Exposé : Introduction à la gestion de projet

I - Introduction

La **gestion de projet** est une démarche visant à organiser de bout en bout le bon déroulement d'un projet. Les résultats attendus du projet sont appelés **fournitures** ou **livrables**.

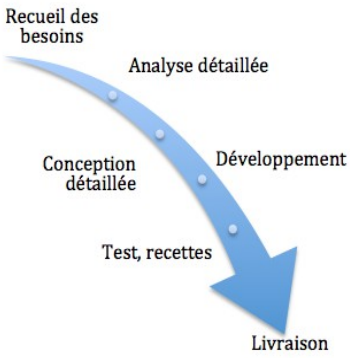
II - Les méthodes de développement d'un projet informatique

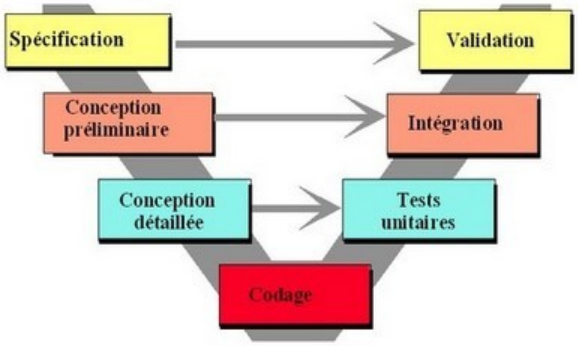
II.1. Introduction

L'objectif du **chef de projet** est de pouvoir mener son projet à terme en respectant les délais et le budget alloué. Pour atteindre cet objectif, il doit prendre en compte **les trois contraintes liées au projet** (**Coût**, **Calendrier** et **Contenu**) et il peut avoir recours à plusieurs **méthodes de gestion de projet** : Les méthodes **classiques** et les méthodes **itératives**.

II.2. Les méthodes classiques

Depuis toujours, les projets sont gérés avec les méthodes dites « **classiques** » qui se caractérisent par recueillir les besoins, définir le produit, le développer et le tester avant de le livrer. On parle alors ici d'une approche prédictive « **cycle en Cascade** ou **Waterfall** » ou « **cycle en V** ».





Dans un « **cycle en V** ou **Cascade** », **les risques sont détectés tardivement** puisqu'il faut attendre la fin du développement pour effectuer la phase de test. Plus le projet avance, plus l'impact des risques augmente. C'est ainsi que sont nées les méthodes **itératives** (ou **incrémentales**), qui tente de formaliser une approche plus pragmatique et maniable.

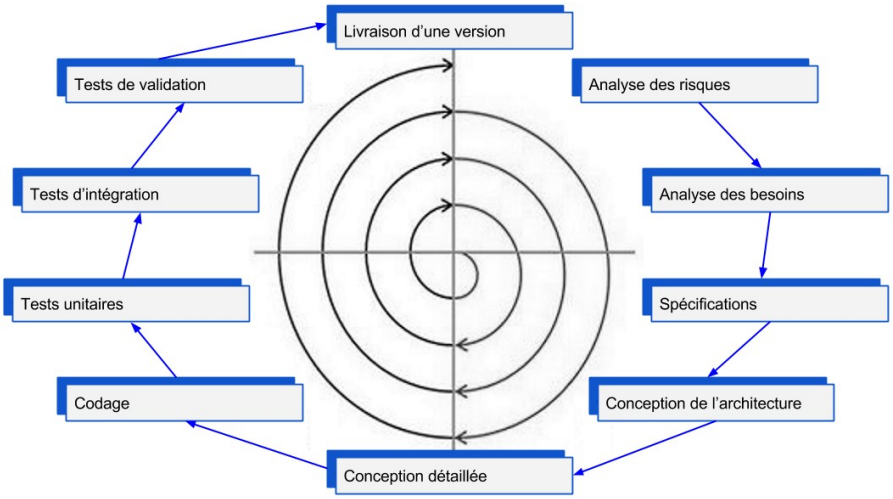
II.3. Les méthodes itératives ou incrémentales

2.3.1. Présentation

Les **méthodes itératives** ou **incrémentales** sont basées sur des **itérations** ou **incréments**. Chaque **itération** comportera de la conception, de la fabrication et du test. Une **itération** se termine par la réalisation d'un **prototype**.

