R5.04 -- Cycle de vie d'un projet informatique
Cours 1 : Gérer un projet en itérations

Halim Djerroud <halim.djerroud@uvsq.fr>

LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

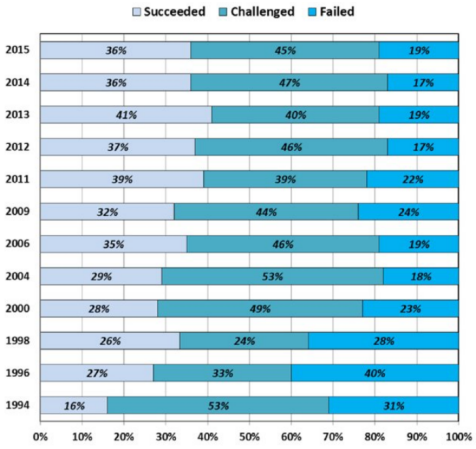
révision : 2.0

Plan

R5.04 -- Cycle de vie d'un projet informatique

- **Cours 1 : Gérer un projet en itérations**
  1. Pourquoi une méthode ?
  2. Cycles de vie : cascade, V, itératif et incrémental
  3. L'agilité et la méthode Scrum
  4. Planifier un sprint : user stories, planning poker, Kanban
  5. Scrum dans la SAÉ
- Cours 2 : Git, forge et intégration continue
- TP (6h) : installer et utiliser sa forge Forgejo

Pourquoi une méthode ?

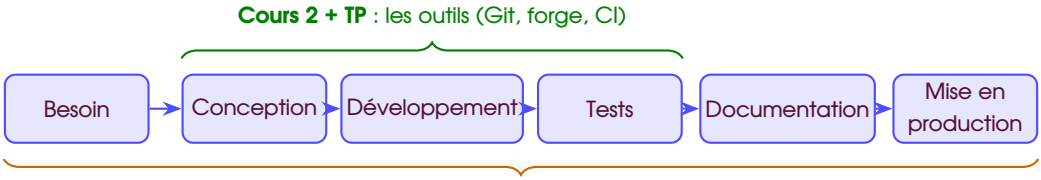


Source : Standish Group, *CHAOS Report*

- Moins de **4 projets informatiques sur 10** aboutissent dans les délais, le budget et avec les fonctionnalités prévues.
- Causes fréquentes :
  - le client ne sait pas exactement ce qu'il veut ;
  - mauvaise communication client / équipe ;
  - les besoins changent en cours de projet ;
  - délais irréalistes, travail mal réparti.

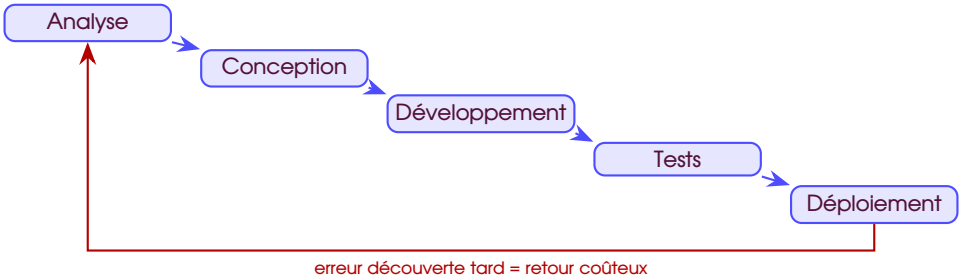
Une méthode sert à **organiser le travail** de l'équipe et à **s'adapter** quand le projet évolue.

Le cycle de vie d'un projet informatique



Toutes ces étapes existent dans tout projet. La question est : **dans quel ordre, et combien de fois ?**

Cycle de vie linéaire : la cascade



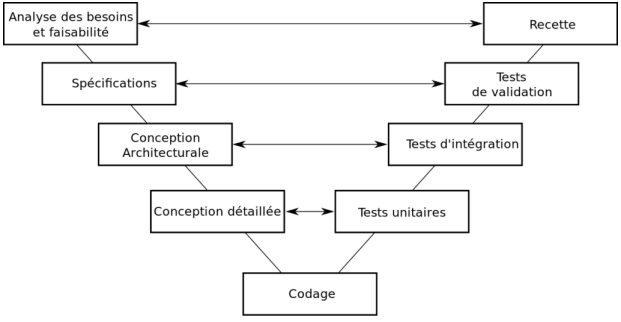
Principe

- Étapes **séquentielles**, chacune **validée** avant la suivante.

Limites

- Produit visible **à la fin**, **rigide** face aux changements.

