

Agile Project management

- » 17-24-02 Novembre-2018. Boughdiri Aymen
- » <http://blog.octo.com/agile-game-france-2014/>



- Aymen est un coach agile, manager avec plus de 6 ans d'expériences dans la gestion des équipes de test et de développement dans le domaine financier et travail collaboratif.
- Aymen a plus de 10 ans d'expérience dans le domaine IT
- Il fournit des formations, des services de conseil, d'accompagnement et de mentorat en développement et de test logiciels, notamment en méthodologies agiles, tests fonctionnels, gestion des tests, conception de tests et améliorations de la qualité logicielle.
- Il a l'expérience dans la mise e place des méthodologie agile, l'établissement et l'amélioration des processus de développement et de test
- Aymen a une maîtrise en Finance et un master en ingénierie financière





Aymen Boughdiri

is awarded the designation Certified ScrumMaster® on
this day, February 23, 2015, for completing the
prescribed requirements for this certification and is
hereby entitled to all privileges and benefits offered by
SCRUM ALLIANCE®.



Member: 000394856 Certification Expires: 23 February 2017

Michel Goldenberg

Certified Scrum Trainer®

Harvey Wheaton

Chairman of the Board

Agenda

Histoire

Principes Lean

Manifeste agile

Scrum

Excercise on Scrum

Summary



History

Ford Motors and industrialization



Histoire Henry Ford

On est au début du 20 ème siècle, après l'industrialisation. Henry Ford construit une usine pour faire des voitures.

Durant l'industrialisation la ligne de production fut développée, c'est la production de masse. Les produits ne se font plus à la mains mais à l'aide des machines. H F a réussi à remplacer 85% des ouvriers par des machines.

H. F. a divisé la ligne de production en petits compartiments, en sections. Chaque section optimisait son travail pour trouver la meilleure façon de travailler.

H Ford a réussi à doubler le salaire des ouvriers et réduire le temps de finir une voiture de 12 heures à 1,5 heure.

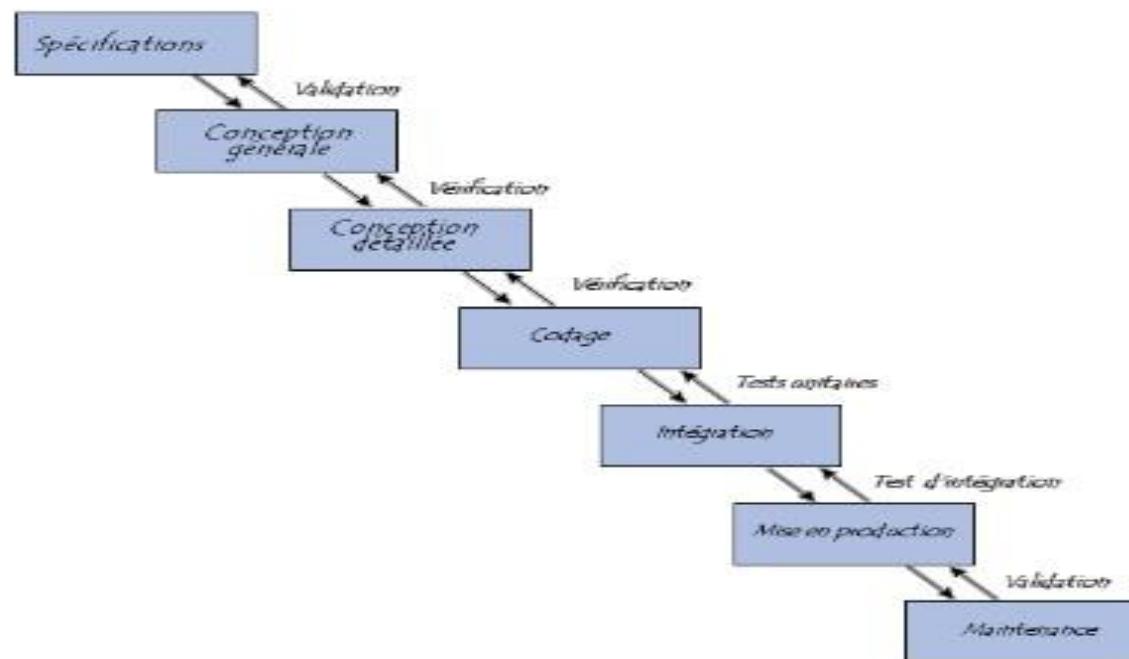
Les ouvriers apprenaient leur travail en 10 minutes → Facile à changer les ouvriers.

Ford avait un grand succès avec la modèle T, appelé T-Ford. La meilleure voiture au monde à ce temps.

Waterfall

Cycle de développement en cascade

Modèle en cascade (Waterfall)



Spécification

- Le but de cette première phase est de spécifier les besoins du logiciel à partir des besoins existants
- Les besoins peuvent se traduire sous plusieurs formes:
 - spécifications générales : un ensemble d'objectifs, de contraintes et de généralités qu'il faudra respecter au cours du développement (Besoins d'affaires)
 - spécifications fonctionnelles : description des fonctionnalités du logiciel de manière aussi détaillée que nécessaire
 - spécifications d'interface : description des interfaces du logiciel avec le monde extérieur de manière aussi détaillée que nécessaire

La conception générale

- La phase de conception générale permet d'envisager plusieurs solutions au problème posé et d'en étudier leur faisabilité.
- On peut faire des prototypes des différentes approches envisagées dans la conception générale afin de les valider.
- La structure d'un logiciel peut être vue sous deux points de vue différents:
 - Le point de vue statique consiste à découper le logiciel en modules si l'on utilise un langage classique ou de définir les objets du système si l'on utilise un langage orienté objet.
 - Le point de vue dynamique consiste à découper le logiciel en tâches pouvant s'exécuter en parallèle ou séquence

La conception détaillée

- Descriptif détaillé des phases informatisées : fiche descriptive de phase de traitement.
- Conception des interfaces utilisateurs des phases conversationnelles :
 - description des écrans de saisie et des écrans d'affichage
 - répartition des tâches homme/machine (voir document dialogue homme/machine), règles de contrôle des données.
- Conception des résultats : maquettes des états de sortie (états imprimés).
- Description des traitements : règles de calcul, algorithmes.

Le Codage

- Le codage, tests unitaires et tests des modules
- Les procédures identifiées lors de la phase précédente sont codées et testées individuellement.
- Le produit de cette phase est le code source et les résultats des tests unitaires.
- Ensuite, chaque module est testé individuellement. On vérifie que les services spécifiés par l'interface d'un module sont effectivement rendus par le module.

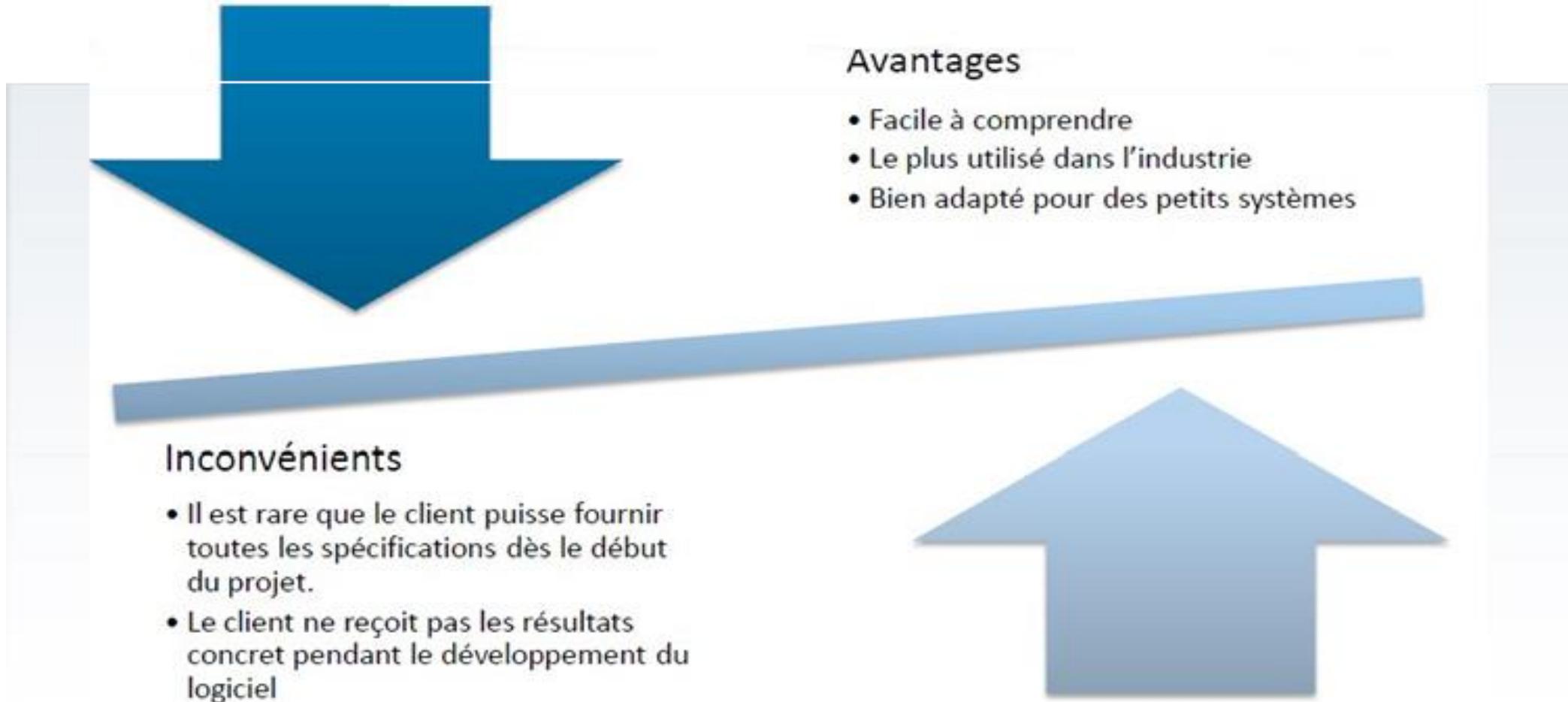
L'intégration du logiciel

- Les différents modules du logiciel sont progressivement intégrés par niveaux successifs en respectant les spécifications des tests d'intégration.
- La phase d'intégration ressemble à une construction où chaque brique de base du logiciel est associée à sa voisine pour former une entité elle-même associée à sa voisine etc... jusqu'à aboutir à la construction toute entière