2.3.2. Cycle en Spirale

Le **cycle en spirale** est basé sur des **itérations** successives se terminant par une livraison (interne ou externe) :

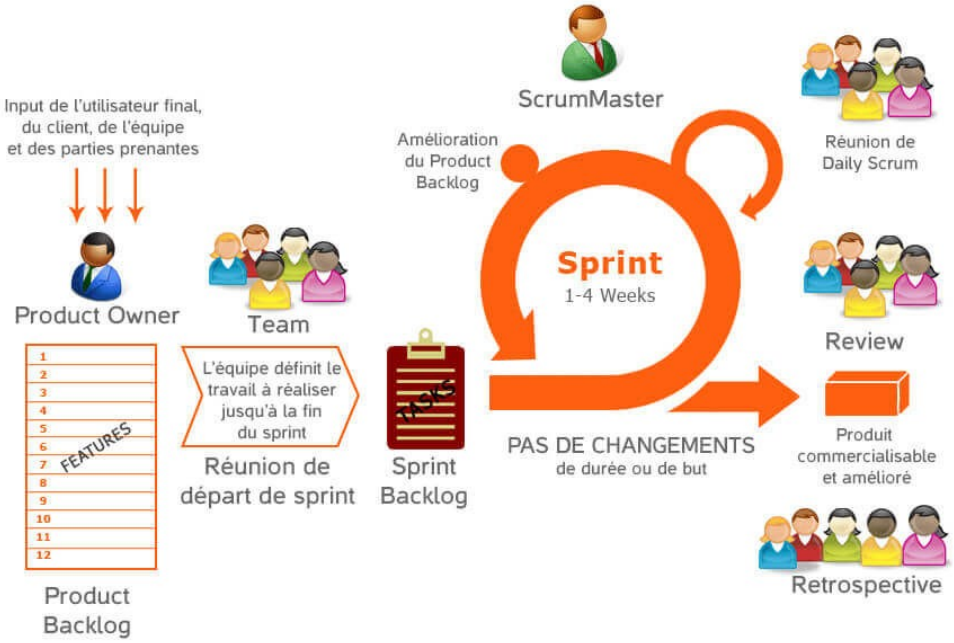


Dans le cas de nos projets de **BTS**, nous utiliserons ce cycle de développement et nous réaliserons **2 itérations** permettant de réaliser **2 prototypes** : **Prototype 0** (fonctionnalités réduites) et **Prototype 1** (fonctionnalités complètes).

2.3.1. Les méthodes Agile

Le mouvement des méthodes **Agile** a commencé en **2001** aux États-Unis suite à un taux d'échec important des projets observés dans les années 90. La méthode **Agile** principale se nomme **Scrum** :

Le **Scrum** ou « **mêlée** », est un terme emprunté au rugby qui désigne la solidarité et la force qui lient les membres de l'équipe au succès de l'itération. Le cycle de vie de **Scrum** est rythmé par des **itérations** de deux à quatre semaines qu'on appelle sprints :



L'**organisation du projet** est guidée par la satisfaction du besoin client, on va donc commencer par créer la liste priorisée (ordonnée) de tout ce qui va être fait par l'équipe pour répondre au besoin client. Cette liste, composée d'éléments, est appelée le **product backlog**.

Avant chaque **sprint**, on effectue une réunion de planification appelée le **sprint planning meeting** qui consiste à sélectionner les exigences prioritaires pour le client dans le produit backlog qui seront développées, testées et livrées au client : le **sprint backlog** (sous-ensemble du **product backlog**).

Des **mêlées** sont organisées quotidiennement durant le **sprint** afin de contrôler l'avancement pour s'assurer que les objectifs sont tenus. A la fin de chaque **sprint**, l'équipe tient une revue de sprint (**Sprint review**) pour faire le bilan de ce qui s'est passé pendant le sprint et en tirer des améliorations pour le suivant.

Dans l'idéal à l'issue de chaque **sprint** on devrait obtenir un, **Incrément du Produit** potentiellement **Livrable**. c'est-à-dire une **logiciel fonctionnel**, **testé** et **documenté**.

III - Les tests logiciel

III.1. Introduction

En informatique, un **test** désigne une **procédure de vérification** partielle d'un système. Les **tests** visent ainsi à vérifier que le système réagit de la façon prévue par ses développeurs (spécifications) ou est conforme aux **besoins du client**.

Type de test	Rôle	Réalisé par
Test unitaire	Tester les modules ou classes au fur et à mesure de leur développement	Le développeur du module
Test d'intégration	Tester l'assemblage des modules, des composants et de leurs interfaces	Le développeur du module ou un autre développeur
Test de système	Tester les fonctionnalités du système dans son ensemble	Les développeurs
Test d'acceptation	Confirmer que le système est prêt à être livré et installer	Le client

Afin de réaliser les **tests unitaires** de nos classes **C++**, on utilisera le framework **CppUnit** détaillé en **Annexe 1**.

