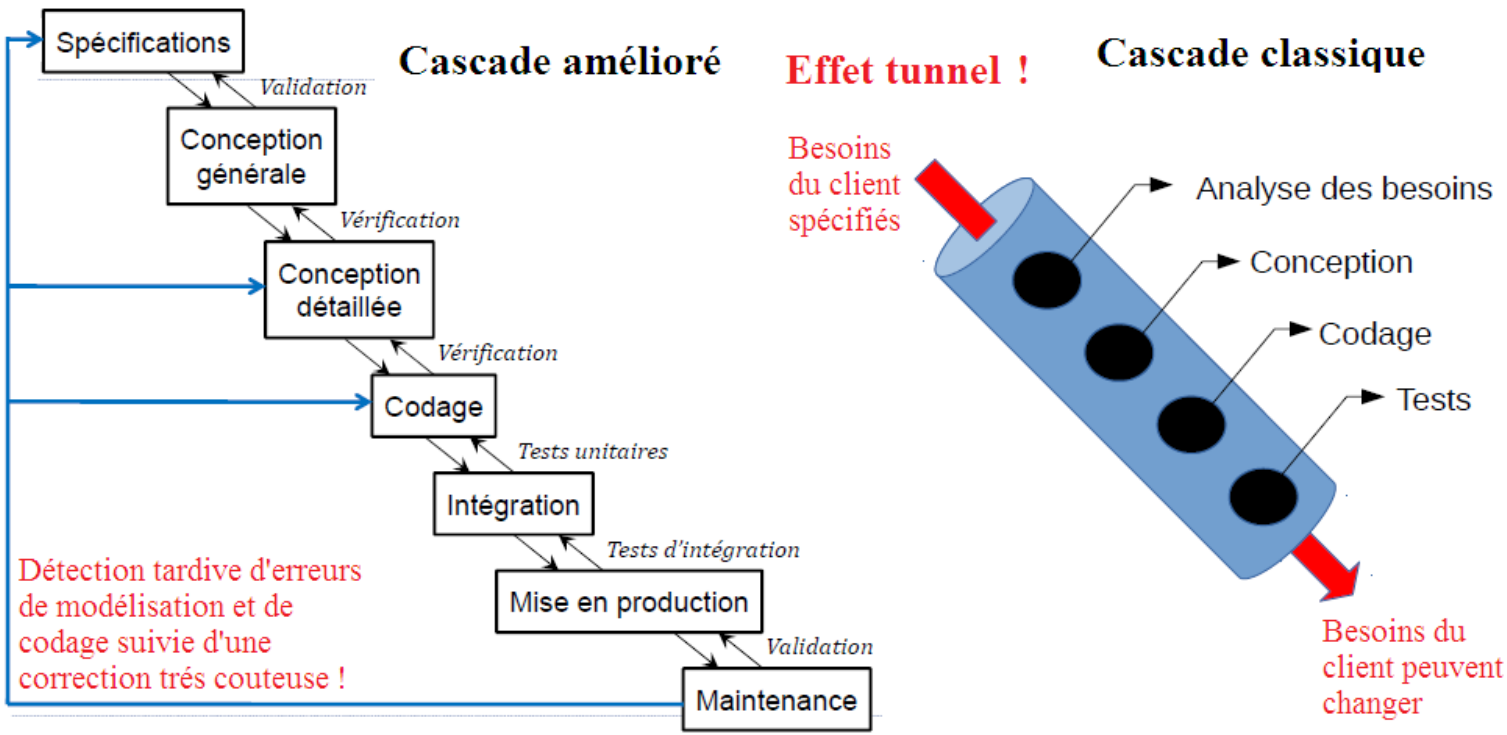
Prototypage

- Types des prototypes :
  - → **Prototype exploratoire (maquette)** : produit lors de la spécification des besoins pour confirmer les fonctionnalités demandées par le client (front-end).
  - → **Prototype expérimental** : produit lors de la conception pour confirmer la structuration interne du logiciel (back-end).
  - → **Prototype évolutif** : produit tout au long du cycle de vie du logiciel en commençant par construire un logiciel embryon jusqu'à aboutir au produit final à travers un ensemble d'itérations.

Plan du cours

Modèle en cascade

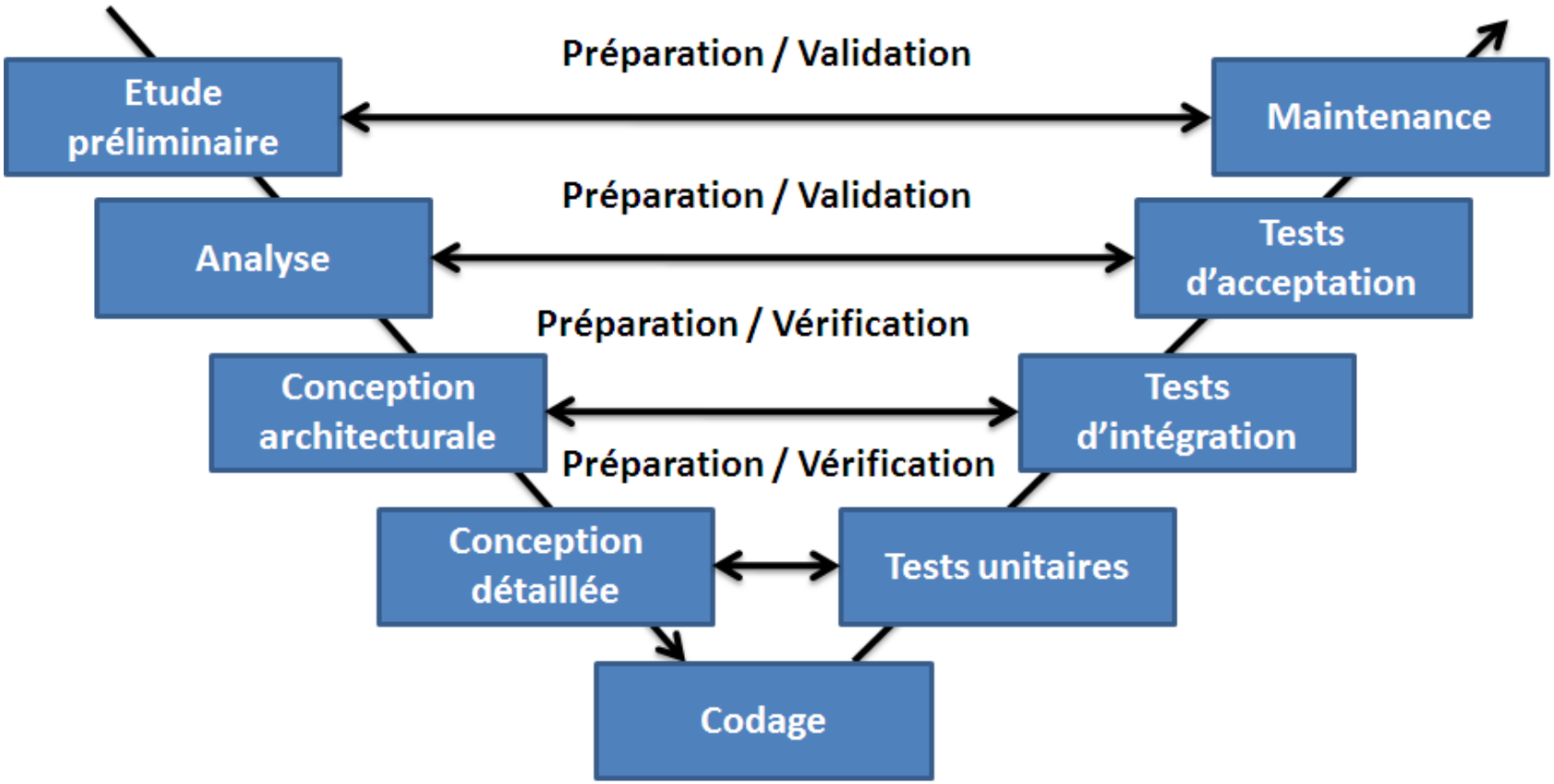
- Le modèle en cascade a permis d'**identifier** les étapes du cycle du vie d'un logiciel.
- La fin de chaque étape est caractérisée par une **documentation** récapitulative.
- La version améliorée de ce modèle rajoute la possibilité de **chevauchement** entre chaque deux étapes successives afin de **valider** la transition entre les deux.