Cycle de vie linéaire : le cycle en V



Principe

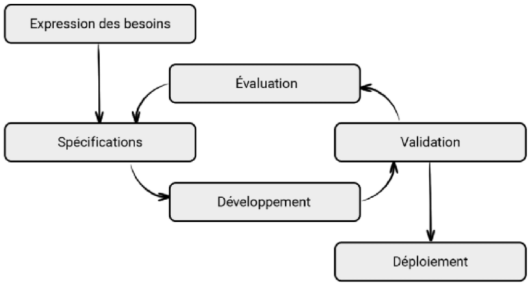
- Chaque étape de conception est associée à une **étape de test**.
- Descente : **définir** le système.
- Remontée : **vérifier et valider**.

Limites

- Mêmes défauts que la cascade.
- Le client ne voit le produit qu'à la **recette**.

LISV Laboratoire d'Ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Cycle de vie itératif et incrémental

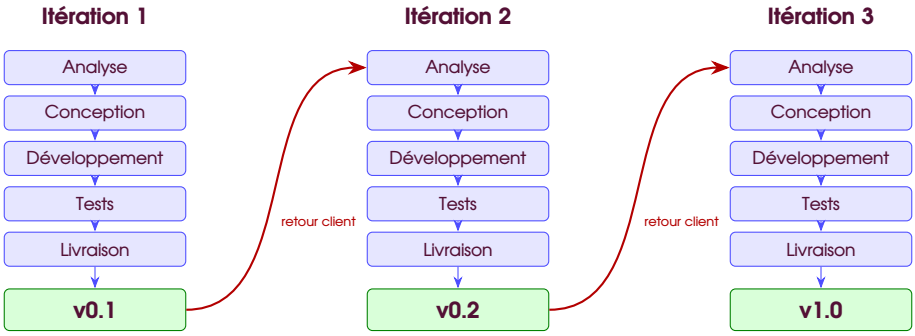


- **Itératif** : on répète le cycle plusieurs fois, en courtes itérations.
- **Incrémental** : chaque itération ajoute des fonctionnalités **utilisables**.
- Le client voit le produit **très tôt** et peut corriger le tir.

Limites

- Demande une planification rigoureuse.
- Un mauvais découpage nuit à tout le projet.

Dans chaque itération : une mini-cascade



On découpe le projet en **petites itérations**. Dans ce cours, chaque itération suit une **cascade** et se termine par une **version livrable** (un incrément).

Les méthodes agiles

Les méthodes agiles sont des méthodes **itératives et incrémentales** qui privilégient :

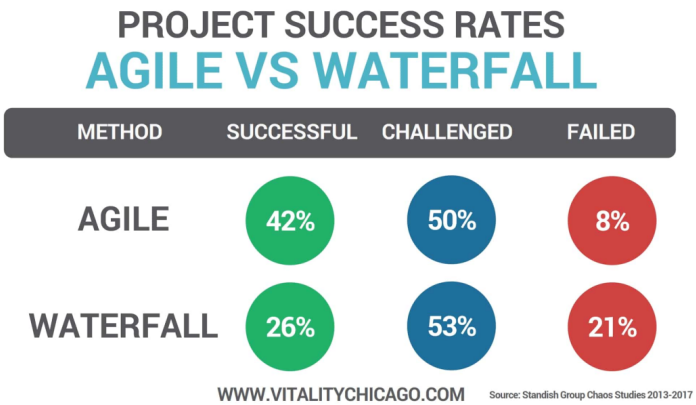
- la livraison **rapide et fréquente** de versions fonctionnelles ;
- la **collaboration** avec le client ;
- l'**adaptation au changement**.

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.

Exemples : **Scrum**, Kanban, Extreme Programming (XP), Lean. . .

Est-ce que ça marche ?



La méthode Scrum

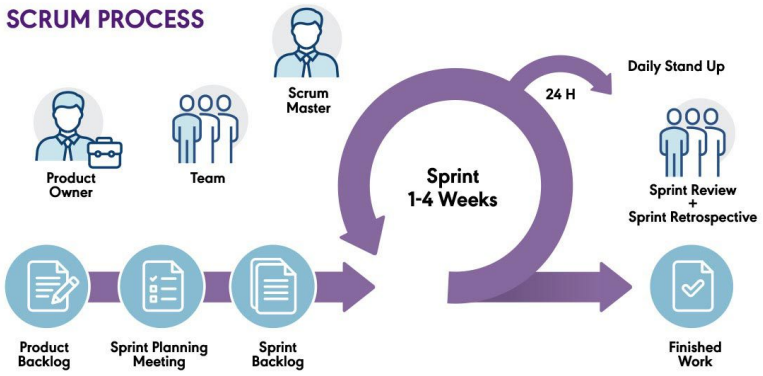
- Méthode agile la plus utilisée en entreprise.
- Le projet avance par **sprints** : itérations de durée fixe (**1 à 4 semaines**).
- Chaque sprint produit un **incrément** fonctionnel et livrable.
- L'équipe est **auto-organisée** : c'est elle qui découpe et répartit le travail.