III.2. Cahier de recette

Afin de valider le bon fonctionnement du projet réalisé, un **cahier de recette** est rédigé dans lequel sont consignés les **scénarios de tests** avec leurs résultats. A la remise du projet au client, un procès **verbal de recette** est signé entre les 2 entités (cf. **Annexe 2**).

Remarque : Si on dispose de la modélisation **SysML** du projet, chaque **campagne de tests** (**CT**) correspond à un **cas d'utilisation** et chaque **scénario de tests** (**ST**) correspond au **diagramme de séquences système** du cas d'utilisation concerné.

IV - Les générateurs de documentation

Un **générateur de documentation** est un outil de programmation qui crée de la documentation destinée aux programmeurs et aux utilisateurs finaux. Pour gérer ces documentations, le générateur se base généralement sur des **codes sources commentés**.

Les générateurs de documentation tels **Doxygen** ou **Javadoc** génèrent automatiquement la documentation à partir du code source. Ils extraient le commentaire du code source et créent des manuels de référence sous des formats **texte**, **HTML**, **PDF**, **DocBook**, ou **RTF**.

Pour commenter une application en vue de la génération de sa documentation avec **Doxygen**, il faut utiliser des **tags JavaDoc** présentés en **Annexe 3**.
Un **tag** commence toujours par @ ou \.

V - La gestion des versions

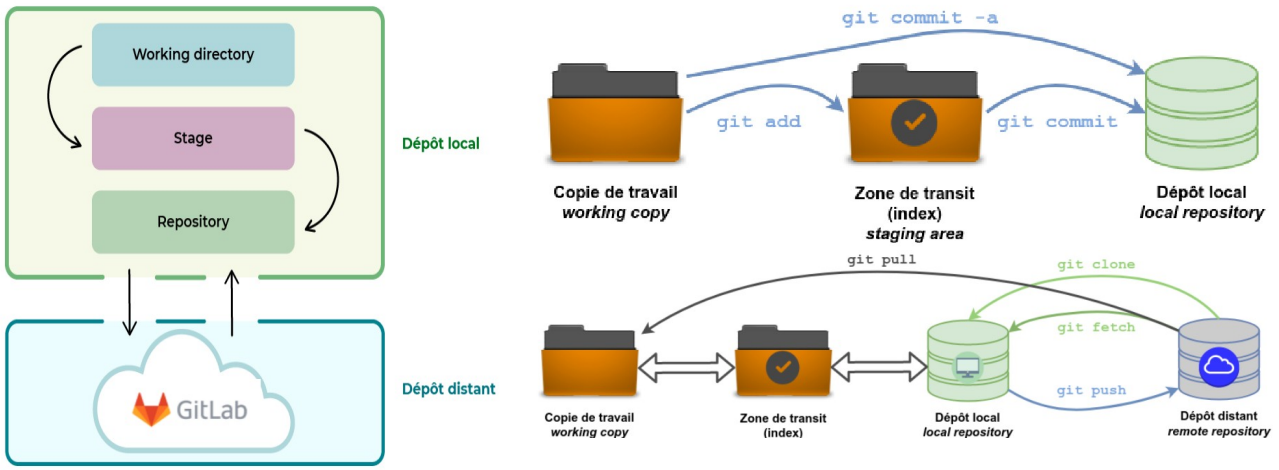
V.1. Introduction

La **gestion de versions** (en anglais ***Version Control System***) consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers. Essentiellement utilisée dans le domaine de la création de logiciels, elle concerne surtout **la gestion des codes source**.
Cette activité étant fastidieuse et relativement complexe, un **appui logiciel** est presque indispensable.

V.2. Les logiciels de gestion de versions

Il existe **différents logiciels de gestion de versions** qui, bien qu'ayant des concepts communs, apportent chacun leur propre vocabulaire et leurs propres usages.

Les logiciels les plus utilisés aujourd'hui sont **Git** (associé à **GitLab** ou **GitHub**) ainsi que **Subversion** (**SVN**).
Dans le cas de **Git** associé à **GitLab** (cf. **Annexe 4**), on peut visualiser sur le schéma ci-dessous le passage d'un fichier dans les différentes zones du **dépôt local** et **distant** :



VI - Les logiciels de gestion de projet

Il existe **différents logiciels de gestion de projet** qui intègrent en totalité ou en partie les fonctionnalités précédentes.

Le travail des logiciels de gestion de projet s'occupent principalement de la gestion des ressources, de la planification et de la gestion du temps (ordonnancement de tâches en vue de leur réalisation future).
Nous allons utiliser dans le cadre de nos projets, un logiciel sous licence **GPL** nommé **Redmine** utilisable en mode **Web** (cf. **Annexe 5**).