Cours 1 - Gestion d'un projet informatique et cycle de vie.
BUT 2 - R3.03 Analyse

Halim Djerroud <halim.djerroud@uvsq.fr>

LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

révision : 2.2

Plan

Gestion d'un projet informatique et cycle de vie

1. Gestion de projet informatique
   - Développer des systèmes complexes
   - Le Génie logiciel
   - Les activités du Génie Logiciel
2. Cycle de vie du logiciel
3. Vers d'autres méthodes de gestion
   - Constat
   - Les méthodes agiles
   - Cycles de vie multiniveaux, l'exemple du processus unifié

Objectif du cours

Objectif du cours :

- **Gestion de projet** pour réaliser des **solutions informatique** avec une **fortes contraintes de qualités** dans le cadre des **systèmes d'information des organisations**.

C'est quoi un systèmes d'information :

Définition 1.1 : Système d'information (SI) source (Reix et al. 2016)

Un système d'information est un ensemble organisé de ressources (matériels, logiciels, personnel, données et procédures) qui permet de :

- acquérir l'information (sur différents supports)
- traiter l'information (effectuer des opérations)
- stocker l'information (conserver l'information)
- communiquer l'information (transmettre : éditer, imprimer, etc.)

Composition d'un SI

Nature hétérogène :

- Le système informatique est composé de ressources matérielles, logicielles, humaines et organisationnelles.
- Les ressources matérielles sont les composants physiques du système, tels que les ordinateurs, les serveurs, les logiciels, les réseaux, etc.
- Les ressources logicielles sont les programmes informatiques qui permettent de faire fonctionner le système.
- Les ressources humaines sont les personnes qui utilisent le système et qui en assurent le développement, la maintenance et le support.
- Les ressources organisationnelles sont les règles, les procédures et les structures de l'organisation qui encadrent le développement et l'utilisation du système.

SI = Matériel + Logiciel + Humain + Organisation.

Tâches accomplies par un SI :

- La collecte, le stockage et la diffusion de l'information
- Le traitement de l'information
- La prise de décision
- La communication
- La collaboration

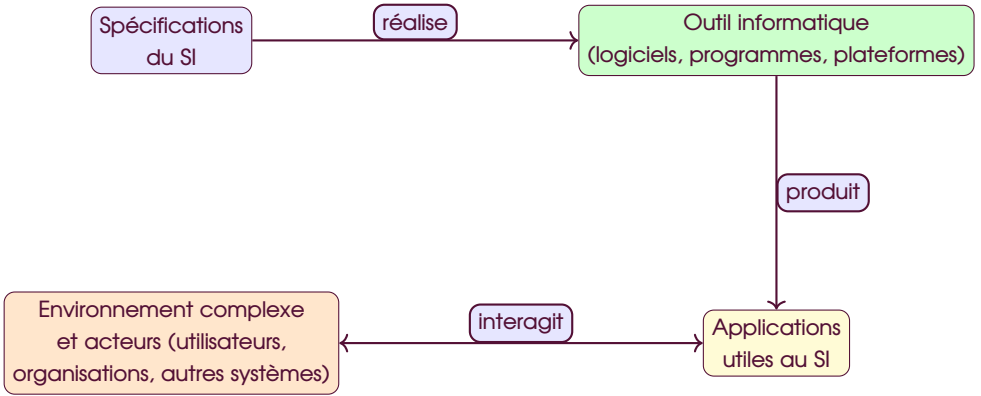
Attention :

Un **système informatique** fait partie d'un système d'information mais **ne constitue pas** nécessairement un **système d'information**.

A quoi sert l'outil informatique

- Les outils de l'informatique permettent de réaliser les spécifications d'un SI.
- Plusieurs systèmes informatiques peuvent interagir pour réaliser ces spécifications, ou remplacer d'autres outils.