Mejdi BLAGHGI, "*Génie Logiciel*", Notes de cours, ISET Djerba.
Régis Clouard, "*Méthode agile : une première approche*", Notes de cours, ENSICAEN.

Modèle en V

- Le modèle en V est une **variante** du modèle en cascade.
- L'apport du modèle en V est la définition et la planification des **tests** appropriés au niveau de chaque étape.



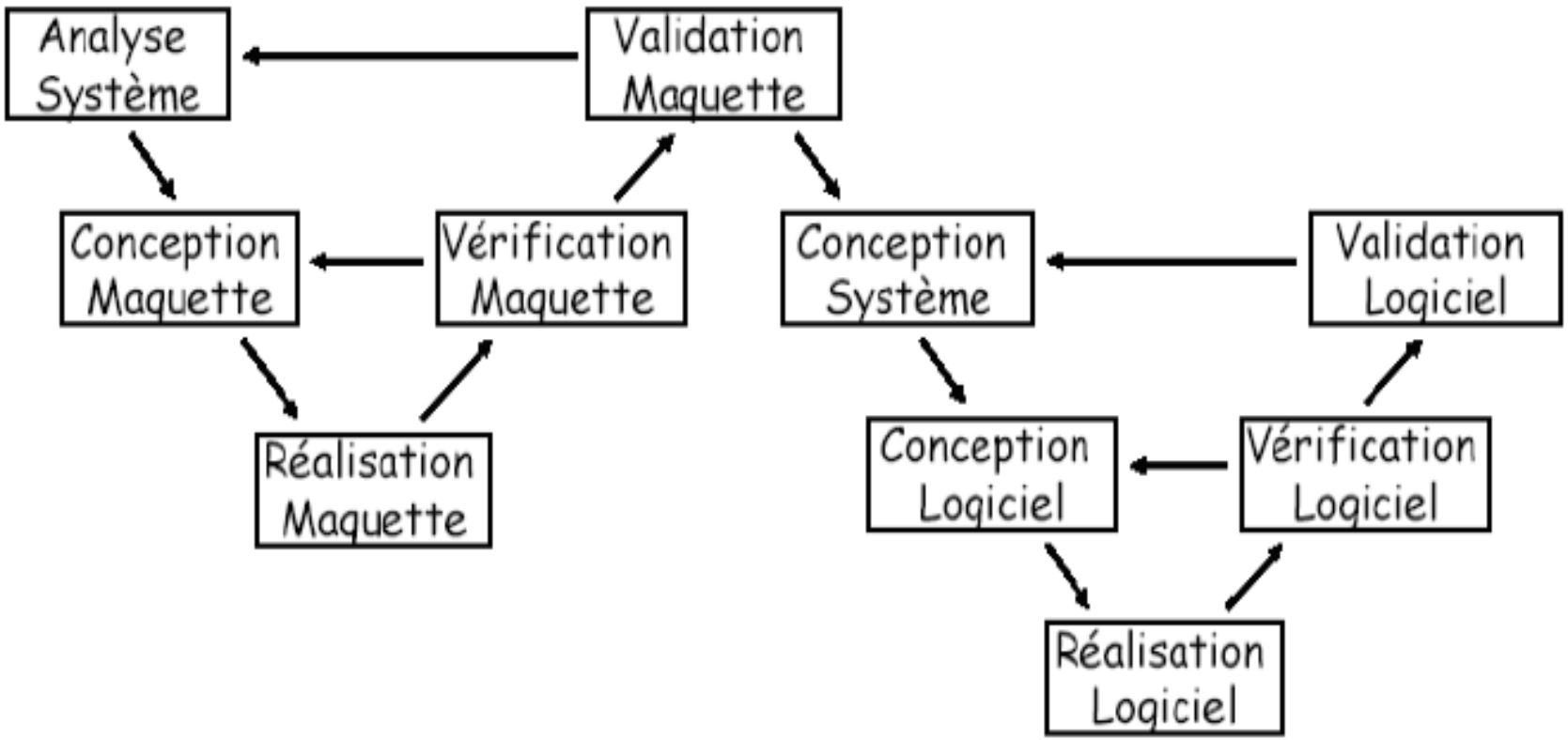
Amami Maha, "*Cours génie logiciel*", ISG de Tunis, 2010.

Types de tests

- **Test unitaire** : les composants du logiciel sont testés **individuellement** en simulant les entrées nécessaires.
- **Test d'intégration** : vérifier si les **interactions** à travers les interfaces et l'échange de données se déroulent correctement entre les différents composants du logiciel lorsqu'ils sont exécutés ensemble.
- **Test d'acceptation (recette)** : Test effectué par l'**utilisateur final** du logiciel avec des données du client (pas nécessairement dans l'environnement réel d'exécution chez le client) pour vérifier que le logiciel répond bien aux besoins exigés (certains besoins peuvent ne pas être exprimés lors de l'étape de l'analyse).
- **Validation** : "*Est que le logiciel répond bien aux **besoins** de l'utilisateur ?*"
- **Vérification** : "*Est ce que le logiciel est bien conforme à la **modélisation** ?*"

Modèle en W

• Le modèle W est une variante du modèle en V en le multipliant par 2. La première partie est dédiée à l'élaboration d'une **maquette** utile pour la validation de la **spécification des besoins** avant la réalisation du logiciel concret.

