Gestion de projet et démarche Agile

Johann Bourcier < johann.bourcier@irisa.fr >

Software Development life cycle - Definition

- In software engineering, a software development process is a process of dividing software development work into smaller, parallel, or sequential steps or sub-processes to improve design, product management.
- It is also known as a software development life cycle (SDLC). The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.

Software Development life cycle - motivation

Software Development life cycle - motivation

- It provides an **effective framework** and **method** to develop software applications.
- It helps in effectively **planning** before starting the actual **development**.
- SDLC allows **developers** to analyze the requirements.
- It helps in **reducing** unnecessary costs during development.

Software Development Life-cycle – Activities?

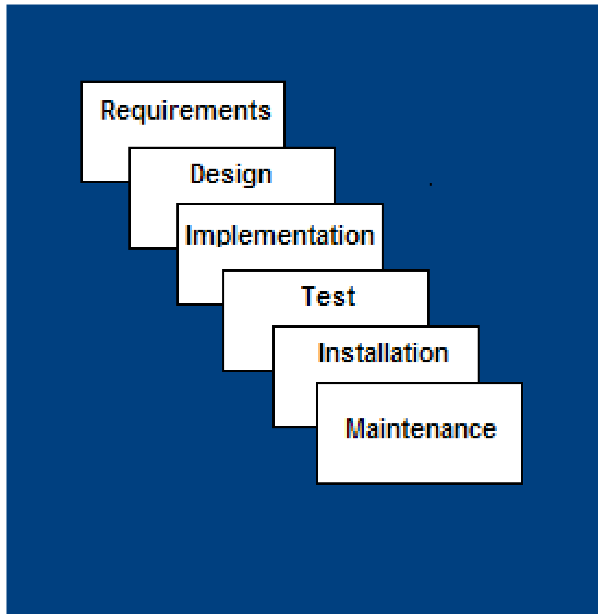
Software Development Life-cycle – Activities?

- Analysis and Planning
- Requirements
- Design and Prototyping
- Software Development
- Testing
- Deployment
- Maintenance and Updates

History

- 1st documented method SDM in the 1970's
- Evolved constantly with new methodolgy or new version :
  - 1970s
    - Structured programming since 1969
    - Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974. SDM stands for System Development Methodology
  - 1980s
    - Structured systems analysis and design method (SSADM) from 1980 onwards
    - Information Requirement Analysis/Soft systems methodology
  - 1990s
    - Object-oriented programming (OOP) developed in the early 1960s and became a dominant programming approach during the mid-1990s
    - Rapid application development (RAD), since 1991
    - Dynamic systems development method (DSDM), since 1994
    - Scrum, since 1995
    - Team software process, since 1998
    - Rational Unified Process (RUP), maintained by IBM since 1998
    - Extreme programming, since 1999
  - 2000s
    - Agile Unified Process (AUP) maintained since 2005 by Scott Ambler
    - Disciplined agile delivery (DAD) Supersedes AUP
  - 2010s
    - Scaled Agile Framework (SAFe)
    - Large-Scale Scrum (LeSS)
    - DevOps

Waterfall Model



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation, testing.
- It is plan-driven