A quoi sert l'outil informatique



Méthodologie pour la gestion de projet informatique

Nous nous intéressons donc ici aux **méthodes** nécessaires au **développement des applications** utiles aux systèmes d'information, et s'intégrant de fait dans un environnement complexe avec différents acteurs.

Afin de proposer une méthodologie pour la gestion de projet informatique dans le contexte des systèmes d'information il nous faut définir :

- Le cadre théorique spécifique à l'informatique et la manière dont on peut **découper les activités** d'un développeur
- Cet enchaînement définit ce que l'on appelle le **cycle de vie logiciel**
- Il n'y a pas de cycle unique.
- Il est possible d'adapter son cycle de vie logiciel et définir un processus métier particulier pour le développeur.

Le Génie logiciel

- Le génie logiciel nous permet de définir une méthodologie pour la réalisation d'un logiciel.

Le Génie Logiciel est la discipline liée à tous les aspects de la production du logiciel complexe et avec d'importantes contraintes de qualité. Elle est liée à l'application de théories, de méthodes et à l'utilisation d'outils pour le développement logiciel d'une façon professionnelle. Elle favorise et permet le travail en équipe.

Les objectifs du génie logiciel :

- Adopter une approche systématique et organisée
- Utiliser les techniques et outils appropriés selon la nature du problème, les contraintes de développement et les ressources

Principes du génie logiciel (GL)

1. **Rigueur et formalisme** : faciliter la communication entre développeurs et avec le client.
2. **Abstraction** : Identifier et définir les concepts du projet ainsi que les éléments qui le compose.
3. **Modularité** (décomposition en sous-système) : proposer une décomposition en sous-système
4. **Anticipation** : prendre en compte contraintes techniques et organisationnelles.
5. **Généralisation** : réutiliser les **composants** et les **méthodes**.
6. **Croissance incrémentale** : faire évoluer le logiciel étape par étape.

Le génie logiciel est à l'informatique ce que le génie civil est à la construction : une ingénierie structurée qui transforme une idée en un logiciel robuste, maintenable et évolutif.

Les activités du Génie Logiciel

Grandes étapes nécessaires pour réaliser un logiciel.

Pour mener à bien un projet informatique, le développeur ou l'équipe de développement doit réaliser un ensemble d'activité allant de l'analyse des besoins du projet à la maintenance du logiciel produit en passant par son implémentation et les tests.

1. Analyse des besoins
2. Spécification
3. Conception
4. Programmation
5. Validation et vérification (Tests)
6. Intégration et gestion des configurations

(a) Analyse des besoins (ADB)

- **Rôle :**
  - Identifier le problème
  - Documenter les exigences
  - Impliquer les utilisateurs et les experts dans le domaine de l'application
- **Entrées :**
  - Cahier des charges (rédigé par le client)
  - Données fournies par les experts du domaine et par les utilisateurs potentiels
- **Résultats :**
  - Document d'ADB destiné aux utilisateurs et utile pour les développeurs.
  - Recueil des besoins
  - Cahier des charges (rédigé de manière technique avec le client)

Remarque

Attention, de nombreux produits ne correspondent pas aux besoins du client. On peut identifier plusieurs causes, comme par exemple

- Le client ne sait pas ce qu'il veut
- Il existe des problèmes de communication entre le client et l'équipe de développement.
- L'équipe de développement ne comprend pas la politique d'organisation du client
- On observe un changement des exigences au cours du projet
- Les délais ne sont pas raisonnables
- etc.