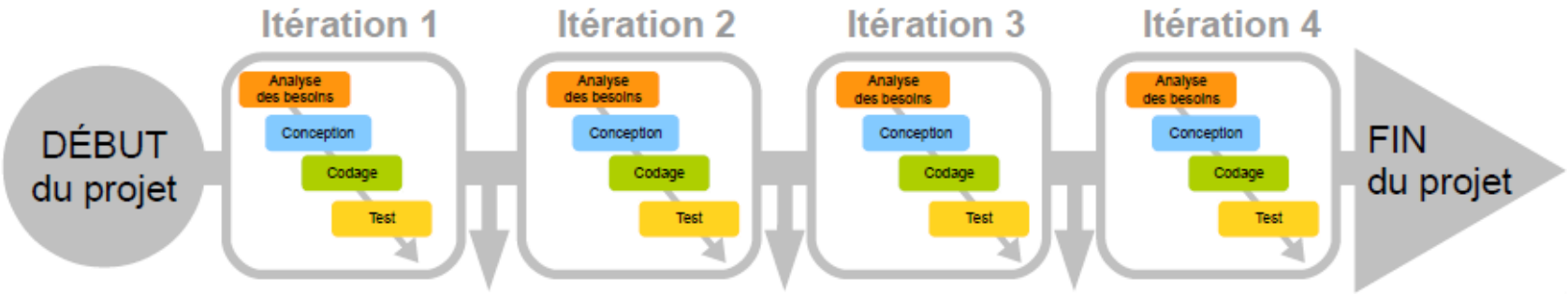


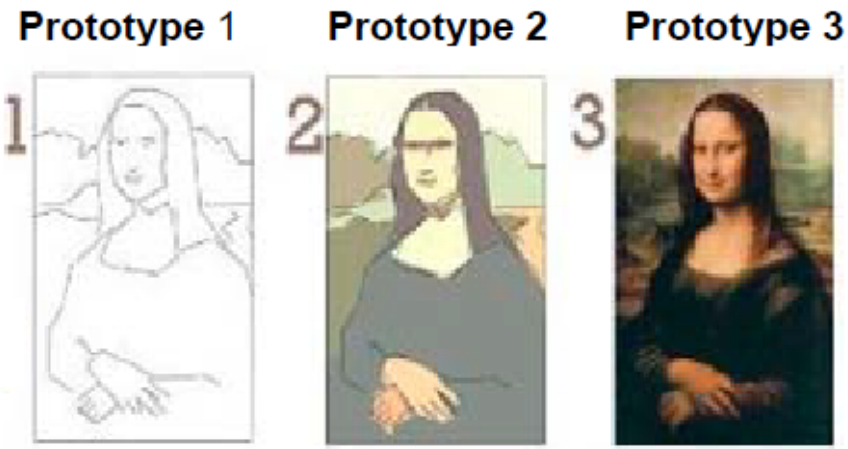
https://www.scribd.com/document/340152545/Chap-Intro-Conception-Des-Systemes-d-Information

Plan du cours

Modèle itératif

- Le **modèle itératif** est sous la forme d'un ensemble d'itérations successives où **chaque itération est un modèle en cascade de courte durée** (environ 2 semaines).
- Le résultat d'une itération est une **version complète du logiciel** qui n'est pas totalement raffinée et approfondie (pouvant être un prototype jetable) utile pour la **validation** par le client.

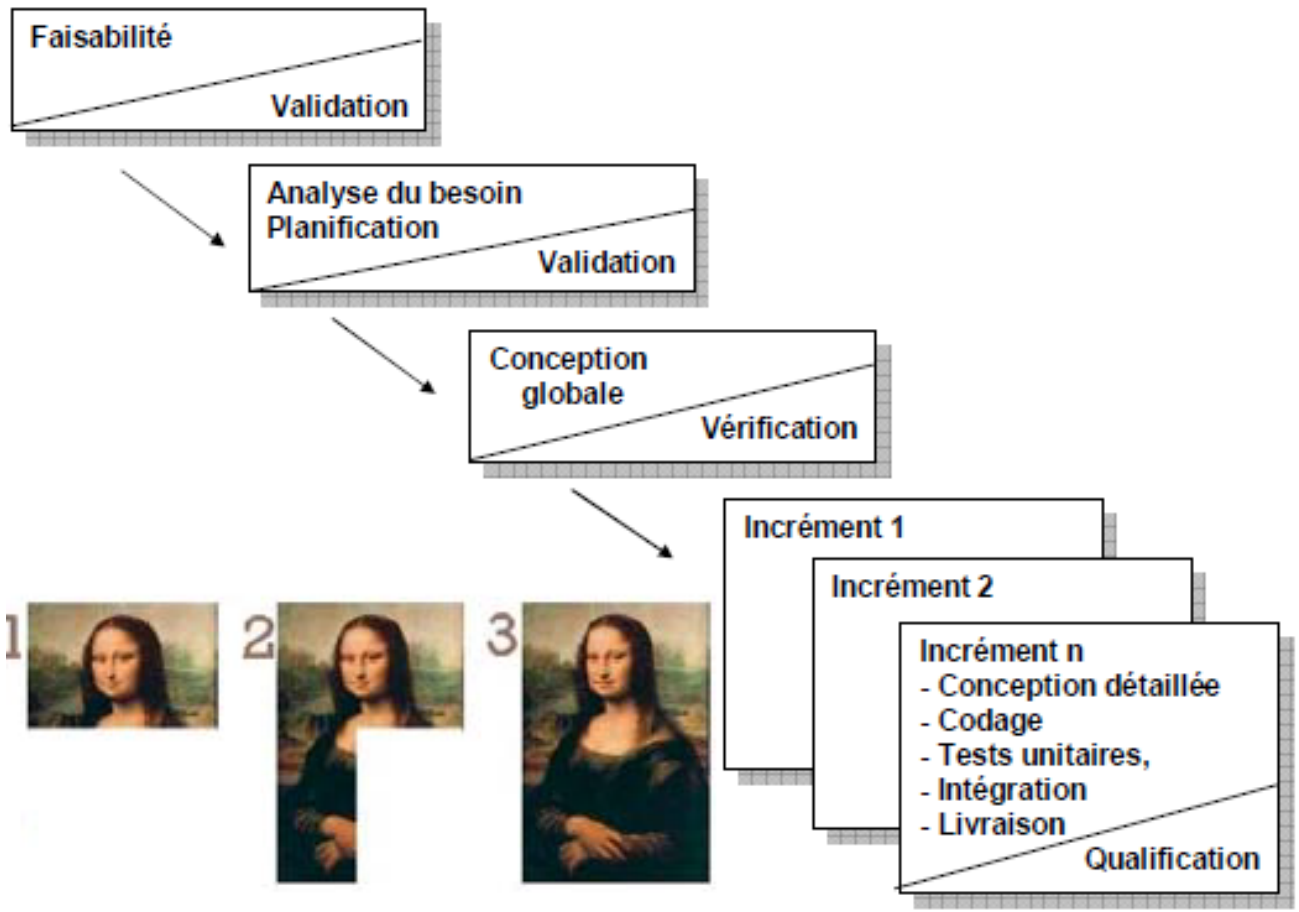




Régis Clouard, "*Méthode agile : une première approche*", ENSICAEN, http://www.djouabri.com/cours/2016-2017/ACOO/Ch2-CycleVie-.pdf