https://www.youtube.com/watch?v=5A5XCuWMG4o

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

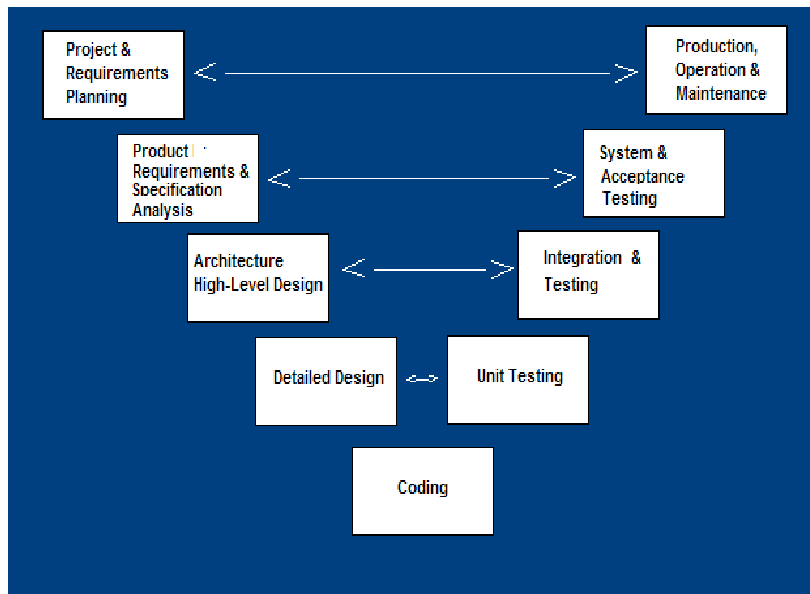
Waterfall Deficiencies

- All requirements must be known at beginning
- Can give a false impression of progress
- Does not reflect problem-solving of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use the Waterfall Model

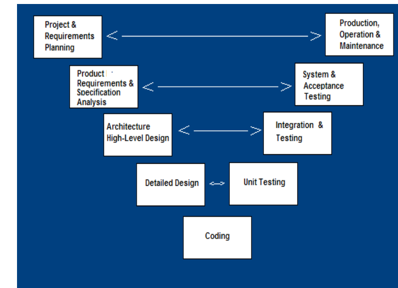
- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Carrying an existing product to a new platform (i.e. diff language).

V-Shaped SDLC Model



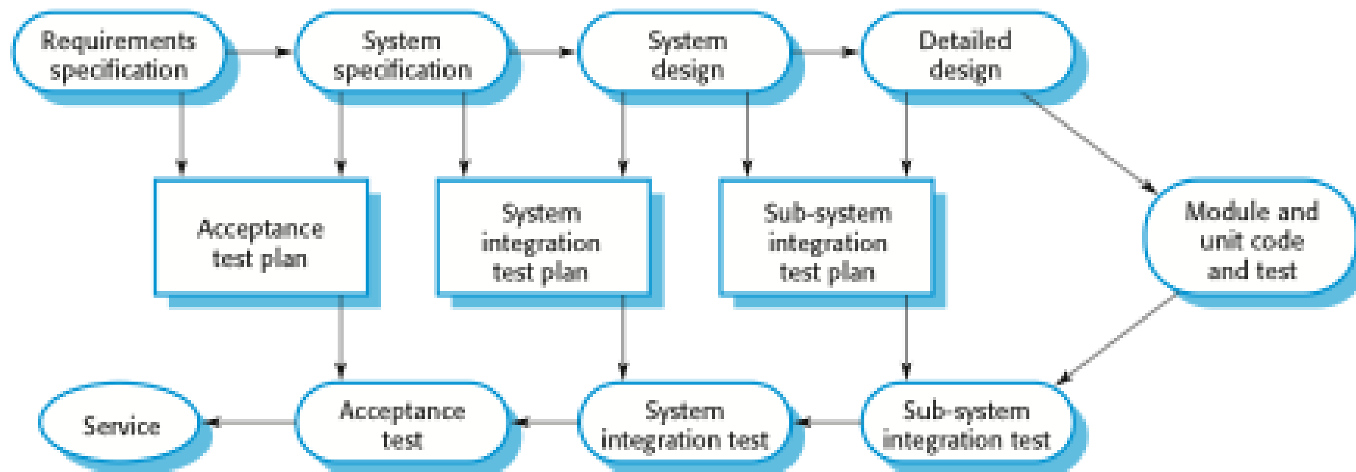
- It is plan-driven
- A modified of the Waterfall model that highlights the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps



- **Project and Requirements Planning** – allocate resources
- **Product Requirements and Specification Analysis** – complete specification of the software system
- **Architecture or High-Level Design** – defines how software functions fulfill the design
- **Detailed Design** – develop algorithms for each architectural component
- **Production, operation and maintenance** – provide for enhancement and corrections
- **System and acceptance testing** – check the entire software system in its environment (client environment)
- **Integration and Testing –** check that modules interconnect correctly
- **Unit testing –** check that each module acts as expected
- **Coding –** transform algorithms into software

Testing phases in a plan-driven software process



Seven Testing Principles: https://www.youtube.com/watch?v=rFaWOw8bIMM
How to write a Test Case: https://www.youtube.com/watch?v=BBmA5Qp6Ghk
Unit Testing Example: https://www.youtube.com/watch?v=Ij5nnGa_DIw
Integration Testing Example: https://www.youtube.com/watch?v=QYCaaNz8emY
System Testing & Acceptance Testing: https://www.youtube.com/watch?v=N8-qNMHOVyw
Top 5 Software Testing Interview Questions: https://www.youtube.com/watch?v=Vb0_bY1tHfQ