(b) Spécification

- **Rôle :**
  - Décrire de façon précise le système à construire, le **QUOI** et non le comment.
- **Entrées :**
  - Document d'ADB destiné aux utilisateurs et utile pour les développeurs.
  - Recueil des besoins
  - Cahier des charges (rédigé de manière technique avec le client)
- **Résultats :**
  - Documentation textuelle avec description de scenarii (cas d'utilisation) possiblement appuyée par des diagrammes (ex : std UML)

(c) Conception

- **Rôle :**
  - Enrichir la spécification du logiciel en l'orientant vers la réalisation, le **COMMENT** et non le quoi.
- **Entrées :**
  - Spécifications
  - Détail de la plateforme de développements (dans le Recueil des besoins)
- **Résultats :**
  - **Modèle** défini dans une documentation textuelle appuyée par des diagrammes (ex : std UML)

(d) Programmation

- **Rôle :**
  - Spécifier le système en utilisant un langage de programmation.
- **Entrées :**
  - Documentation de la conception détaillée
  - Spécification (si pas de conception détaillée)
- **Résultats :**
  - Code qui implémente les fonctionnalités
  - Documentation technique du code
  - Dépôt, historique de gestion de version

(e) Validation (e1) et Vérification (e2)

- **Rôle :**
  - (e1) Répond à la question "Le logiciel satisfait-il les attentes de l'utilisateur ? "
  - (e2) Répond à la question "Le logiciel satisfait-il les spécifications ? "
- **Entrées :**
  - Documentation de la conception détaillée pour les tests unitaires
  - Documentation de la conception architecturale pour les tests d'intégration
  - Documentation d'ADB pour les tests d'acceptation
- **Résultats :**
  - Cas de tests
  - Analyse des résultats d'exécution des cas de tests

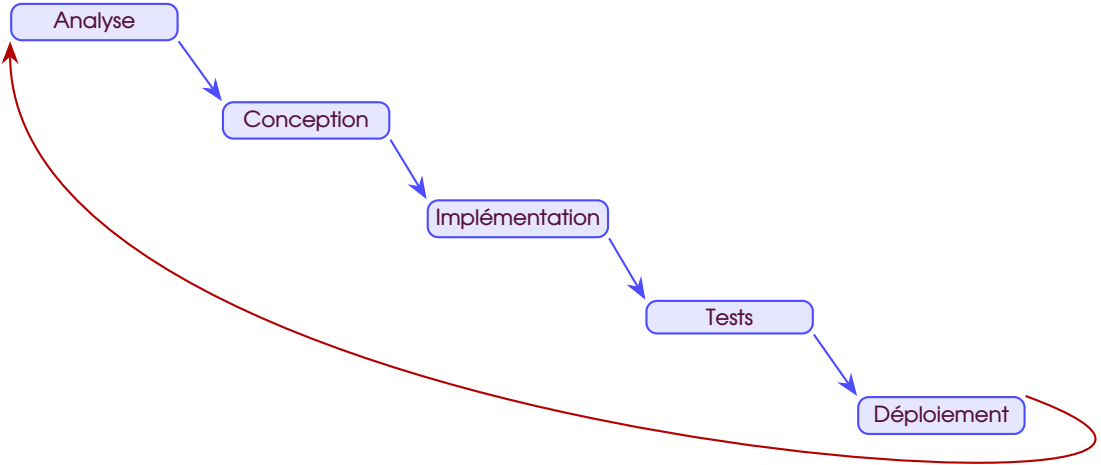
(f) Intégration et gestion des configurations

- **Rôle :**
  - Décrire le système et ses évolutions.
- **Entrées :**
  - Version du logiciel
- **Résultats :**
  - Dossier de maintenance

Processus de développement

- L'équipe projet suit un processus de développement : ensemble d'activités et de résultat qui conduisent à la création d'un produit logiciel
- Le processus de développement indique la forme dans laquelle les activités sont connectées entre-elles. L'ordre dans lequel s'enchaînent les activités s'appelle le cycle de vie du produit logiciel