Modèle incrémental

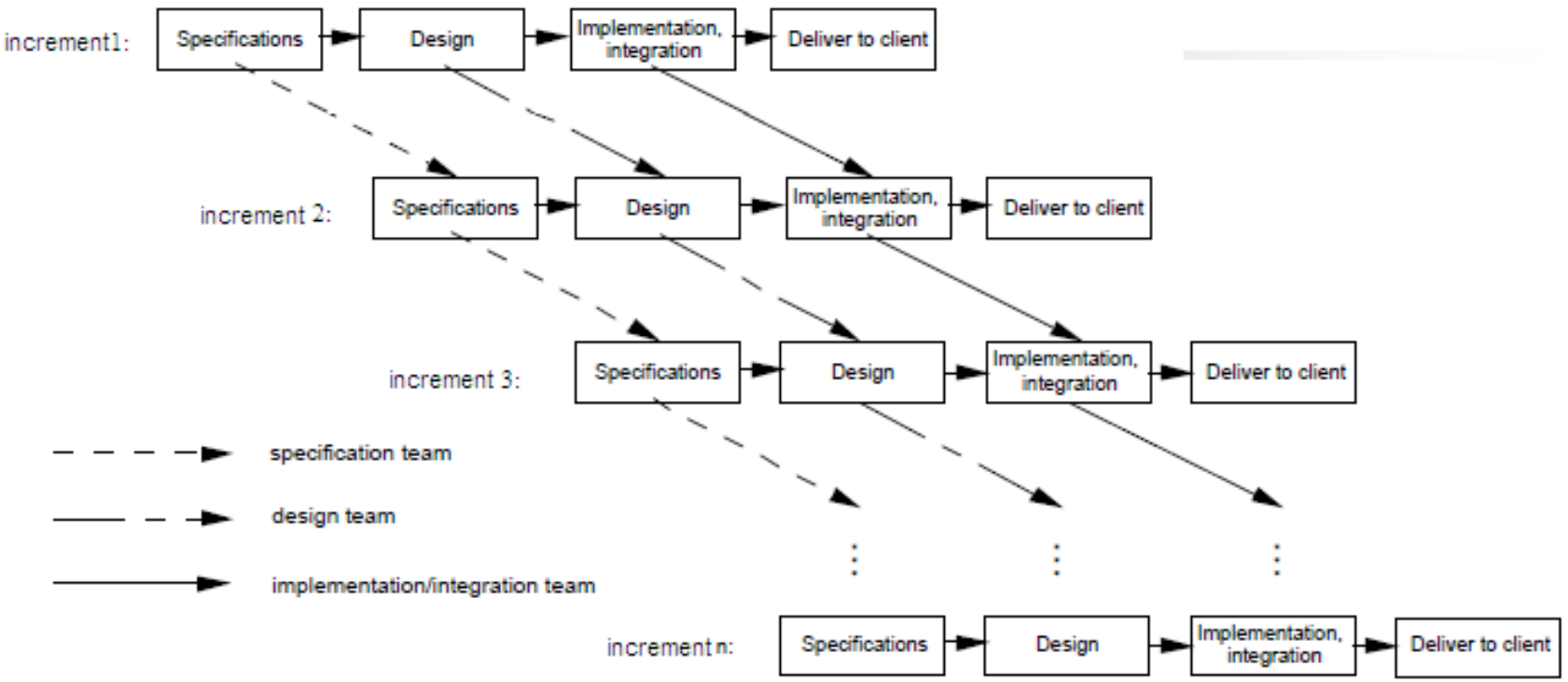
- Le **modèle incrémental** consiste à partitionner (selon les exigences du clients) le modèle du logiciel en un ensemble d'**incréments** individuellement cohérent, **fonctionnels et livrables** au client (exemple : un SGF pour un OS).



Florence PETIT, "*Cycle de vie des systèmes informatiques*", 2007,
http://www.djouabri.com/cours/2016-2017/ACOO/Ch2-CycleVie-.pdf

Modèle incrémental

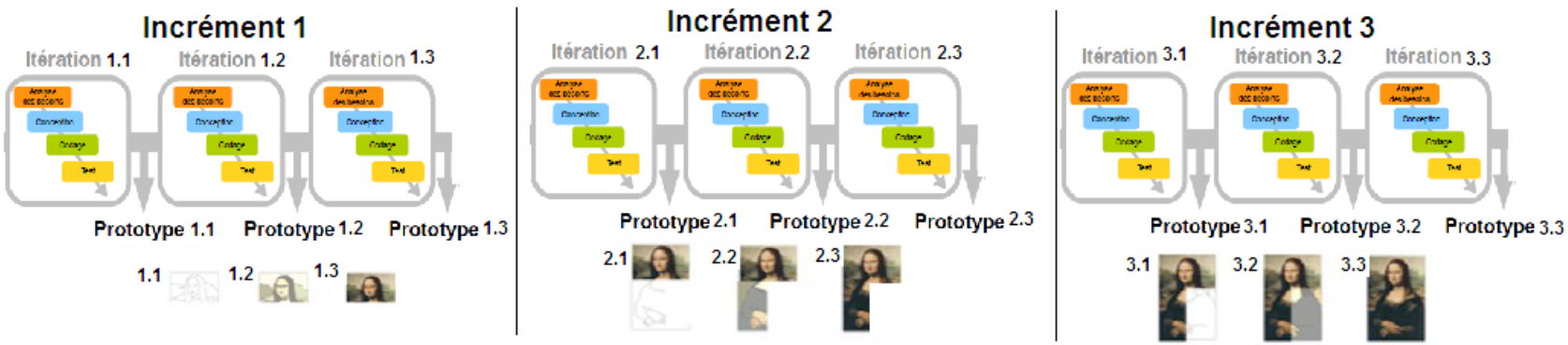
- Il existe une **deuxième version** du modèle incrémental où les incréments sont développés **séparément** dés l'étape de **spécification** des besoins.
- L'ensemble d'incréments peuvent être développés **successivement ou parallèlement** (déconseillé pour des raisons d'amélioration et des problèmes d'intégration).