V-Shaped Strengths

- Stress planning for **verification and validation** of the product in early stages of product development
- Why testing is important?
- **Each deliverable must be testable**
- Project management can **track progress by milestones…**
- **Easy to use**

V-Shaped Weaknesses

- Does not easily handle **concurrent events/tasks, because tasks are created plan-driven…**
- Does not handle **iterations** or phases
- Does not easily handle **dynamic changes in requirements (diffucult to change plans in any time!!!)**
- Does not contain **risk analysis** activities

When to use the V-Shaped Model

- Excellent choice for **systems requiring high reliability** – e.g. hospital patient control applications
- **All requirements are known** clearly
- **If solution and technology are known well**

Some big software Fails

- https://en.wikipedia.org/wiki/List_of_failed_and_overbudget_custom_software_projects

Démarche incrémentale

Pratique ancienne et courante :

- Vise à découper un projet en livraisons successives
- Permet de vérifier l'avancement
- Et d'échelonner les paiements

N'est pas contradictoire avec un cycle en V

- La solution est définie au commencement du projet
- Seule sa réalisation est décomposée

Itératif

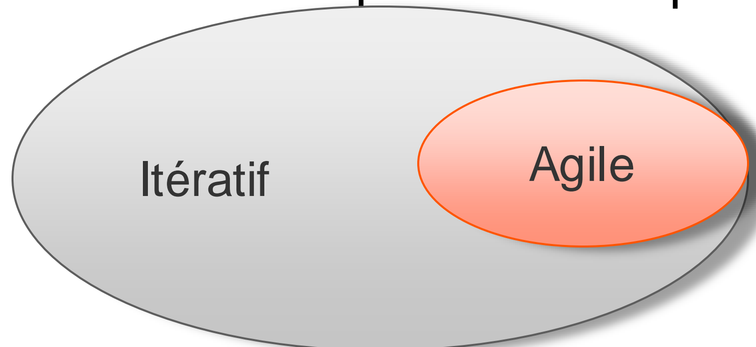
> Contrairement à l'incrémental :
  - On démarre avec une vision produit et une première esquisse
  - On construit le produit au fur et à mesure des itérations

> Le principe clef est d'utiliser la boucle de **rétroaction**
  - Apprendre en développant
  - **Apprendre en UTILISANT** le produit issu de l'itération précédente

On part de l'hypothèse suivante :

=> Il n'est pas possible de spécifier à priori un bon produit

> Elément nécessaire mais pas suffisant pour être agile



Agile

Principes du lean management

Amélioration continue

Respect des personnes

Remettre en cause chaque chose

Adopter le changement

Le focus est mis sur la réalisation de produit utilisable

Le client est au cœur du processus

Le produit est construit par le dialogue avec le client :