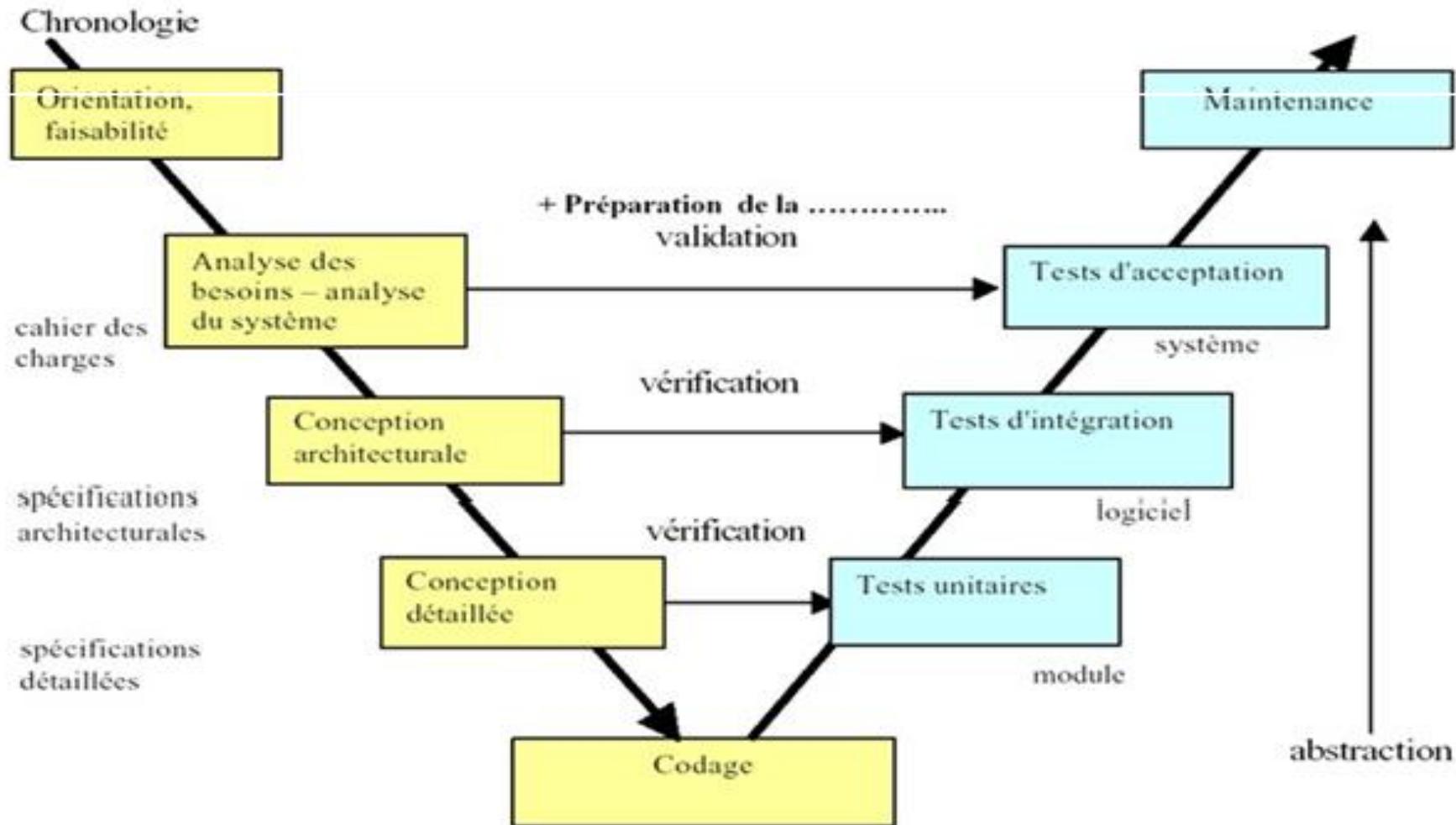
La maintenance

- La maintenance est une activité qui comprend la formation de l'utilisateur et l'assistance technique.
- Elle débute à la livraison du logiciel et s'achève à la fin de l'exploitation du système
- Elle peut inclure aussi des upgrade de système d'exploitation ou la mise en place des nouvelles réglementations

Cycle de vie en cascade: Avantages inconvénients



Cycle de vie en V



Cycle de vie en V

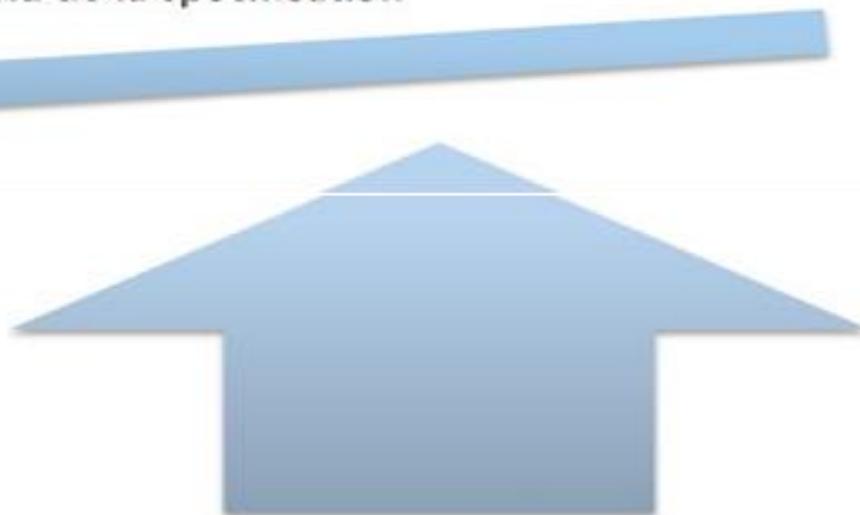


Inconvénients

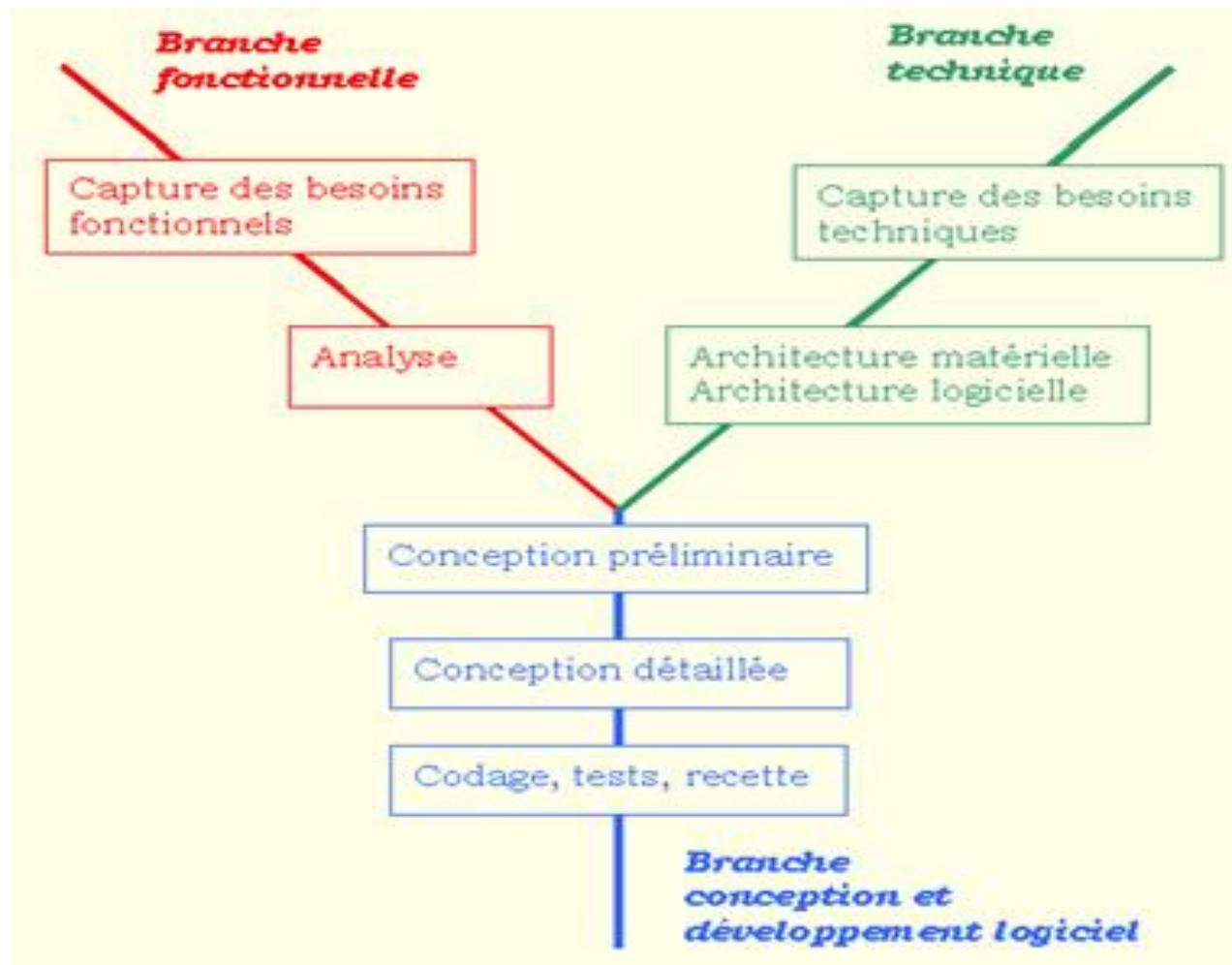
- Pas de résultats intermédiaires dont on peut discuter avec le client
- très coûteux si des erreurs sont constatées

Avantages

- Avec toute décomposition doit être décrite la recomposition, toute description d'un composant est accompagnée de tests
- Principe qui évite un écueil bien connu de la spécification