Edited, Lydie du Bousquet, "*Processus de développement Cycles de vie*", IMAG.

Modèle itératif et incrémental (IID)

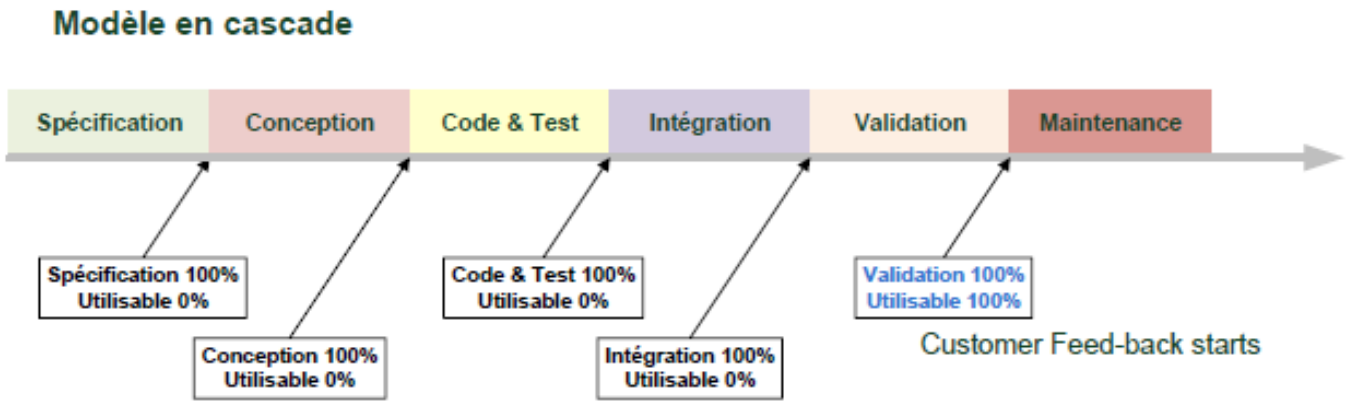
- Dans le cadre du **modèle itératif combiné à l'incrémental (IID)**, chaque incrément est développé à l'aide un ensemble d'itérations.
- Chaque **itération** est un modèle en cascade qui génère un **prototype de l'incrément**.

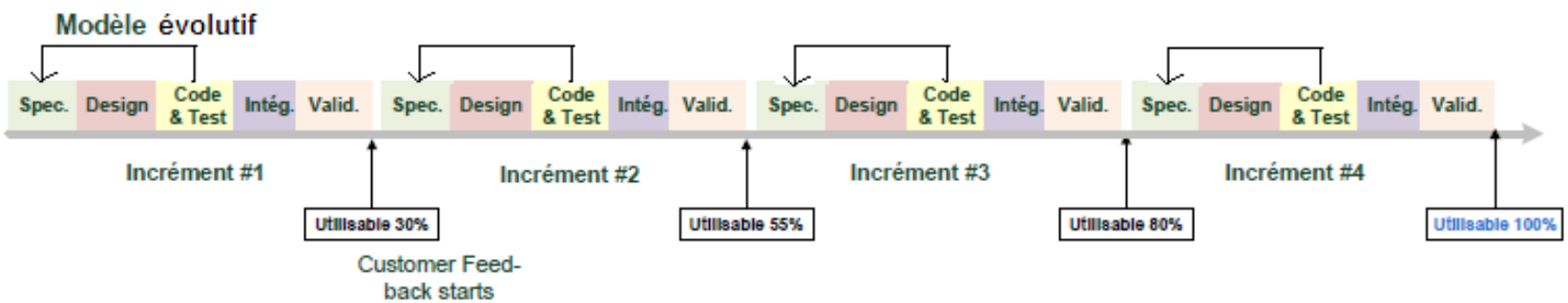


Edited, Régis Clouard, "*Méthode agile : une première approche*", ENSICAEN, M. Blay-Fornarino, "*Processus Unifié*", 2011.

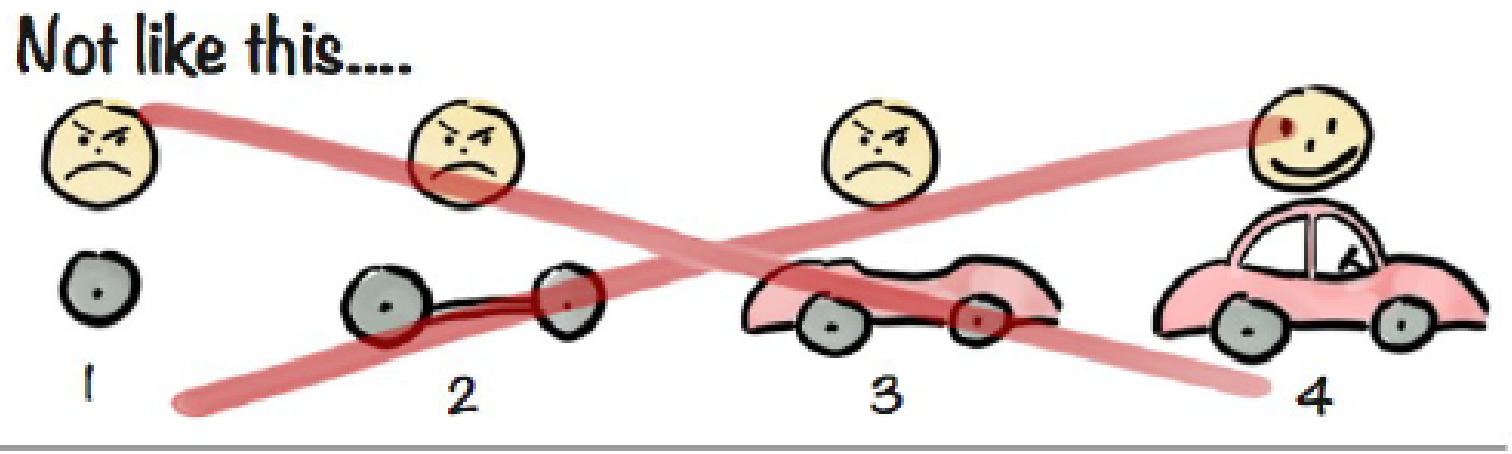
Modèle évolutif (*Evolutionary*)