Cycle de vie linéaire Cascade



Cycle de vie en cascade

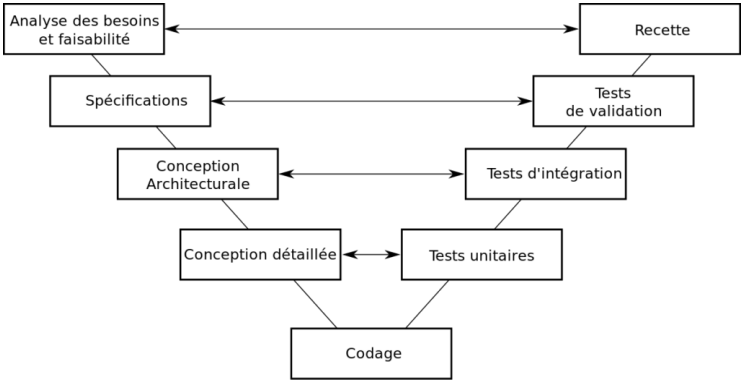
Principe du cycle en cascade :

- Déroulement **séquentiel** des étapes du projet (analyse → conception → implémentation → tests → déploiement).
- Chaque phase doit être **terminée et validée** avant de passer à la suivante.
- Approche simple et **structurée**, bien adaptée aux projets courts et bien définis.

Limites principales du modèle en cascade :

- **Rigidité** : difficile de revenir en arrière une fois une étape terminée.
- **Peu flexible** face aux changements de besoins en cours de projet.
- **Décalage temporel** : le produit final n'est visible qu'à la fin du cycle.
- **Risque élevé** de découverte tardive d'erreurs ou de mauvaises interprétations.
- **Faible implication du client** durant la réalisation.

Cycle de vie linéaire V



Cycle de vie en V

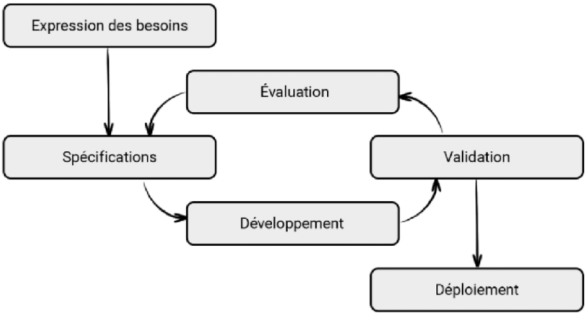
Principe du cycle en V :

- Chaque étape de conception est associée à une étape de test.
- Les tests doivent être définis **en même temps** que la conception.
- Descente du V = **définition du système**.
- Remontée du V = **validation et vérification**.

Limites principales du modèle en V :

- **Rigidité** : difficile de revenir en arrière si une erreur est découverte.
- **Peu flexible** face aux changements de besoins en cours de projet.
- **Coût élevé** des corrections lorsqu'elles apparaissent tardivement.
- **Manque d'agilité** : pas adapté aux projets innovants ou incertains.
- **Client impliqué tardivement** : la validation réelle se fait souvent à la fin.

Cycle de vie itératif et incrémental



Cycle de vie incrémental

Principe du cycle incrémental :

- Le développement est découpé en **incréments** (versions partielles du logiciel).
- Chaque incrément apporte **de nouvelles fonctionnalités utilisables**.
- Le client peut disposer rapidement d'une **version fonctionnelle minimale**.
- Favorise l'**adaptation** et l'**amélioration progressive**.

Limites principales du modèle incrémental :

- **Complexité de gestion** : nécessite une planification rigoureuse.
- **Intégration progressive difficile** : risque de problèmes d'assemblage entre incréments.
- **Dépendance au découpage** : un mauvais découpage des incréments peut nuire au projet.
- **Coûts plus élevés** si des tâches sont répétées d'un incrément à l'autre.
- **Risque d'inachèvement** : le client peut rester trop longtemps sur une version partielle.

Les méthodes agiles

Définition : Les méthodes agiles sont des approches de gestion de projet logiciel qui privilégient :

- L'**adaptation au changement** plutôt que le suivi rigide d'un plan.
- La **collaboration** avec le client et les utilisateurs.
- La livraison **rapide et fréquente** de versions fonctionnelles.
- La **communication** et l'implication de l'équipe.