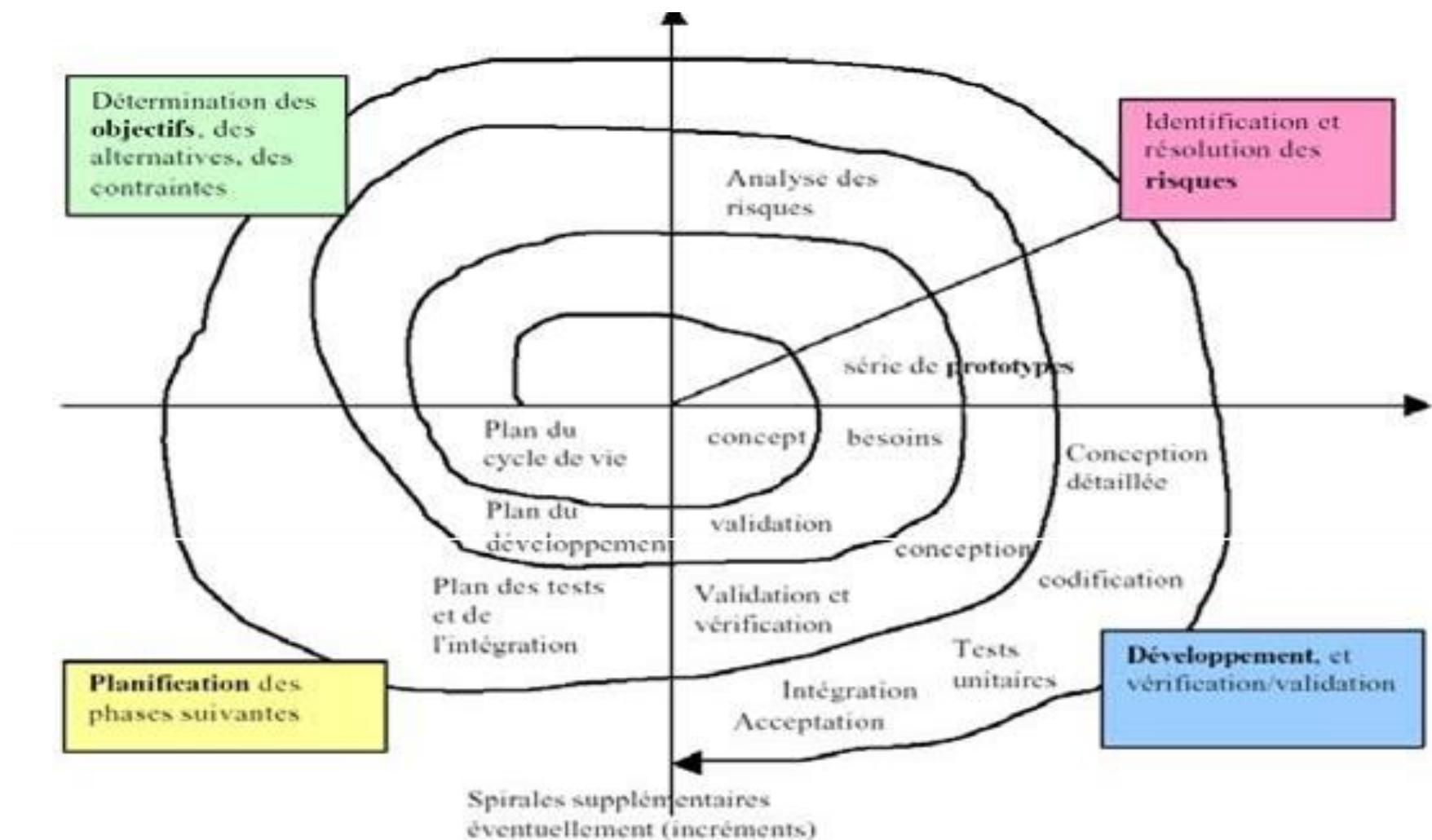


Cycle de vie en Y

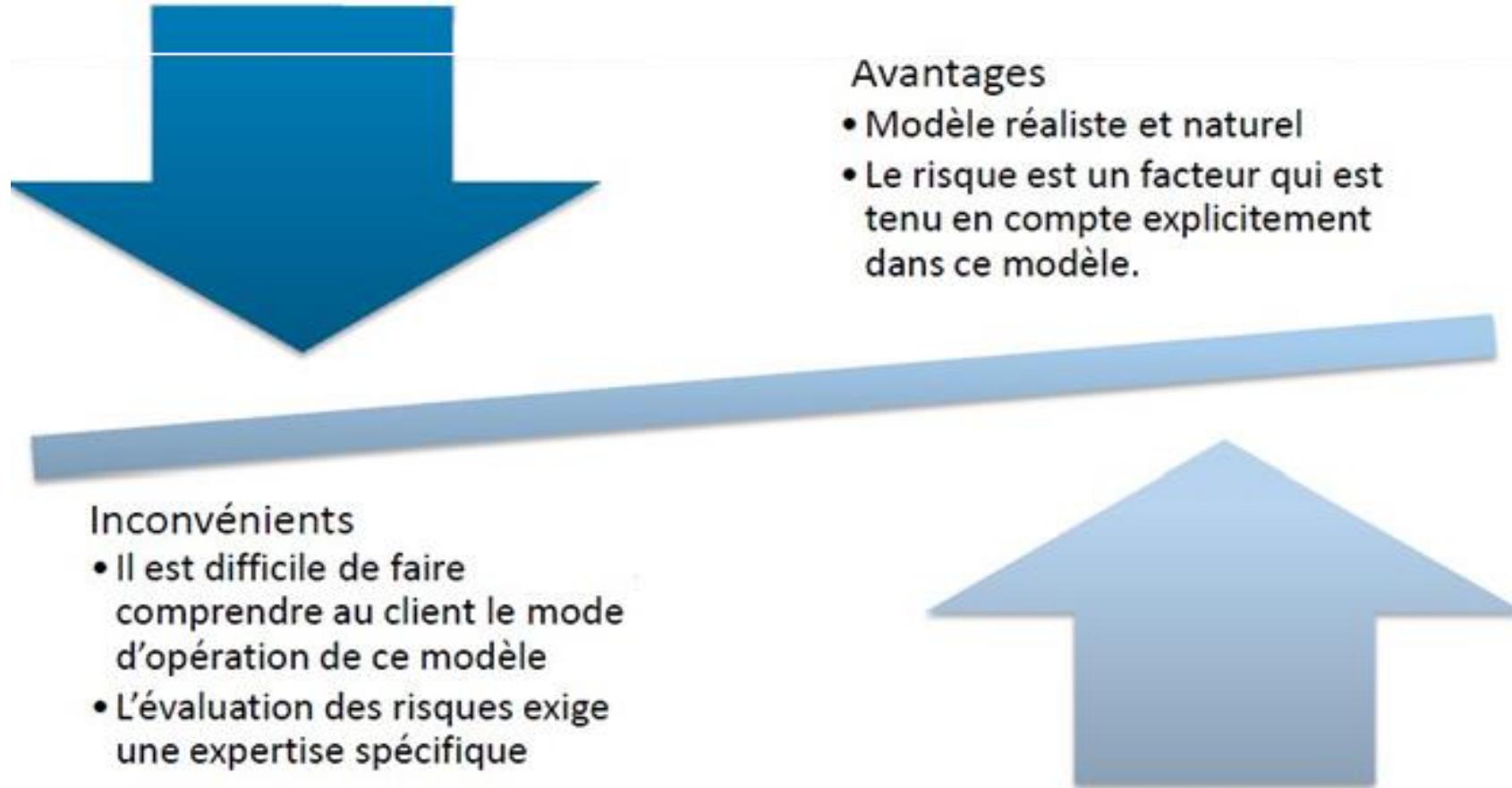


Vise à éviter les temps d'attente technique

Cycle de vie en Spirale



Cycle de vie en Spirale



Premier constat

- Maitriser les 4 variables d'ajustement sur un projet:
 - Périmètre fonctionnel (ce qu'on veut faire ?)
 - Coût (avec quel budget ?)
 - Durée (en combien de temps ?)
 - Qualité
- Définir et réserver les moyens nécessaires
- Prévoir l'organisation et la gestion du projet
- Fonder la gestion sur des prévisions !!!???

Les limites des approches classiques

- La rigidité du formalisme (la non-prévisibilité de tous les événements)
- La levée tardive des facteurs à risques (interfaces, tests de performances, ...)
- Les meilleures idées ne viennent pas forcément au début du projet
- Il est plus facile de construire par étape que tout imaginer dès le début
- Les besoins peuvent évoluer pendant le projet
- Chiffrages et reste à faire sont difficiles à évaluer

Les limites des approches classiques

- Dessine 20 voitures et colorie la moitié des voitures rouges

A worksheet page featuring a math problem at the top and drawing tasks below.

Top Section:

- Two boxes show $72 \rightarrow 12$.
- A calculation $\cancel{7} \cancel{2} \rightarrow 12$ is shown with a red checkmark.
- The word "soixante-douze" is written next to the calculation.
- A red checkmark is present on the right.

Middle Section:

E'est une bonne réponse ... mais ce n'est pas exactement ce que j'avais demandé! (It's a good answer ... but it's not exactly what I asked for!)

Task 4: Colorie la moitié des voitures en rouge.

Twenty small car icons are arranged in two rows of ten. Half of the cars (ten) have been colored red.

Task 5: Complète et calcule.

Task 6: Combien de cerises y a-t-il?

The Software Development Process



How the requirements were explained



How it was understood



How it was designed



How it was developed



What the testers received



How the consultant described it



How it was documented



What was installed



How it was billed



How it was supported



What was marketed



When it was delivered



What was really needed



The disaster recover plan



The DIY version



How it performed under load



The disaster recover plan

« Effet Tunnel »