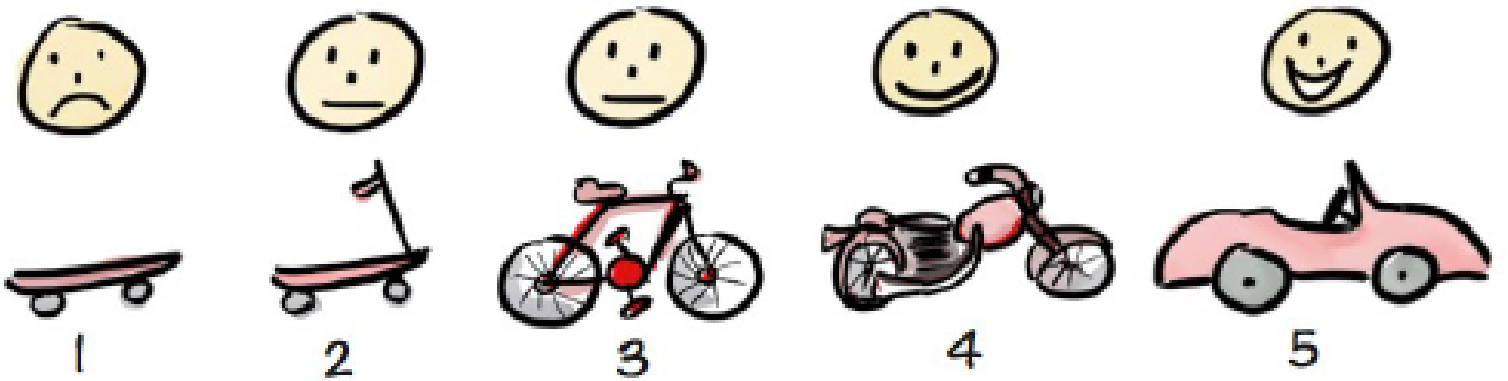
• Le modèle **évolutif** est un modèle **IID** où chaque incrément doit être **livré au client** pour obtenir son ***feedback*** très tôt dans le cycle de vie du logiciel ce qui améliore considérablement la **qualité du logiciel**, particulièrement, la conformité avec les **besoins de l'utilisateur** et la réduction du **coût de la maintenance**.





Modèle IID évolutif





Skateboard : assurer le transport de A vers B

Scooter: éviter le risque de tomber

Bike: transport plus rapide et plus confortable sans être couvert

motorcycle: ajouter un moteur (+rapide +confortable)

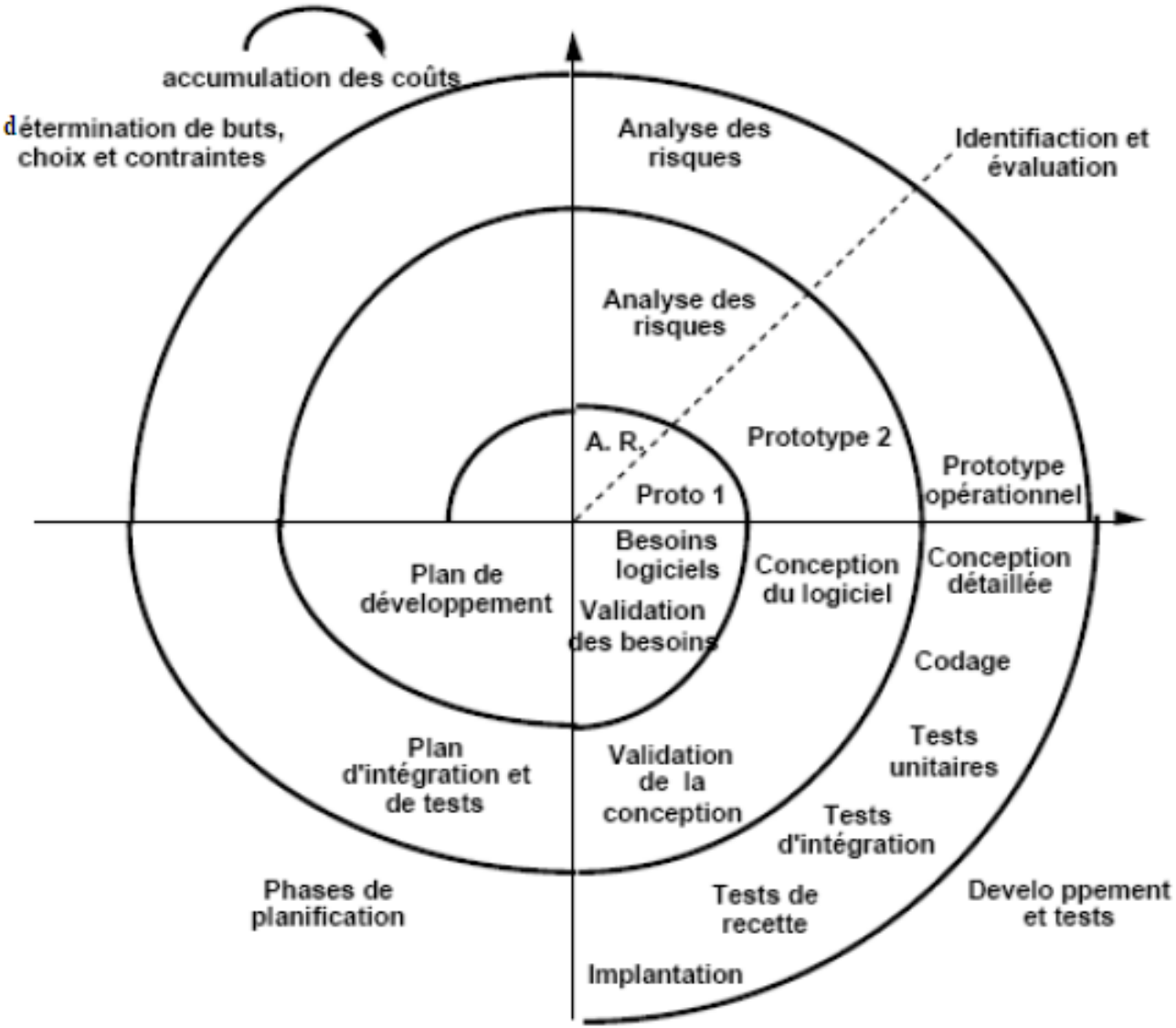
convertible car: +rapide, +confortable, +de places, non couverte

https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp

Modèle IID évolutif

- **MVP** (*Minimum Viable Product*) : chaque incrément délivré doit ajouter de la valeur à la version globale du logiciel en ajoutant **une vraie fonctionnalité** souhaité par le client.
- Le prototypage nous permet de **découvrir** et prendre en considération **les préférences de l'utilisateur** (pour l'exemple de la voiture : couleur, couvert, non couvert...) et d'éviter les modifications lors de la maintenance. Fixer les besoins de l'utilisateur à priori **n'est pas toujours satisfaisant et sûr** !
- Dans certains cas, le prototypage évolutif permet de **réduire le coût** de développement d'un logiciel en évitant des versions supplémentaires et plus complexes du logiciel : les **versions intermédiaires** du logiciel peuvent satisfaire le client qui annulera lui-même des ajouts supplémentaires de fonctionnalités qu'il jugera inutiles après avoir testé ces versions intermédiaires.