- Une équipe, pas un donneur d'ordre et des exécutants

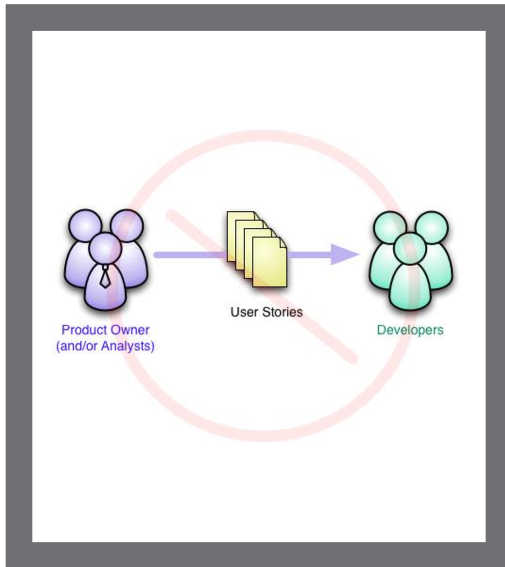
En laissant l'équipe s'organiser

- Choisir ses méthodes et ses outils

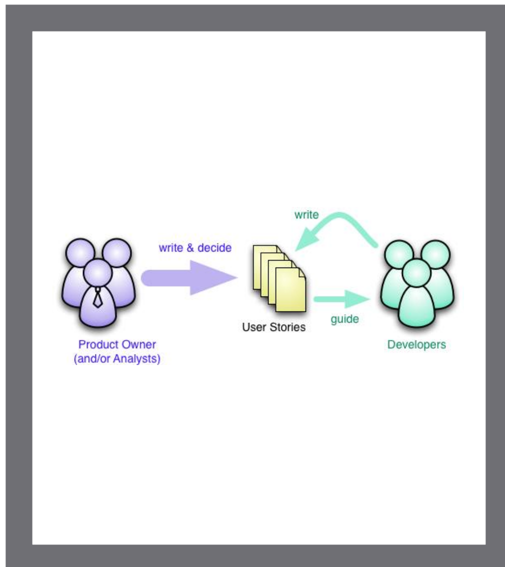
Construire un produit de façon agile

(repris du blog de Martin Fowler) :

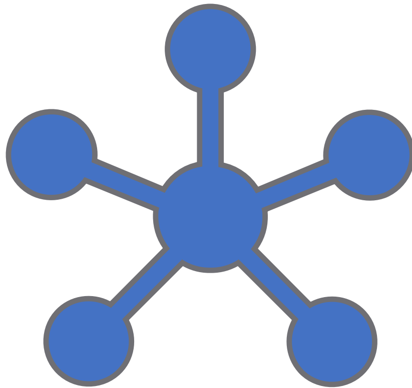
> Pas agile :



> Agile :



Les principes SCRUM (1/2)