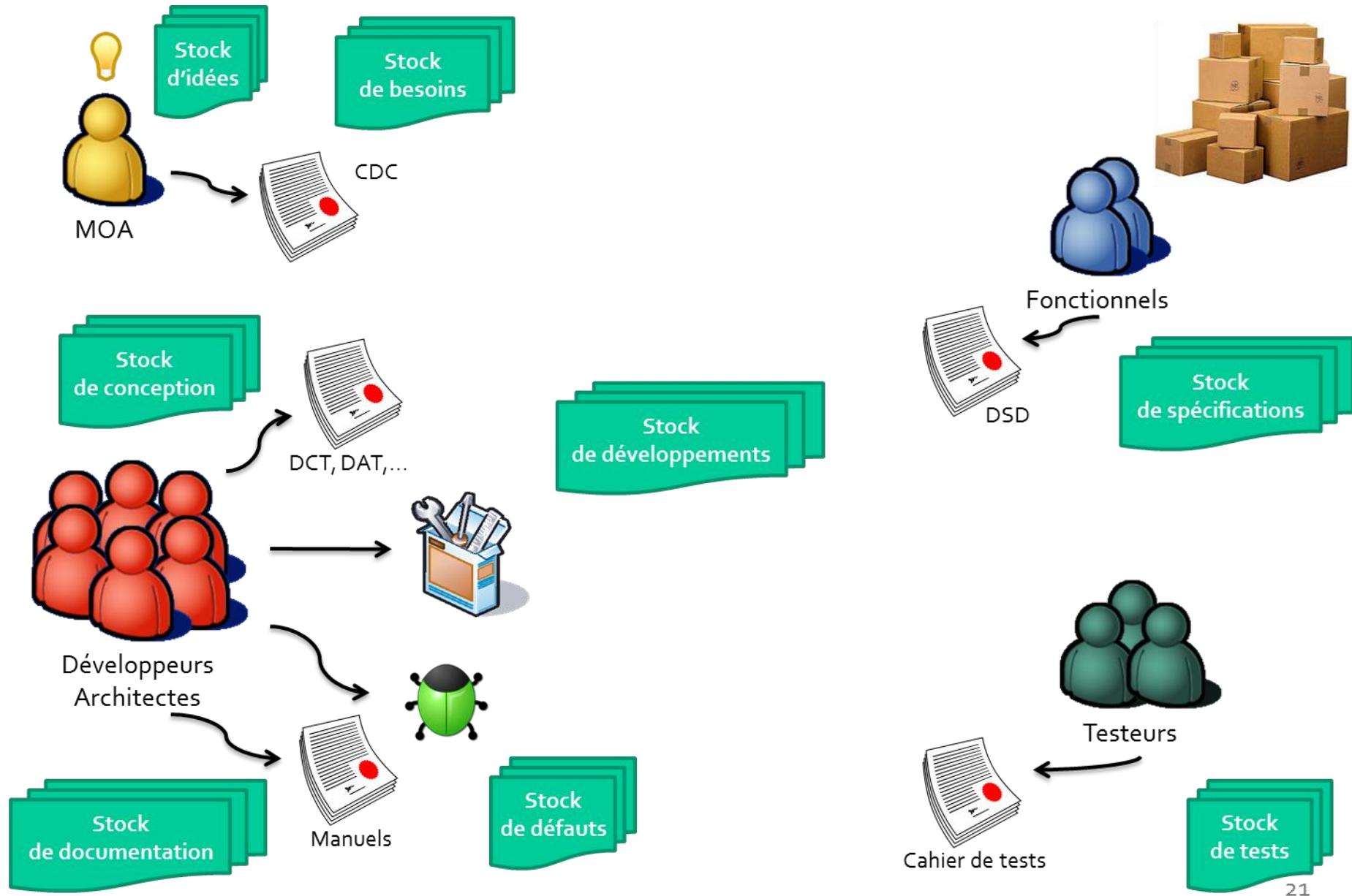
- **Définition** : Expression imagée signifiant que l'équipe projet MOA reste sans nouvelle pendant tout le temps entre la validation des besoins et la livraison du module quasi fini...
- **Le risque** : une dérive multiple – délai, budget et surtout non-conformité du produit livré avec la conception initiale... Ce qui guette toute équipe de développeurs laissée à elle-même !

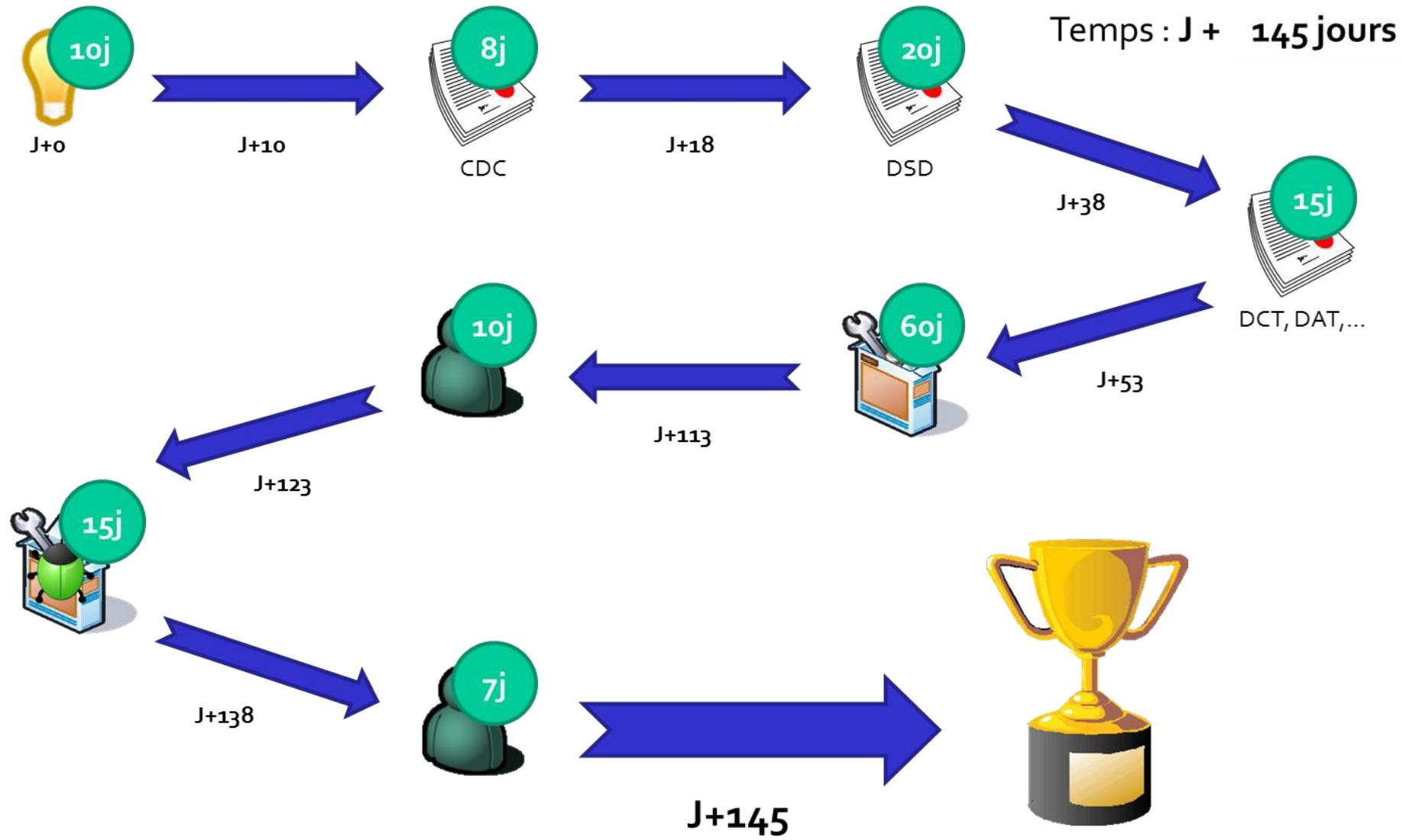


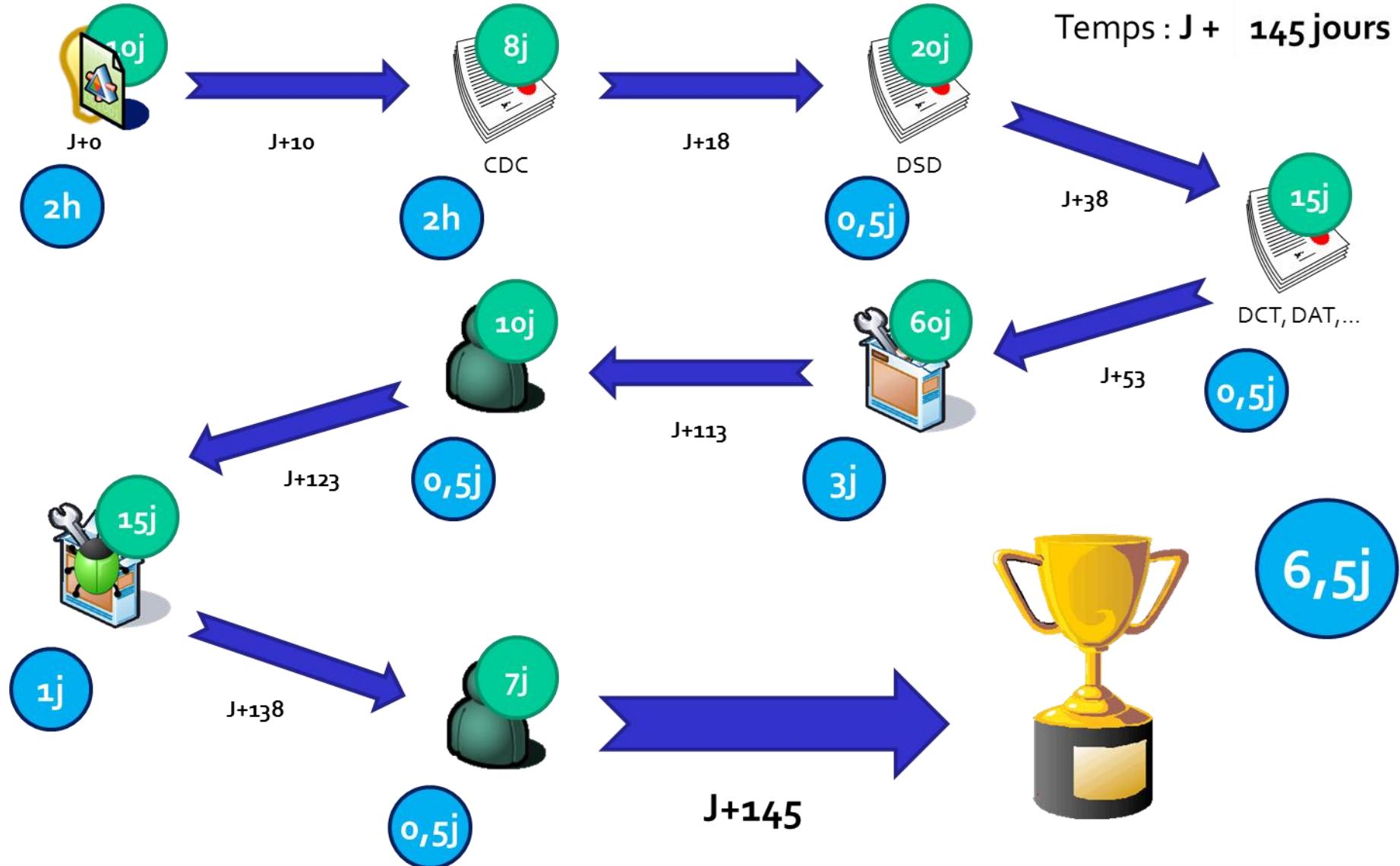
« Effet Papillon »