Modèle en spirale



http://www.djouabri.com/cours/2016-2017/ACOO/Ch2-CycleVie-.pdf

Modèle en spirale

- Le modèle en spirale est un **modèle IID évolutif cyclique** où chaque cycle se déroule en **quatre phases** :
  1. **Fixer les objectifs** : déterminer les contraintes et les objectifs du cycle à partir de l'analyse des besoins ou/et des résultats du cycle précédent.
  2. **Analyser les risques** : étudier les difficultés qui peuvent se produire et prévoir des **alternatives** en utilisant éventuellement du **prototypage**.
  3. **Développer l'application** : appliquer le cycle de vie complet d'un logiciel y compris les différents types de tests selon **le modèle en cascade où en V**.
  4. **Planifier** : faire un **bilan** du cycle achevé, **évaluer son résultat par le client** et planifier le **cycle suivant**.

Modèle en spirale

- Chaque cycle est planifié en prenant en compte les possibles **risques**. Un **risque** est un problème ou une difficulté qui peut se produire à un certain moment du cycle de vie du logiciel.
- Types de risques :
  - **Risques technologiques** :

→ exigences qui dépassent les capacités technologiques,
→ non maîtrise des technologies impliquées dans le développement ou dans l'utilisation du logiciel,
→ évolution et changement des technologies pendant le cycle de vie du logiciel...

  - **Risques liés au processus de développement** :

→ mauvaise gestion du projet,
→ calendrier non réalisable et budget dépassant les vraies moyens,
→ calendrier abandonné sous la pression des clients,
→ insuffisance des données,
→ développement de fonctionnalités non compatibles avec les besoins du client...

Modèle en spirale

- Types de risques :
  - **Risques humains** :

  → incapacité et manque de motivation du personnel,

  → manque et surestimation des compétences,

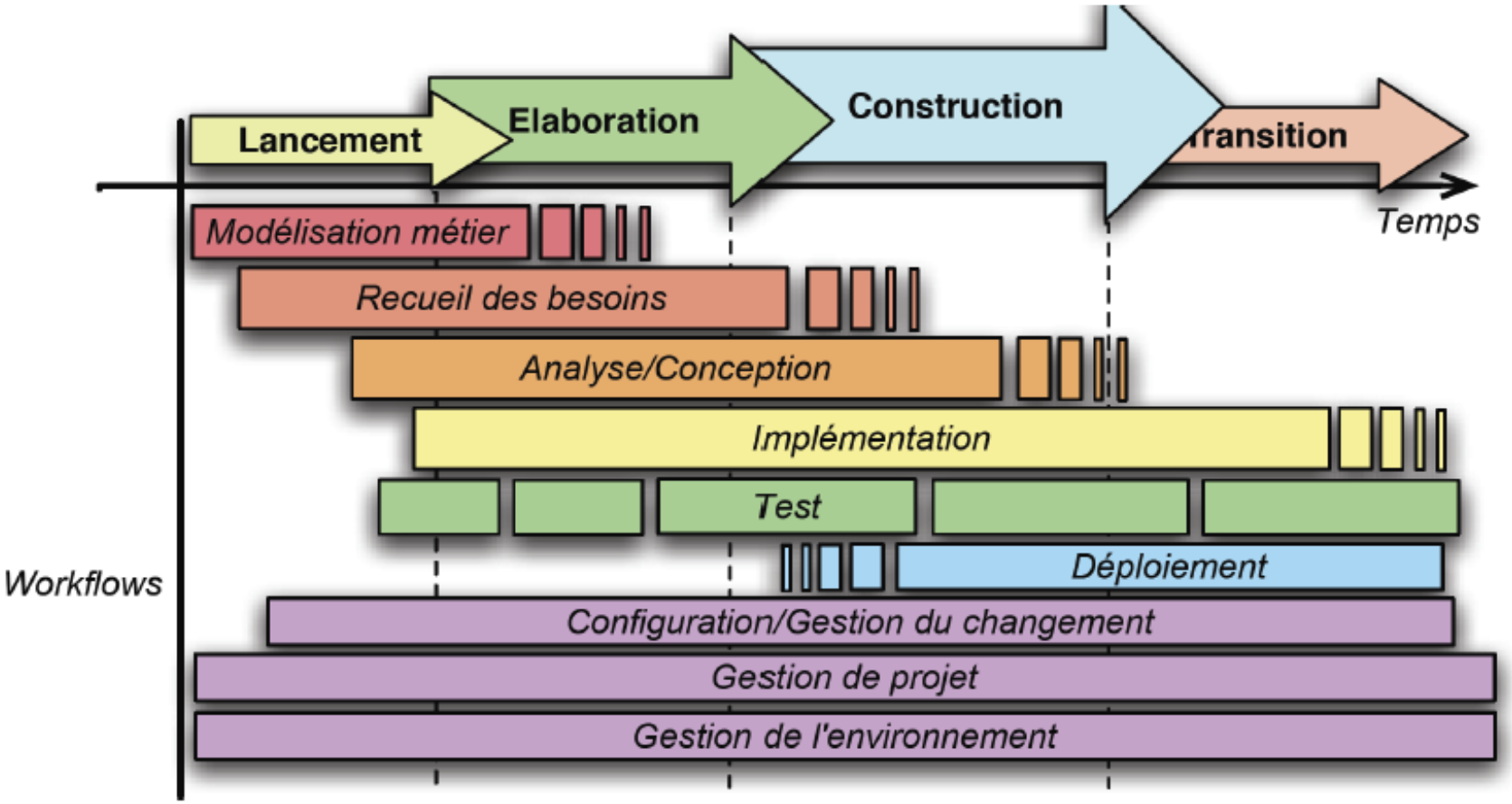
  → manque de communication et de coopération entre les membres de l'équipe de projet...
- Les risques sont traités par ordre de **priorité**.
- L'analyse des risques rend les cycles de cette méthode très longs (peuvent durer des mois) à cause du recours fréquent aux formations du personnel → **adéquat pour les grand projet non maîtrisé**.

Unified Process (UP)

- Le *Unified Process* est une méthode **IID** pouvant être **évolutive**.
- Les principes du *Unified Process* sont les même que pour le *Rational Unified Process*. L'appellation ***RUP*** est spécifique à l'IBM.
- Les méthodes **UP** sont **centrées sur** :
  - les **use-cases** : le cycle de vie du logiciel est piloté par les **fonctionnalités** exigées par le client (chaque livrable doit ajouter une fonctionnalité au logiciel par ordre de priorité).
  - l'**architecture** du logiciel : il est préférable que le logiciel soit modélisé par différents patrons d'architecture et selon différentes vues (**diagrammes UML**) pour couvrir tous les aspects du logiciel.
  - les **risques** : les premières versions du logiciel doivent résoudre les risques les **plus prioritaires** et les plus importants en premier lieu.
  - La livraison d'un ensemble de **documents** à la fin de chaque phase.