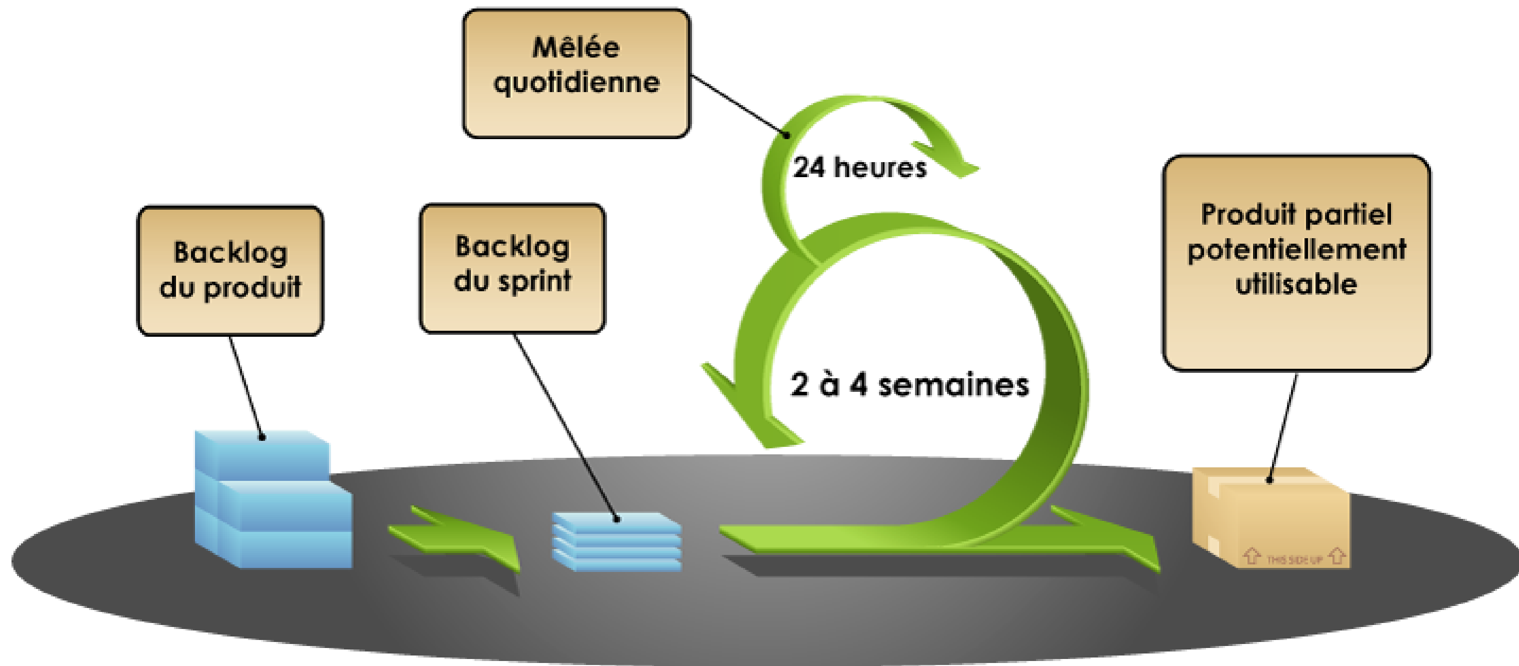


- Scrum est un processus Agile qui vise à produire la plus grande valeur métier dans la durée la plus courte.

- Du logiciel qui fonctionne est produit à chaque itération appelée **sprint** (toutes les 2 à 4 semaines).

- Le client définit les priorités. L'équipe s'organise pour déterminer la meilleure façon de produire les exigences les plus prioritaires.

- A chaque fin de sprint, tout le monde peut voir fonctionner le produit courant et décider ce qui doit être fait dans le prochain sprint ou livrer le produit s'il est jugé satisfaisant par le client.



Principes SCRUM (2/2) : le process

Mise en œuvre

Anticiper les évolutions

Un développement piloté uniquement par les histoires utilisateurs ne permet pas de construire un système évolutif…

Fixer le cadre des développements

Le modèle d'analyse :

- Définit le langage commun avec le client, fixe les notions
- Donne le modèle du problème et son éco système

Le modèle d'architecture décrit:

- Le principe de la solution
- L'organisation statique et dynamique du logiciel
- Les règles respectées lors de la conception
- Les technologies utilisées
- Comment sont prises en compte les contraintes non fonctionnelles

Le sprint 0 (½)

- Un sprint à part, au démarrage du projet
- Permet de définir les bases du produit
- Permet de valider les points techniques durs
- Se concrétise par une première fonctionnalité, même partielle, qui démontre que l'ensemble de la construction tient.
- Permet de faire une première estimation macroscopique du backlog produit

Le sprint 0 (2/2)

Les bases du produit

Le modèle d'analyse

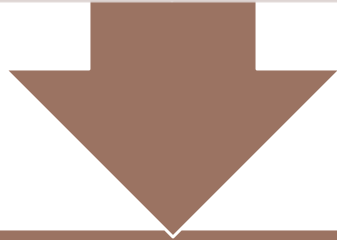
Le modèle d'architecture

Le cadre de production : choisir les outils, se mettre d'accord sur la façon de travailler : gestion de configuration, règles de codage, référence pour les tests

Les sprints suivants

Permettent l'obtention d'une version plus complète du produit

Le résultat obtenu est stable	Il est utilisable par le client



En implémentant de nouvelles histoires utilisateur

En fonction des priorités définies par le client	De façon à lever les incertitudes au plus tôt

Le cycle de vie d'un sprint

- Planification
- Réunion quotidienne
- Revue de sprint
- Rétrospective

Revue de planification de sprint

Objectif :

identifier les cas d'utilisation qui seront implémentés dans le sprint

Se mettre d'accord sur le contenu de la démo de fin de sprint

Eléments d'entrée :

Le product backlog, la liste des cas d'utilisation identifiés pour le produit

La capacité à produire de l'équipe pour le sprint

Eléments en sortie :

Le sprint backlog : les cas d'utilisation qui seront implémentés dans le sprint

La description de la façon dont sera démontré chaque cas d'utilisation

La définition des tâches nécessaires à la réalisation des histoires et leur estimation en heure.

Réunion quotidienne

Objectif :

Recenser ce qui a été fait

Identifier les problèmes rencontrés, ce qui gène (*impediment*)

Définir ce qui sera fait dans la journée

Eléments d'entrée :

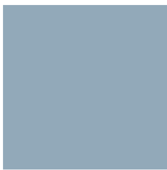
Le sprint backlog,

Un outil de suivi des tâches, de leur affectation et de leur reste à faire

Eléments en sortie :

Le sprint backlog remis à jour : reste à faire, commentaires et précision sur les tâches

Revue de sprint



Objectif :

Valider l'avancement

Analyser le résultat

Définir les actions pour la suite

Eléments d'entrée :

Le produit issu du sprint

Eléments en sortie :