Valeurs du Manifeste Agile (2001) :

- Les **individus et leurs interactions** plus que les processus et les outils.
- Un **logiciel opérationnel** plus qu'une documentation exhaustive.
- La **collaboration avec le client** plus que la négociation contractuelle.
- L'**adaptation au changement** plus que le suivi d'un plan.

Méthodes agiles et cycles de vie

Relation entre méthodes agiles et cycles de vie :

- Les cycles de vie classiques (Cascade, V, Incrémental) définissent une **organisation des étapes**.
- Les méthodes agiles apportent une **philosophie de gestion** :
  - Itérations courtes et incrémentales.
  - Validation continue avec le client.
  - Adaptation permanente aux changements.
- Chaque itération agile correspond à un **mini-cycle de vie complet** :
  - Analyse → Conception → Implémentation → Tests → Livraison.

Comparatif :

- **Cycle classique** : un seul grand cycle, produit visible à la fin.
- **Cycle agile** : enchaînement de petits cycles, produit évolutif visible rapidement.

Principales méthodes agiles

Quelques méthodes agiles largement utilisées :

- **Scrum** :
  - Organisation par **sprints** (cycles courts de 2 à 6 semaines).
  - Rôles : *Scrum Master*, *Product Owner*, équipe de développement.
  - Livraison d'un produit opérationnel à chaque sprint.
- **Extreme Programming (XP)** :
  - Mise en avant de la **qualité du code**.
  - Programmation en binôme, intégration continue, tests unitaires automatisés.
- **Kanban** :
  - Gestion visuelle du travail via un **tableau de tâches**.
  - Flux continu, limitation du nombre de tâches en cours (WIP).
- **Lean Software Development** :
  - Inspiré du lean manufacturing.
  - Élimination du gaspillage, amélioration continue, respect des personnes.

Méthode agile : Scrum

- Organisation du projet en **sprints** (cycles courts de 2 à 6 semaines).
- Chaque sprint aboutit à un **produit fonctionnel et livrable**.
- L'équipe est **auto-organisée** et pluridisciplinaire.
- Le client participe via le **Product Owner**.

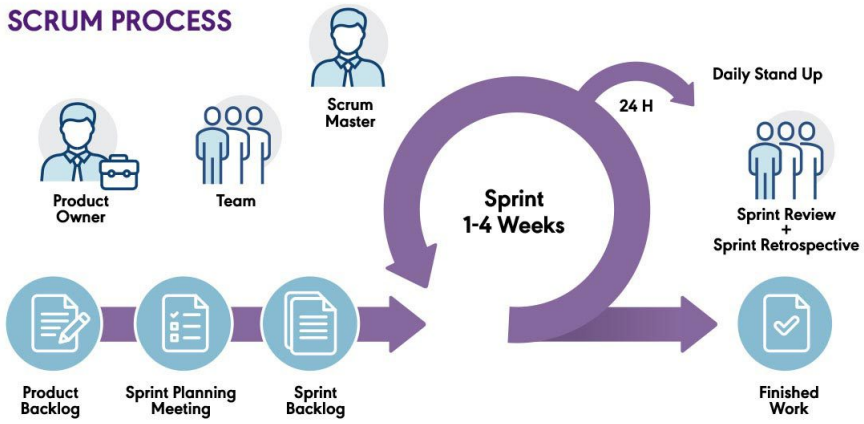
Rôles clés :

- **Product Owner** : définit et priorise les besoins (backlog produit).
- **Scrum Master** : facilite la méthode et supprime les obstacles.
- **Équipe de développement** : réalise le produit incrémental.

Artefacts principaux :

- **Backlog produit** : liste des fonctionnalités à développer.
- **Backlog sprint** : tâches prévues pour le sprint en cours.
- **Incrément** : version livrable du produit.