Scrum repose sur :

- **3 rôles** : Product Owner, Scrum Master, Développeurs ;
- **3 artefacts** : Product Backlog, Sprint Backlog, Incrément ;
- **5 événements** : le Sprint, qui contient Sprint Planning, Daily Scrum, Sprint Review et Rétrospective.

Référence : *The Scrum Guide* (K. Schwaber, J. Sutherland, 2020), `https://scrumguides.org`

Scrum : vue d'ensemble



Les 3 rôles

Product Owner

- Porte la **voix du client**.
- Écrit et **priorise** le Product Backlog.
- Accepte ou refuse ce qui est livré.

Scrum Master

- **Garant de la méthode**.
- Anime les événements.
- **Lève les obstacles** qui bloquent l'équipe.
- N'est pas un chef de projet !

Développeurs

- Équipe de 3 à 10 personnes.
- **Estiment**, découpent et réalisent le travail.
- Responsables **ensemble** de l'incrément.

Les artefacts

- **Product Backlog** : liste **priorisée** de tout ce qu'il reste à faire sur le produit (user stories). Il évolue en permanence.
- **Sprint Backlog** : les stories choisies pour le sprint en cours, **découpées en tâches**, avec un **objectif de sprint**.
- **Incrément** : la version du produit obtenue à la fin du sprint, qui respecte la **Definition of Done**.

Les événements d'un sprint

Événement	Rôle	Durée max.
Sprint Planning	Choisir les stories du sprint, fixer l'objectif, découper en tâches.	8h / mois de sprint
Daily Scrum	Chaque jour, debout : *qu'ai-je fait hier ? que vais-je faire ? qu'est-ce qui me bloque ?*	15 min
Sprint Review	Montrer l'incrément au Product Owner / client et recueillir ses retours.	4h / mois de sprint
Rétrospective	L'équipe améliore sa **façon de travailler** : ce qui a marché, ce qu'on change.	3h / mois de sprint

Les durées sont **fixes** (*time-box*) : on ne rallonge pas un sprint, on réduit son contenu.

User story

Une **user story** décrit une fonctionnalité **du point de vue de l'utilisateur** :

En tant que *<rôle>*, **je veux** *<action>* **afin de** *<bénéfice>*.

Exemple (porte-monnaie électronique) :

- *En tant que* ***commerçant****, je veux* ***débiter la carte*** *d'un client afin d'* ***encaisser un achat****.*

Critères d'acceptation :

- le débit est refusé si le solde est insuffisant ;
- chaque débit est enregistré dans l'historique des transactions.

Epic : une story trop grosse, à redécouper (ex. *« gérer les cartes »*).

Une bonne user story : INVEST

- **I**ndépendante des autres stories
- **N**égociable avec le Product Owner
- **V**aleur pour l'utilisateur
- **E**stimable par l'équipe
- **S**mall : tient dans un sprint
- **T**estable : critères d'acceptation clairs

Mauvaise story

« Faire la base de données »

→ pas de valeur visible pour l'utilisateur, pas testable.

Meilleure story

« En tant qu'administrateur, je veux consulter l'historique d'une carte afin de traiter une réclamation. »

Estimer : le planning poker

- On estime la **complexité relative** des stories en **points** (pas en heures).
- Chaque développeur a un jeu de cartes (suite de Fibonacci modifiée) :

0	$\frac{1}{2}$	**1**	**2**	**3**	**5**	**8**	**13**	**20**	**40**	**100**	**?**

Déroulé pour chaque story :

1. Le Product Owner présente la story, l'équipe pose ses questions.
2. Chacun choisit une carte **en secret**.
3. Tout le monde **révèle en même temps**.
4. Les estimations extrêmes (la plus basse, la plus haute) **s'expliquent**.
5. On revote jusqu'à un **consensus**.

Pourquoi le planning poker ?

- Vote **simultané** : personne n'est influencé par l'avis du plus expérimenté.
- Les écarts révèlent les **malentendus** : « tu as oublié qu'il faut aussi gérer le refus ! »
- Les **points sont relatifs** : une story à 8 est environ 4 fois plus complexe qu'une story à 2.
- Une story à **40 ou 100** est trop grosse : c'est une epic, il faut la découper.
- La carte **?** signifie : « je n'ai pas assez d'informations ».