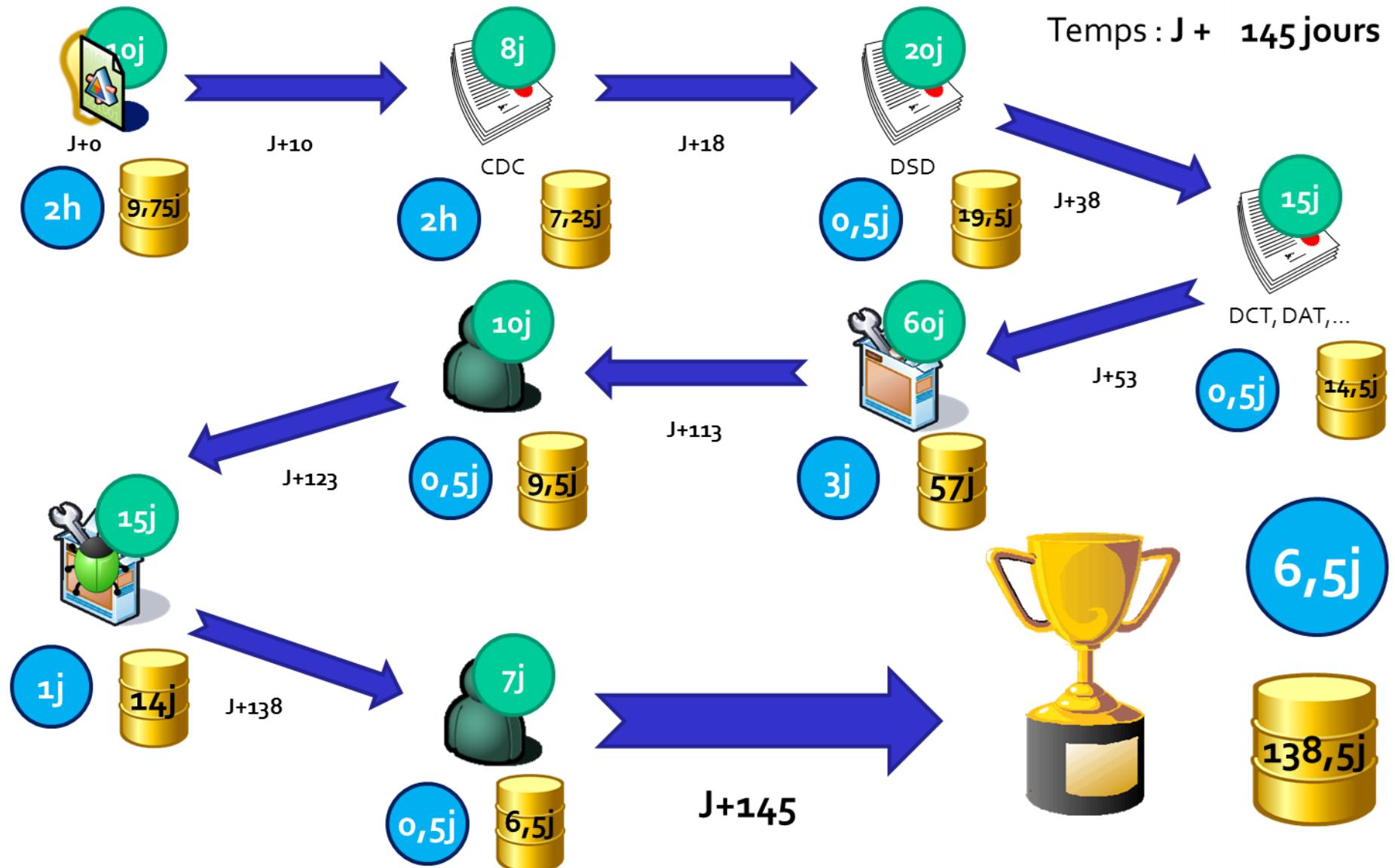
- **Définition :** une expression qui résume une image concernant le phénomène fondamental de sensibilité aux conditions initiales en théorie du chaos. Elle est parfois exprimée à l'aide d'une question :
« Un simple battement d'ailes d'un papillon peut-il déclencher une tornade à l'autre bout du monde ? ».
- **Le risque :** un élément mineur de perturbation qui par des effets induits peut provoquer des conséquences majeures sur d'autres!



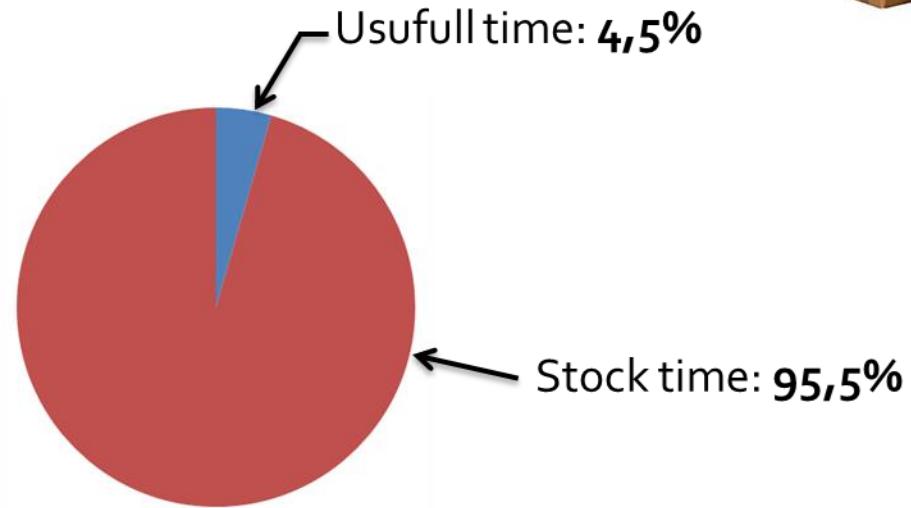




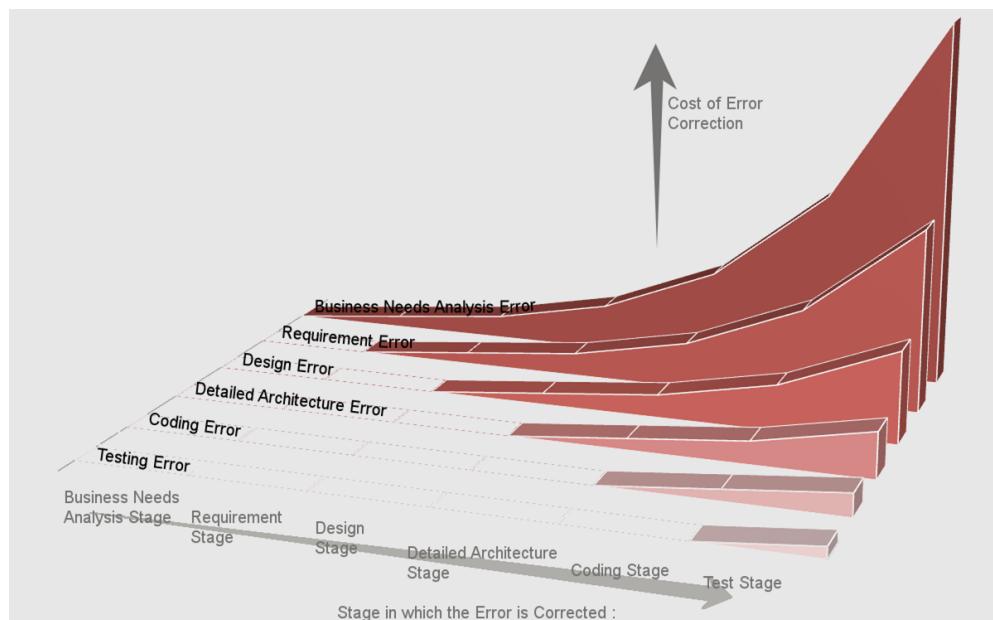




The stock time is very huge



- Time to Market ≥ 145 j
- La valeur ajoutée de développement diminue
- Les bug sont dissimulés dans les stocks
- Les bug sont découverts du coup à des stades très retardés du cycle



Agenda

Historique

Principes Lean

Manifeste agile

Scrum

Excercise on Scrum

Summary



History

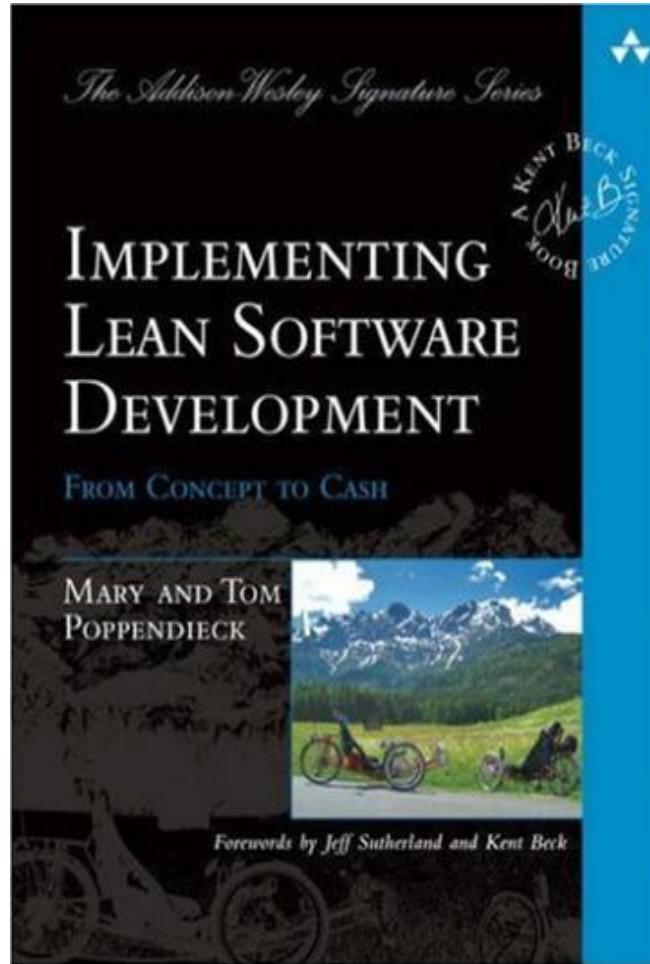
Toyota Carmaking



Success factors of Toyota

- Stopper la ligne de production
- La technique des 5 pourquoi
- Les erreurs ne doivent pas se propager dans la ligne de production
- Just-in-Time
- Respect des travailleurs
- Time-to-Market

Lean Principles and Practices



Les principes lean

1. Éliminer les sources de gaspillage
2. Favoriser l'apprentissage
3. Reporter la décision
4. Livrer vite
5. Responsabiliser l'équipe
6. Construire la qualité intrinsèque
7. Optimiser le système dans son ensemble

Eliminer les sources de gaspillage

- La philosophie Lean considère que tout ce qui n'ajoute pas de valeur au client est considéré comme un gaspillage. Ces gaspillages peuvent inclure:
 - **Travail partiellement réalisé:** Le codage abandonné au cours du processus de développement, constitue un gaspillage
 - **Processus supplémentaires:** Si une activité peut être zappé ou si le résultat peut être obtenu sans elle, il s'agit de déchets Des processus supplémentaires tels que la paperasserie ...
 - **Fonctionnalité supplémentaire:** Evolution qui ne sont pas utilisées souvent par les clients sont des gaspillages
 - **Changement de tâches:** Changer les gens de tâches sont une source de gaspillage
 - **Attente:** Attendre d'autre activités, équipes ou processus est une source de gaspillage
 - **Mouvement:** Le mouvement ou effort supplémentaire pour finir un travail est une source de gaspillage
 - **Bug:** bug et mauvaise qualité du logiciel est une source de gaspillage
 - **Management des activités:** La gestion (micro management) qui ne produit pas de valeur est une source de gaspillage
- Afin d'éliminer le gaspillage, il faut d'abord le déterminer (ou reconnaître).
- Une technique de cartographie des flux de valeur est utilisée pour identifier les gaspillages. La deuxième étape consiste à identifier les sources de gaspillage et à les éliminer. L'enlèvement des gaspillages devrait se faire de manière itérative jusqu'à la liquidation des processus et procédures apparemment essentiels.