SCRUM


SCRUM PROCESS
Scrum Master
Daily Stand Up
24 H
Product Owner
Team
Sprint 1-4 Weeks
Sprint Review + Sprint Retrospective
Product Backlog
Sprint Planning Meeting
Sprint Backlog
Finished Work

Méthode agile : Extreme Programming (XP)

- Développement en **itérations courtes**, avec livraisons fréquentes.
- **Tests unitaires automatisés** écrits en même temps que le code.
- **Refactoring** régulier pour garder le code simple et maintenable.
- **Programmation en binôme** (pair programming) pour améliorer la qualité.
- **Intégration continue** : code intégré et testé plusieurs fois par jour.
- Collaboration étroite et continue avec le **client**.

Avantages :

- Améliore la **qualité du logiciel** et réduit les bugs.
- Grande **flexibilité** face aux changements de besoins.
- Favorise le **travail en équipe** et le partage de connaissances.

Limites :

- Exige une **discipline stricte** de l'équipe.
- Programmation en binôme peut sembler **coûteuse**.
- Moins adapté aux projets très grands et dispersés.

Méthode agile : Kanban

- Gestion du flux de travail à l'aide d'un **tableau Kanban**.
- Les tâches sont représentées par des **cartes** organisées en colonnes :
  - **À faire** (To Do)
  - **En cours** (In Progress)
  - **Terminé** (Done)
- Limitation du **travail en cours** (WIP : Work In Progress).
- Approche **visuelle et continue** : le flux de tâches est optimisé en permanence.

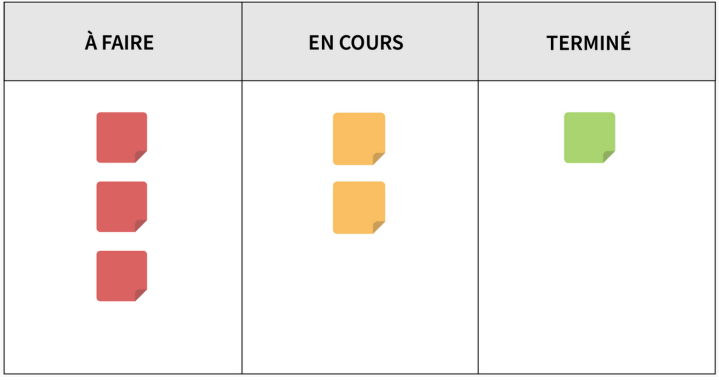
Avantages :

- Grande **simplicité** et mise en place rapide.
- Améliore la **transparence** et la communication dans l'équipe.
- Permet de **visualiser les blocages** et d'optimiser le flux.

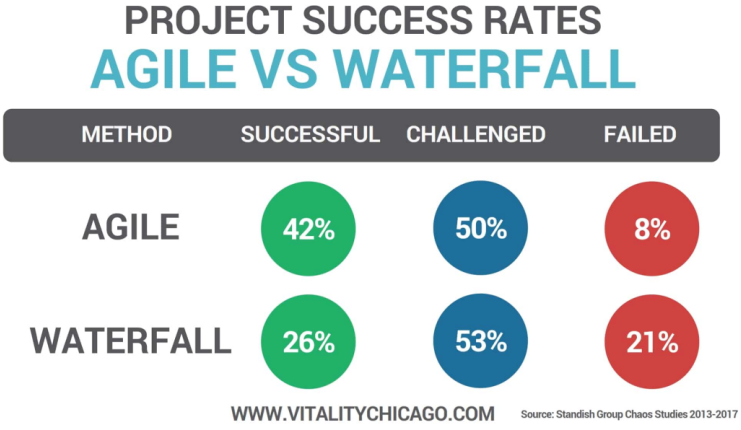
Limites :

- Moins structuré que Scrum : nécessite une **autodiscipline élevée**.
- Ne définit pas de rôles ni de rituels spécifiques.
- Moins adapté aux **projets très complexes**.