Unified Process (UP)

- Les phases des méthodes Unified Process :



Pierre Gérard, "*Processus de Développement Logiciel*", Université de Paris 13.

Unified Process (UP)

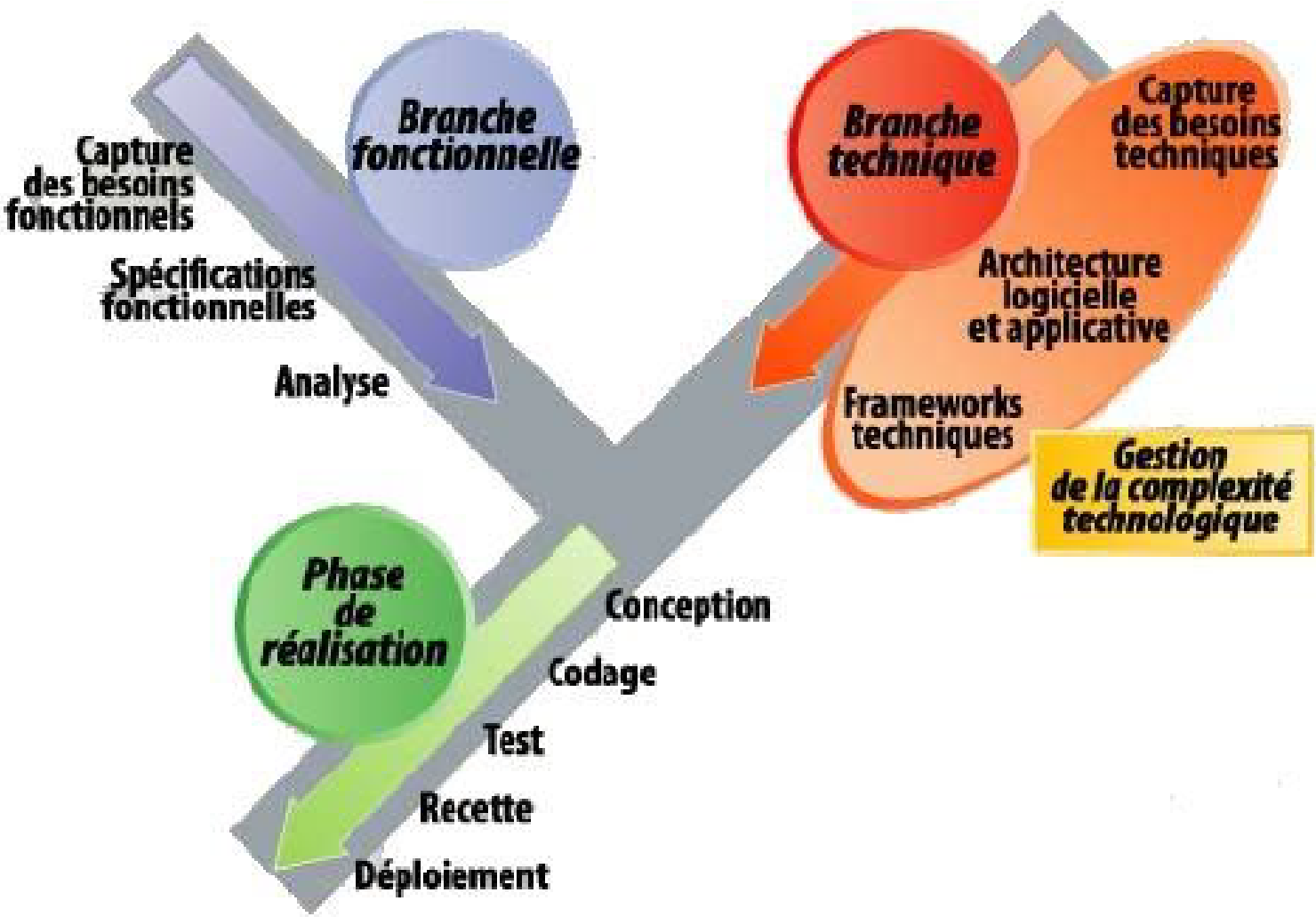
- ***Inception*** :
  - étude de faisabilité, planification du projet,
  - estimation des coût, estimation des risques,
  - maquettage pour les cas d'utilisation les plus importants,
  - Choisir les patrons d'architecture adéquats.
- ***Elaboration*** :
  - valider les besoins de l'utilisateur (**diagramme de cas d'utilisation**),
  - élaborer l'architecture et les diagrammes conceptuels du système (diagramme de paquetage, diagramme de classe),
  - implémenter le noyau architectural du logiciel (principaux composants) à travers plusieurs itérations de courte durée (en exécutant les test unitaires et d'intégration).
  - déterminer et résoudre les risques les plus importants à travers le prototypage,
  - préparer un plan précis en estimant le coût pour la phase de construction.

Unified Process (UP)

- ***Construction*** :
  - modéliser les détails internes du logiciel en utilisant les **diagrammes UML** (diagramme d'activités, diagramme de séquences, diagramme de communication, diagramme d'états-transitions, diagramme d'interactions global),
  - implémenter les fonctionnalité détaillés du logiciel à travers plusieurs itérations de courte durée (en exécutant les test unitaires et d'intégration),
  - livrer une version exécutable du logiciel à la fin de chaque itération qui peut être testée et validée par le client (optionnel),
  - déterminer et résoudre les risques les moins importants,
  - livrer à la fin de cette phase un produit final.
- ***Transition*** :
  - installer le produit fini chez le client,
  - réaliser les tests d'acceptation (recette) et résoudre les erreurs liées,
  - possibilité d'élaborer plusieurs itérations de raffinement du logiciel selon le *feedback* du client,
  - résoudre d'éventuels problèmes de compatibilité avec l'environnement de déploiement.

Méthode *Two Tracks Unified Process (2TUP)* : Modèle en Y

- L'idée principale de ce modèle est de dissocier les besoins fonctionnels et les souhaits techniques définis par l'utilisateur.



Noël NOVELLI, "*Génie Logiciel*", Université d'Aix-Marseille.

Références

- Renaud Marlet, "*Cycle de vie*", LaBRI / INRIA, 2007.
- Francois Jacquenet, "*GÉNIE LOGICIEL - PROCESSUS DE DEVELOPPEMENT DU LOGICIEL*", Faculté des Sciences et Techniques, Université Saint-Etienne.
- Lydie du Bousquet, "*Processus de développement Cycles de vie*", IMAG.
- Pierre Gérard, "*Processus de Développement Logiciel*", Université de Paris 13.
- `http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf`
- `https://fr.wikipedia.org/wiki/Unified_process`
- `https://en.wikipedia.org/wiki/Comparison_of_project_management_software`
- `https://template.pro/logiciels-diagramme-de-gantt/`