Construire la qualité intrinsèque

- Le client doit avoir une expérience globale du système. Il s'agit de l'intégrité perçue: comment le produit est annoncé, livré, déployé, consulté, son utilisation est intuitive, son prix et sa capacité à résoudre les problèmes...
- L'intégrité conceptuelle signifie que les composants distincts du système fonctionnent ensemble dans la totalité, avec un équilibre, avec flexibilité, facilité de maintenance, efficacité et réactivité. Ceci pourrait être réalisé en comprenant la source du problème et les fixer en même temps, et non de manière séquentielle. Les informations nécessaires sont reçues par petits lots - pas dans un seul bloc - de préférence par une communication en face à face et non par une documentation écrite. Le flux d'informations doit être constant dans les deux sens - du client au développeur et vice-versa, évitant ainsi la grande quantité d'informations stressante après un long développement isolé.

Favoriser l'apprentissage

- Le développement de logiciels est un processus d'apprentissage continu basé sur des itérations lors de l'écriture de code, la conception de logiciels et la résolution des problèmes impliquant les développeurs écrivant le code. La valeur logicielle est mesurée en aptitude à l'utilisation et non en conformité avec les exigences.
- Au lieu d'ajouter de la documentation ou une planification détaillée, vous pouvez essayer différentes idées en écrivant du code en construisant le produit. Le processus de collecte des exigences des utilisateurs pourrait être simplifié en présentant des écrans aux utilisateurs finaux et en recueillant leurs commentaires. L'accumulation de défauts peut être éviter par l'exécution de tests dès que le code est écrit.
- Le processus d'apprentissage est accéléré par l'utilisation de cycles d'itération courts, avec des tests de refactorisation et d'intégration. L'augmentation du retour d'information via de brèves sessions de retour d'information avec les clients aide à déterminer la phase actuelle de développement et à ajuster les efforts pour les améliorations futures. Au cours de ces brèves sessions, les représentants des clients et l'équipe de développement apprennent les problèmes du métier et déterminent les solutions possibles pour un développement ultérieur. Ainsi, les clients comprennent mieux leurs besoins, en fonction du résultat des efforts de développement, et les développeurs apprennent à mieux répondre à ces besoins.

Reporter la décision

- Comme le développement de logiciels est toujours associé à une certaine incertitude, il convient d'obtenir de meilleurs résultats avec une approche basée sur les options, en retardant le plus possible les décisions jusqu'à ce qu'elles puissent être prises sur la base de faits et non sur des hypothèses et prédictions incertaines. Plus un système est complexe, plus la capacité de changement doit y être intégrée, ce qui permet de retarder des engagements importants et cruciaux. L'approche itérative promeut ce principe - la capacité de s'adapter aux changements et de corriger les erreurs, ce qui peut coûter très cher s'il est découvert après la sortie du système.
- Une approche de développement logiciel agile peut faire avancer la création d'options plus tôt pour les clients. Vous devez donc retarder certaines décisions critiques jusqu'à ce que les clients comprennent mieux leurs besoins. Cela permet également une adaptation ultérieure aux changements et la prévention de décisions antérieures plus coûteuses liées à la technologie. Cela ne signifie pas qu'aucune planification ne devrait être impliquée. Au contraire, les activités de planification devraient être centrées sur les différentes options et s'adapter à la situation actuelle, tout en clarifiant les situations confuses en établissant des schémas d'action rapide. L'évaluation de différentes options est efficace dès lors que l'on se rend compte qu'elles ne sont pas libres, mais qu'elles offrent la souplesse nécessaire pour prendre des décisions tardives.

Livrer vite

- À l'ère de l'évolution technologique rapide, ce n'est pas le plus gros qui survit, mais le plus rapide. Plus tôt le produit final sera livré sans défauts majeurs, plus vite le retour d'information pourra être reçu et intégré à la prochaine itération. Plus les itérations sont courtes , meilleur est l'apprentissage et la communication au sein de l'équipe. Avec la rapidité, les décisions peuvent être retardées. La rapidité assure la satisfaction des besoins actuels du client et non de ce qu'ils demandaient hier. Cela leur donne l'occasion de retarder leur décision quant à ce dont ils ont réellement besoin jusqu'à ce qu'ils acquièrent de meilleures connaissances. Les clients apprécient la livraison rapide d'un produit de qualité.
- L'idéologie de la production juste à temps pourrait être appliquée au développement de logiciels, en tenant compte de ses exigences et de son environnement. Ceci est réalisé en introduisant le résultat souhaité et en laissant l'équipe s'organiser et se répartir les tâches pour obtenir le résultat souhaité pour une itération spécifique. Au début, le client fournit les informations nécessaires. Cela pourrait simplement être présenté sous forme de petites cartes ou d'histoires - les développeurs estiment le temps nécessaire à la mise en œuvre de chaque carte. Ainsi, l'organisation du travail se transforme en système autonome: chaque matin, lors d'une réunion informelle, chaque membre de l'équipe passe en revue ce qui a été fait hier, ce qui doit être fait aujourd'hui et demain, et invite les collègues ou le client. Cela nécessite une transparence du processus, ce qui est également bénéfique pour la communication en équipe. Une autre idée clé du système de développement de produits de Toyota est la conception basée sur les ensembles. Si un nouveau système de freinage est nécessaire pour une voiture, par exemple, trois équipes peuvent concevoir des solutions au même problème. Chaque équipe découvre l'espace problématique et conçoit une solution potentielle. Comme une solution est jugée déraisonnable, elle est coupée. À la fin d'une période, les dessins survivants sont comparés et l'un choisi, éventuellement avec quelques modifications basées sur l'apprentissage des autres - un excellent exemple de report de l'engagement jusqu'au dernier moment possible. Les décisions en matière de logiciels pourraient également tirer parti de cette pratique pour minimiser les risques inhérents à une grande conception initiale.

Responsabiliser l'équipe

- Dans la plupart des entreprises, on croyait traditionnellement en ce qui concerne la prise de décision. Les dirigeants disent aux travailleurs comment faire leur propre travail. Dans une "technique de work-out", les rôles sont inversés - les gestionnaires apprennent à écouter les développeurs afin qu'ils puissent mieux expliquer les actions à entreprendre et suggérer des améliorations. L'approche Lean suit le principe Agile [6] "trouver des personnes de qualité et les laisser faire leur propre travail" [7] en encourageant le progrès, en détectant les erreurs et en supprimant les obstacles, mais en ne gérant pas la micro-gestion.
- Une autre croyance erronée a été la considération des personnes en tant que ressources. Les personnes peuvent être des ressources du point de vue d'une fiche de données statistiques, mais dans le développement de logiciels, ainsi que dans toute activité organisationnelle, les utilisateurs ont besoin de plus que la simple liste de tâches et l'assurance de ne pas être dérangés pendant l'achèvement. des tâches. Les gens ont besoin de motivation et d'un objectif plus élevé pour travailler efficacement dans la réalité, avec l'assurance que l'équipe peut choisir ses propres engagements. Les développeurs doivent avoir accès au client; le chef d'équipe doit apporter son aide et son assistance dans les situations difficiles et veiller à ce que le scepticisme ne gâche pas l'esprit de l'équipe.

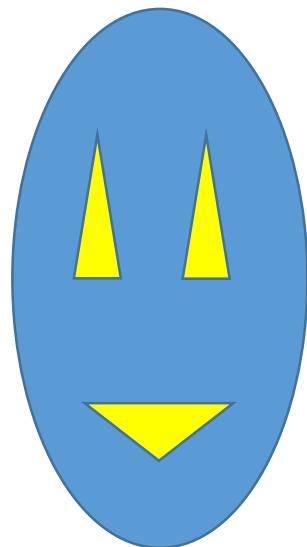
Optimiser le système dans son ensemble

- Les systèmes logiciels actuels ne sont pas simplement la somme de leurs composants, mais aussi le produit de leurs interactions. Les défauts des logiciels ont tendance à s'accumuler au cours du processus de développement - en décomposant les tâches les plus importantes en tâches plus petites et en normalisant les différentes étapes du développement, il convient de rechercher et d'éliminer les causes profondes des défauts. Plus le système est grand, plus les organisations impliquées dans son développement et les pièces développées par différentes équipes sont nombreuses, plus il est important de disposer de relations bien définies entre différents fournisseurs, afin de produire un système avec des composants en interaction douce
- La pensée Lean doit être bien comprise par tous les membres d'un projet avant d'être mise en œuvre dans une situation concrète et réelle. "Pensez grand, agissez petit, échouez vite, apprenez vite" [8]: ces slogans résument l'importance de la compréhension du domaine et de la pertinence de la mise en œuvre de principes lean tout au long du processus de développement logiciel. Ce n'est que lorsque tous les principes Lean sont mis en œuvre ensemble, combinés avec un "sens commun" fort en ce qui concerne l'environnement de travail, qu'il existe une base de succès pour le développement de logiciels.