Vélocité

Nombre de points **terminés** par l'équipe en un sprint.
Elle sert à choisir combien de stories prendre au sprint suivant.

À vous de jouer : planning poker

Estimez ces stories du **porte-monnaie électronique** (5 min) :

#	User story	Points
1	En tant que **client**, je veux **consulter le solde** de ma carte afin de savoir ce que je peux dépenser.	
2	En tant que **commerçant**, je veux **débiter la carte** d'un client afin d'encaisser un achat.	
3	En tant qu'**administrateur**, je veux **personnaliser une carte vierge** (titulaire, numéro) afin de la remettre à un client.	
4	En tant qu'**administrateur**, je veux **consulter l'historique** des transactions d'une carte afin de traiter une réclamation.	

Règle : la story 1 vaut **2 points**. Estimez les autres **par rapport à elle**.

Le Sprint Planning

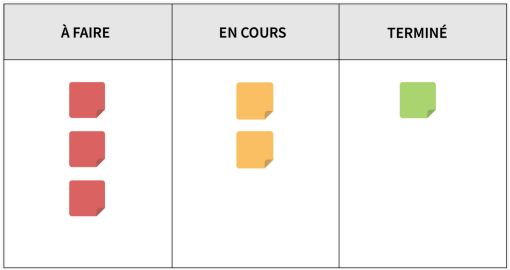
1. Le Product Owner présente l'**objectif** et les stories les plus prioritaires.
2. L'équipe **estime** les stories (planning poker).
3. L'équipe **sélectionne** ce qu'elle peut faire, selon sa vélocité.
4. Chaque story est **découpée en tâches** techniques.
5. Les tâches sont placées dans le **Kanban**, colonne « À faire ».
6. Chaque développeur **choisit** ses tâches.

Découpage de la story 2

Débiter la carte d'un client

- Commande de débit sur la carte
- Vérifier le solde suffisant
- Enregistrer la transaction en base
- Écran de saisie du montant
- Tests du débit

Suivre le sprint : le tableau Kanban



- Une **carte** = une tâche.
- Une **colonne** = un état.
- Les cartes avancent de gauche à droite.
- On **limite le travail en cours** (WIP) : on termine avant de commencer autre chose.
- Tout le monde voit **qui fait quoi** et **ce qui bloque**.

Kanban : exemple en plein sprint

À faire	En cours	En revue	Terminé
Écran de saisie du montant	Vérifier le solde suffisant *Alice*	Commande de débit sur la carte *Léa*	Lire le solde de la carte
Tests du débit	Enregistrer la transaction *Karim*		Afficher le solde

Dans le TP, ce tableau sera construit dans la **forge Forgejo** : une tâche = un **ticket** (issue).

Quand une tâche est-elle « terminée » ?

Definition of Done (DoD)

Liste de critères, **fixée par l'équipe**, que toute tâche doit respecter pour passer dans « Terminé ».

Exemple de DoD pour la SAÉ :

- le code est sur une **branche** dédiée à la tâche ;
- il a été **relu** par un autre membre de l'équipe ;
- les **tests passent** ;
- il est **fusionné** dans la branche principale ;
- le ticket est **fermé**.

« Ça marche sur ma machine » **n'est pas** terminé.

Scrum dans la SAÉ

Ce qu'on applique

- Backlog de user stories
- Sprints avec objectif et incrément livrable
- Sprint Planning avec planning poker
- Tableau Kanban à jour
- Definition of Done
- Review et rétrospective en fin de sprint

Ce qu'on adapte

- **Durée des sprints** : calée sur le calendrier des séances de SAÉ.
- **Daily** : un point de 5 min en début de séance.
- **Product Owner** : l'enseignant joue le client.
- **Scrum Master** : un étudiant de l'équipe, rôle tournant.

Tout le suivi (backlog, tâches, sprints, code) se fera dans **votre forge** installée en TP.

À retenir

- On découpe le projet en **petites itérations**, chacune livre un **incrément**.
- **Scrum** : 3 rôles, 3 artefacts, des événements à durée fixe.
- Les besoins s'écrivent en **user stories**, estimées au **planning poker**.
- Au Sprint Planning, les stories sont **découpées en tâches** et suivies dans un **Kanban**.
- Une tâche n'est terminée que si elle respecte la **Definition of Done**.

Prochain cours

Comment l'équipe travaille **à plusieurs sur le même code** : Git, pull requests, forge et intégration continue.