Méthode agile : Kanban



Gain observé entre un choix Agile



Processus Unifié (*Unified Process*) (UP).

Processus Unifié

Le processus unifié (PU), ou « unified process (UP) » en anglais, ou « Unified Software Development Process (USDP) » est une famille de méthodes de développement de logiciels orientés objets. Elle se caractérise par une démarche itérative et incrémentale, pilotée par les cas d'utilisation, et centrée sur l'architecture et les modèles UML (wikipedia)

Processus Unifié (PU)

Définition : Le Processus Unifié (PU), ou *Unified Process*, est une méthode de développement logiciel :

- **Itérative et incrémentale** : le logiciel est construit par étapes successives.
- **Orientée objet** et centrée sur les **cas d'utilisation** (UML).
- **Centrée sur l'architecture** : l'architecture logicielle guide le développement.

Les 4 phases principales du PU :

1. **Création** : définir la vision, objectifs et périmètre du projet.
2. **Élaboration** : valider la faisabilité et définir l'architecture.
3. **Construction** : développer et tester les fonctionnalités.
4. **Transition** : livrer, former les utilisateurs et corriger les derniers défauts.

Chaque phase est divisée en **itérations**, livrant progressivement un logiciel de plus en plus complet.

Agile vs Processus Unifié

Points communs :

- Développement **itératif et incrémental**.
- Livraison progressive du logiciel.
- Importance de la collaboration avec le client.

Différences principales :

Méthodes agiles

- Cycles très courts (**1 à 6 semaines**).
- Peu de documentation, priorité au **logiciel opérationnel**.
- Adaptation rapide aux changements.
- Auto-organisation des équipes.
- Exemples : Scrum, XP, Kanban.

Processus Unifié (UP)

- Cycles plus longs et plus **formalisés**.
- Documentation UML importante (cas d'utilisation, diagrammes).
- Planification plus structurée.
- Conçu pour les projets complexes et de grande taille.
- Basé sur 4 phases : Création, Élaboration, Construction, Transition.

Livrable

Définition 3.2 (Livrable)

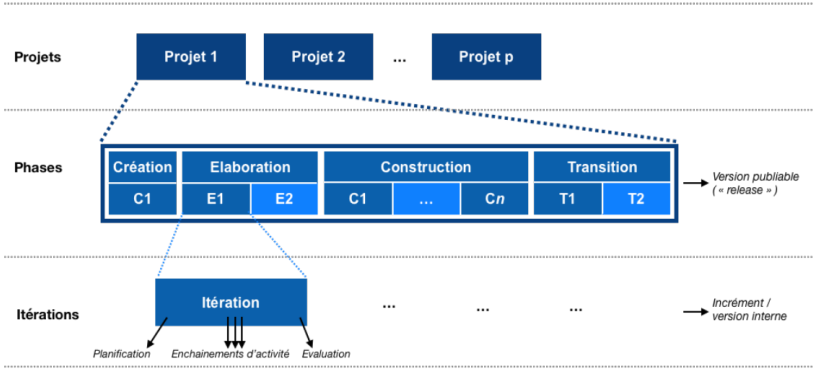
- un **corps de code source** réparti sur plusieurs composants pouvant être compilés et exécutés
- **un manuel** (et/ou une documentation technique)
- **les produits associés** (ou dépendances)

Il doit prendre en compte les besoins des utilisateurs, et de tous les intervenants (tous ceux amenés à l'exploiter) Il comprend donc :

- Un recueil des besoins
- Les cas d'utilisations (besoins fonctionnels)
- Les spécifications non fonctionnelles
- Les cas de test

Processus Unifié

Le Processus Unifié est un enchainement de cycles :