Le product backlog remis à jour avec de nouveaux items ou des changements de priorité

Retrospective

Objectif :

Identifier les bonnes pratiques à poursuivre

Identifier ce qui marche mal et qu'il faut abandonner

Définir les améliorations à apporter

Eléments d'entrée :

L'histoire du sprint (*impediments*)

Eléments en sortie :

Une ou deux résolutions pour le sprint suivant

Comment concevoir et se partager le travail

- Travailler à partir de réunion de co-design
  - Pour choisir les principes de solution
- Maintenir les modèle globaux à jour
  - Chacun enrichit au fur et à mesure des travaux
  - Consigner les raisons des choix techniques
- Partager le travail par fonctionnalité plutôt que par domaine technique
  - Permet la propriété collective du code : chacun est responsable

Comment définir et estimer les tâches

Cherchez l'ordre de grandeur, pas la valeur exacte

Par exemple : 1, 2, 4, …

Définissez un Critère de fin de tâche et tenez en compte au moment de l'estimation

Pensez aux tests et vérifiez qu'ils sont reproductibles

Définissez bien la référence pour éviter le « chez moi, ça marche »

Avoir une idée du design est nécessaire pour estimer

Eviter de calquer les tâches sur le modèle de design, choisissez plutôt quelque chose de testable

Backlog

- Ensemble d'histoire utilisateurs qui bout à bout compose l'ensemble des fonctionnalités du logiciel

User story

- *une user story est une explication non formelle, générale d'une fonctionnalité logicielle écrite du point de vue de l'utilisateur final.*
- *Son but est d'expliquer comment une fonctionnalité logicielle apportera de la valeur au client.*

User story

- Objectif :
  - Donner l'importance aux utilisateurs dans la définition du besoin
  - Donne du contexte et du sens aux développeurs pour le développement d'une fonctionnalité

  - Les user stories désignent une série de phrases, rédigées dans un langage simple, qui décrivent le résultat souhaité. Elles n'entrent pas dans le détail.

User Story

- Template :
  - En tant que : pour qui et quel est son rôle parmi l'ensemble des utilisateurs
  - Je souhaite : décrit l'intention (et non les fonctionnalités ou la manière de le réaliser)
  - Afin de : objectif global (comment le désir immédiat de l'utilisateur de faire quelque chose s'intègre-t-il à la vue d'ensemble ?)

Références

- http://www.12manage.com/methods_demingcycle.html
- http://en.wikipedia.org/wiki/Spiral_model
- www.mountaingoatsoftware.com/scrum
- www.scrumalliance.org
- www.controlchaos.com
- http://danube.com/scrumworks/basic
- http://www.infoq.com/minibooks/scrum-xp-from-the-trenches

Lexique

User Story / Histoire utilisateur

- Décrit une fonctionnalité à développer en se plaçant du point de vue de l'utilisateur
- En tant qu'utilisateur je veux pouvoir réaliser telle action sur le système.

Backlog : ensemble d'histoires utilisateur

- Un pour le produit, global et un pour chaque sprint

Sprint : itération de développement

Burn Down chart : courbe de reste à faire.

- Une pour le produit et une pour chaque sprint

Game Time

- Vous êtes une équipe.
- Entre chaque équipier les balles prennent l'air. (C'est à dire que l'on ne peut pas se passer la balle de main en main sans qu'à un moment la balle soit seule dans l'air).
- Pas de passe à votre voisin direct. (Pas de passe à votre voisin le plus proche c'est simple non ?).
- Le point de départ est le point d'arrivée (La personne qui plonge la main dans le sac pour attraper une balle est aussi celle qui place les balles qui ont parcouru le système dans un autre sac).
- Tous les membres de l'équipe doivent toucher la balle une seule fois sauf le point de départ qui est aussi le point d'arrivée. (Et une seule seulement ! ce qui ne veut pas dire une seule à la fois…).
- Une balle qui tombe, touche le sol, ou qui ne respecte pas ces règles est perdue.
- Itération = 2mn, introspection = 1mn30 & estimation = 30s (voir moins, c'est plutôt un buffer).
- 5 itérations

Explanation between different software development process

- https://www.youtube.com/watch?v=ld8QaVM8zJE&list=PLAwxTw4SYaPkNAtqsKcFkUGpf4j67NBaz&index=10