Principes des cycles

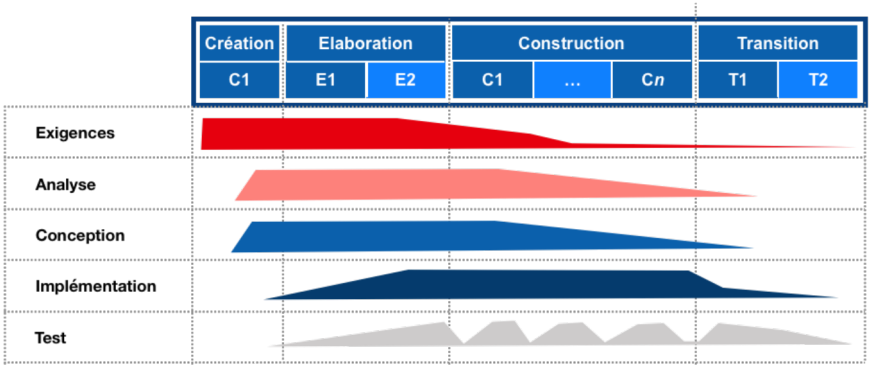
- À chaque cycle correspond une nouvelle version du logiciel (voir Figure 5)
- Un cycle est composé de 4 phases (voir Figure 6) :
  1. Étude préliminaire / création
  2. Élaboration
  3. Construction
  4. Transition
- Chaque phase se divise en itérations

Principes des cycles

- À chaque cycle correspond une nouvelle version du logiciel (voir Figure 5)
- Un cycle est composé de 4 phases (voir Figure 6) :
  1. **Étude préliminaire/Création (*Inception*)** : vision, acteurs, périmètre, cas d'utilisation majeurs.
  2. **Élaboration** : définition de l'architecture, validation des risques.
  3. **Construction** : développement incrémental, tests réguliers.
  4. **Transition** : livraison, formation des utilisateurs, corrections.
- Chaque phase se divise en itérations

Cycle

Un cycle := Phases composées d'itérations, chaque itérations est composées d'activités :



Une itération dans le PU

Chaque itération suit les activités classiques du génie logiciel :

- **Analyse des besoins** → identifier et clarifier.
- **Spécification** → définir le *quoi*.
- **Conception** → définir le *comment*.
- **Programmation** → coder les fonctionnalités.
- **Validation et vérification** → tester le code et le système.
- **Intégration et maintenance** → gérer les versions, préparer l'évolution.

Résultat : un *incrément exécutable* du logiciel.

Cas d'utilisation dans le PU

Les cas d'utilisation interviennent à toutes les phases du PU :

- **Création** : Identifier les acteurs et les cas d'utilisation principaux → cadrer le projet.
- **Élaboration** : Raffiner les cas critiques, détailler les scénarios → guider l'architecture.
- **Construction** : Implémenter les cas d'utilisation priorisés → incréments fonctionnels.
- **Transition** : Utiliser les cas d'utilisation comme base des **tests d'acceptation**.

Les cas d'utilisation sont le **fil conducteur du Processus Unifié**.

Cas d'utilisation dans le PU

Rôle central :

- Le PU est **piloté par les cas d'utilisation**.
- Les cas d'utilisation décrivent les **besoins fonctionnels du système**.
- Ils servent de **fil conducteur** tout au long du projet :
  - Création → identification des cas d'utilisation principaux.
  - Élaboration → détailler les cas critiques et guider l'architecture.
  - Construction → implémenter les cas d'utilisation priorisés.
  - Transition → valider le système via les scénarios utilisateurs.

Prochain cours

Prochain objectif : Apprendre à **modéliser les cas d'utilisation** avec UML.

Nous verrons :

- Comment identifier les **acteurs** et leurs interactions.
- Comment représenter les scénarios sous forme de **diagrammes UML**.
- Comment relier ces modèles aux autres artefacts du PU.

Les cas d'utilisation seront notre **point d'entrée** vers l'analyse et la conception UML.