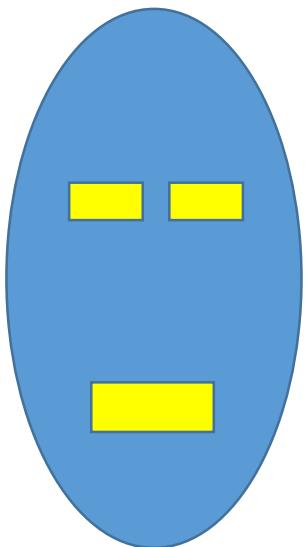
Exercise, Mr Happy Face 😊



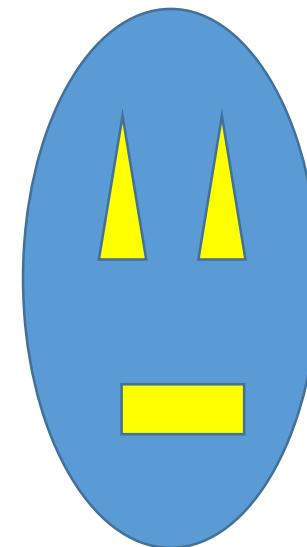
Happy Faces – The Models



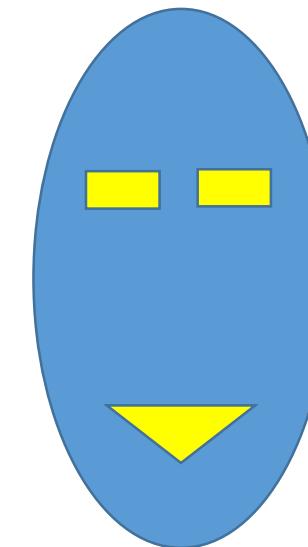
Adelaide



Laurent



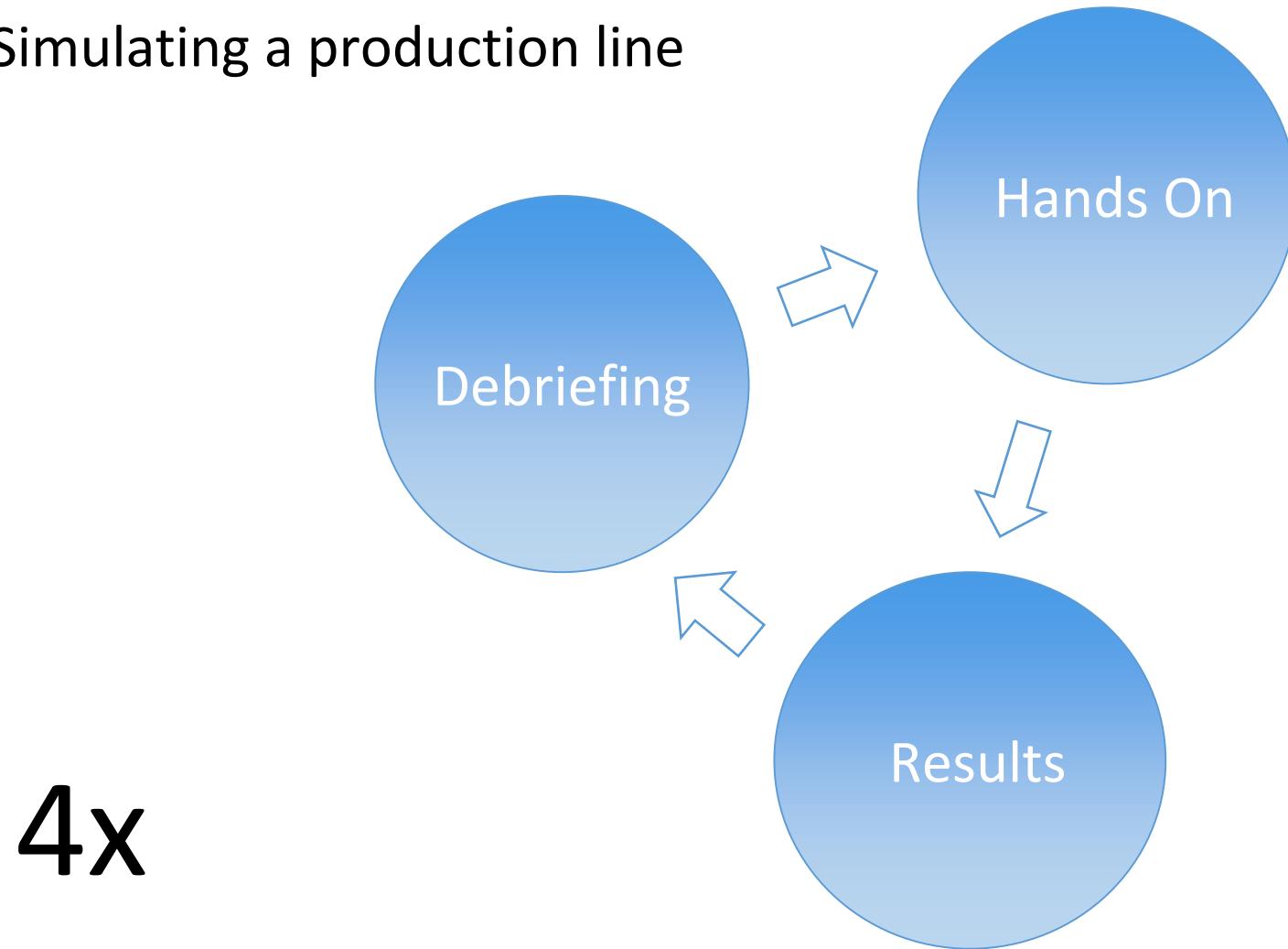
Pierre



Sofie

Exercise

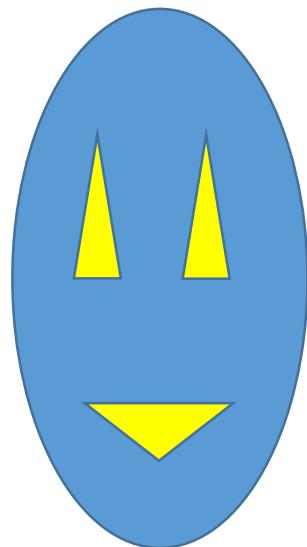
Simulating a production line



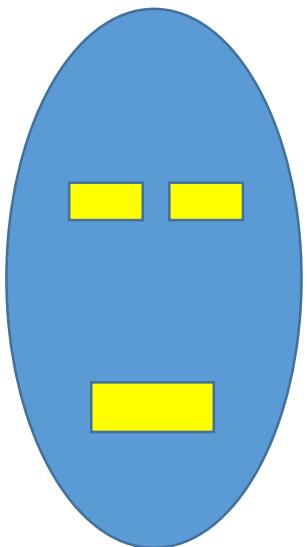
First Run – Push Process

1. Roles:
 1. Sales Manager
 2. Development team
2. A face is cut from the blue paper
3. Draw the Eyes and Mouth (design) 
4. The appropriate eyes are affixed
5. The appropriate mouth is affixed 
6. The face is taped to the wall 
7. **Create one face before starting, to try it out**
8. *Wait for my signal* 
4 minutes

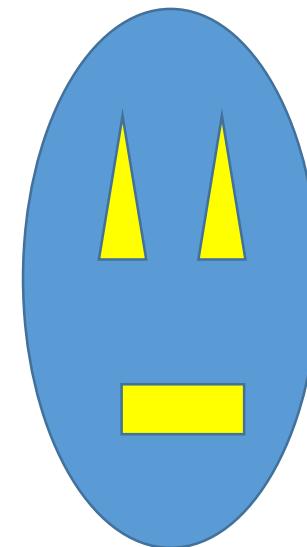
Happy Faces – The Models



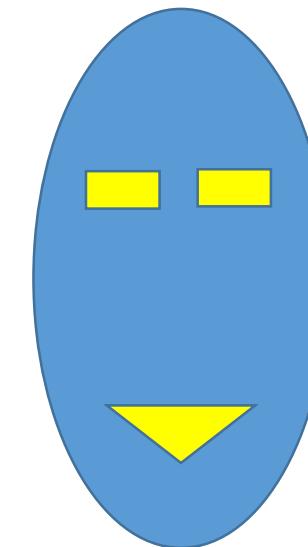
Adelaide



Laurent



Pierre



Sofie

Calculate profit and loss

Every sold face = +\$400

Every unsold complete face = -
\$200

Every unsold eye = -\$25

Every unsold mouth = -\$50

Every uncompleted face = -\$100



Retrospective



**What good things did we do?
What could have been better?
What can we improve?**

Gaspillage

Inventaire

sur/sous Production



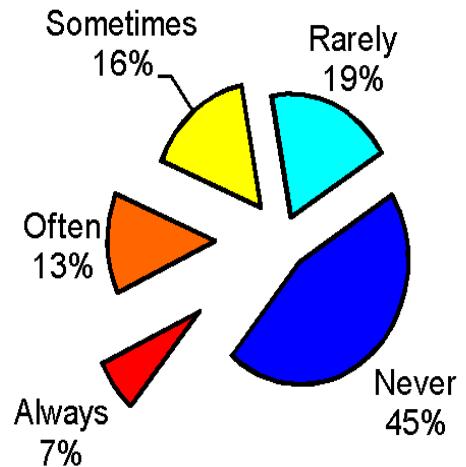
1. Eliminer le gaspillage



Enlever toutes les activités sans valeur ajoutées

Example: Développer plus qu'on peut tester

Features and Functions Used in a Typical System



Standish Group Study Reported at XP2002 by Jim Johnson, Chairman



7 gaspillages du developement logiciel

Sur-production

Attente

Transportation

Sur-process

Inventaire

Motion (people)

Defects

Extra évolution

Attente de décision

Handoffs

Extra steps (specifications)

Travail en cours

Travail incomplet

Trouver l'information

Bugs

Développer pour aujourd'hui

Petit incrément

Include customers

Revu du process (simplifier)

Détailler le just in time

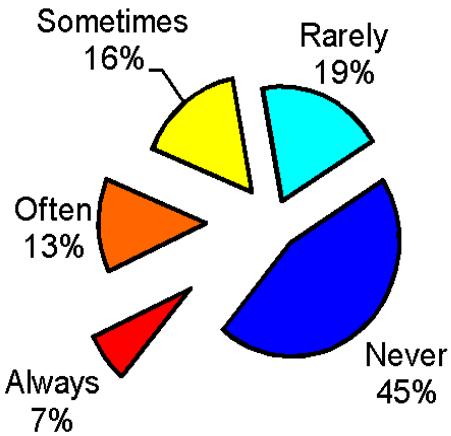
Tous dans le même plateau

Tester tot

Myth: Spécification précauce évite le gaspillage

Ne forcer pas les client à donner la liste de ce qu'ils veulent,
Ils créeront une liste de tout, ce qu'ils veulent et ce qu'ils veulent pas

Features and Functions Used in a Typical System



Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

2. Développer dans la qualité

Myth:

Avoir le produit parfait à la première
release

Release avec des itérations

Release le scope minimale au début

Refactoriser le code pour améliorer qualité

Refactoriser pour améliorer l'architecture

Utiliser m'intégraton contenu

L'objectif: valeur durable

Myth:

Le but des tests est de trouver les bug

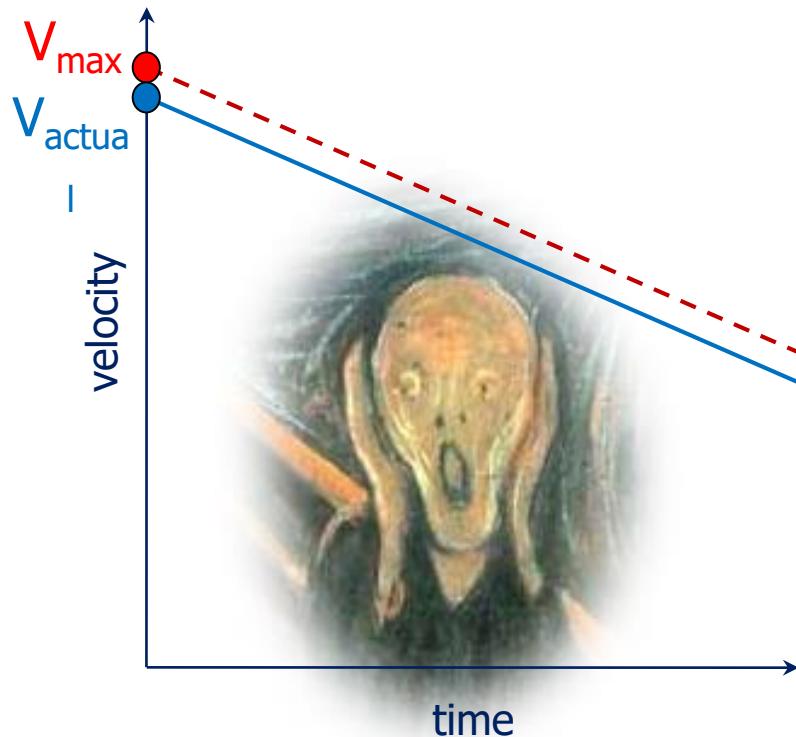
Le but des tests et de **prévenir** les bug

Vérification finale est **cruciale**

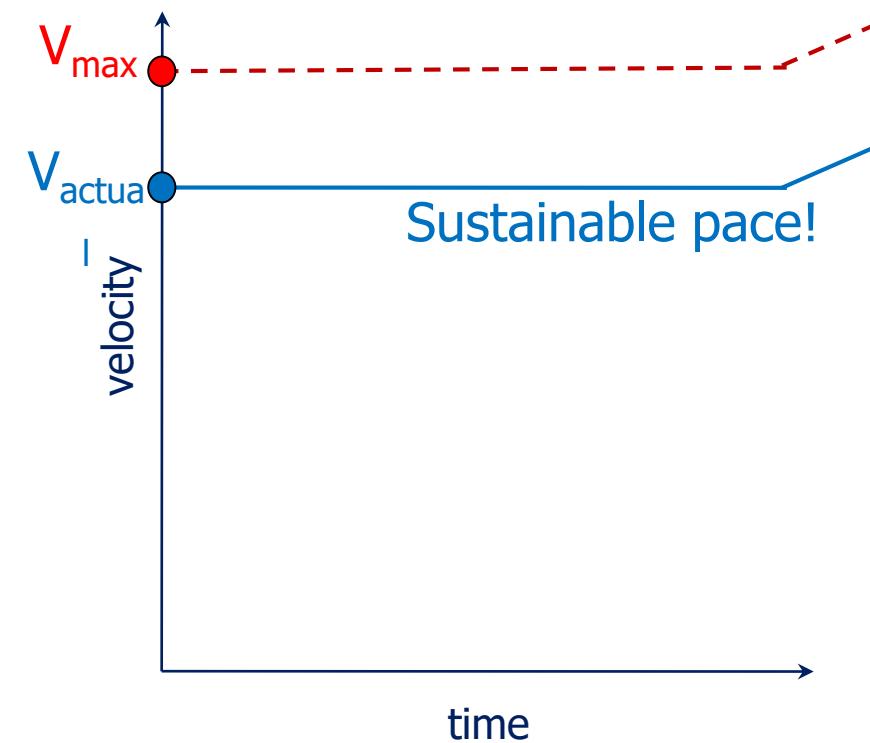
mais trouver des bug doit etre une
exception

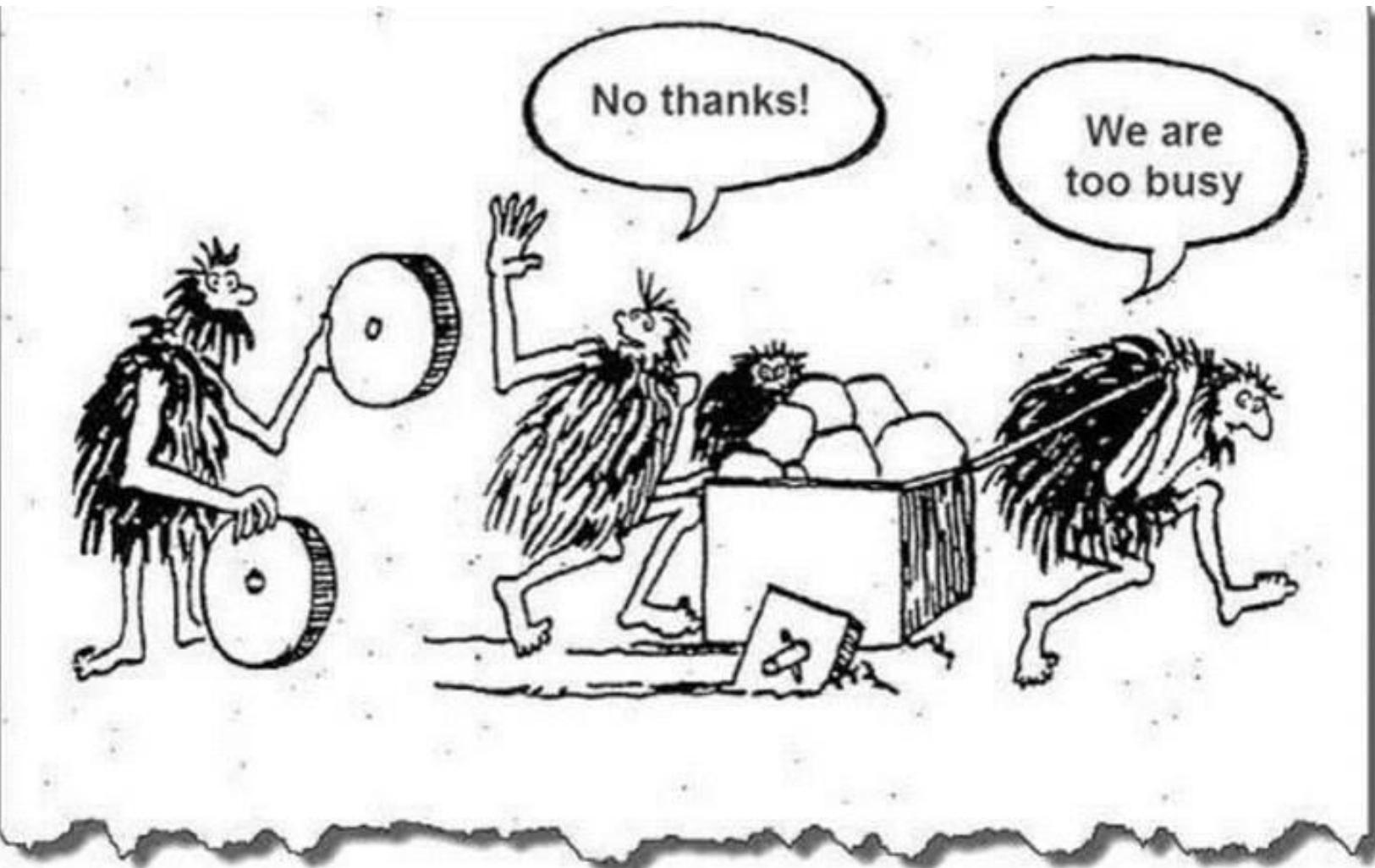
Technical debt = Financial debt

Pas de refactoring
Pas de tests unitaire
Qualité médiocre
Heures supplémentaires



Refactoring
Unit tests
Attention à la qualité
Heure de travaille normale



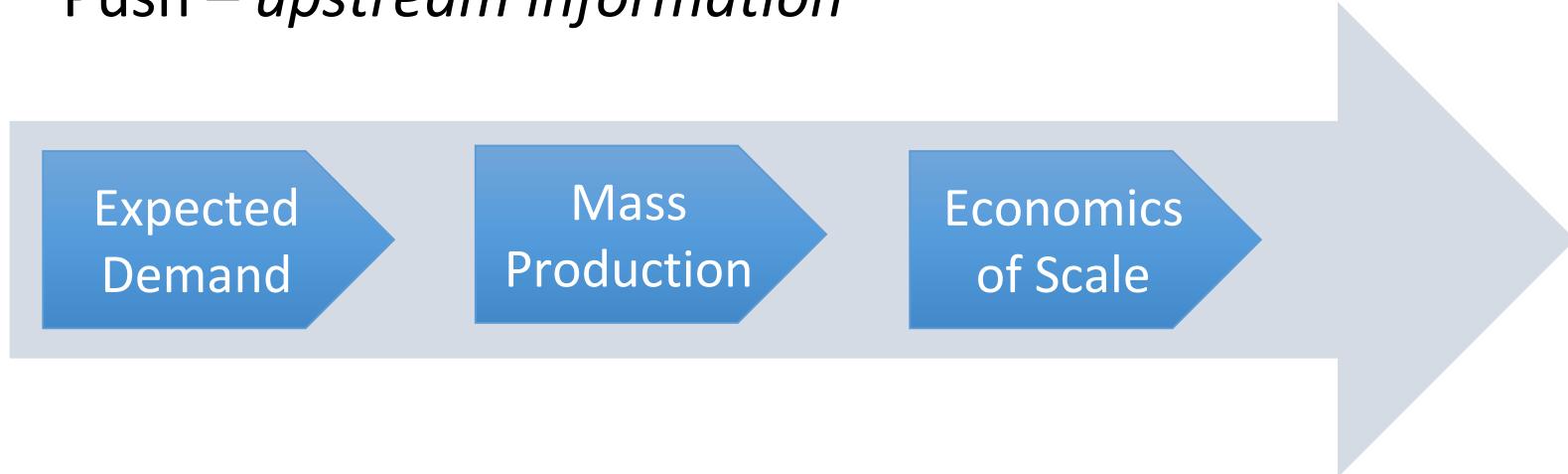


No thanks!

We are
too busy

Push and Pull Systems

Push – *upstream information*



Push

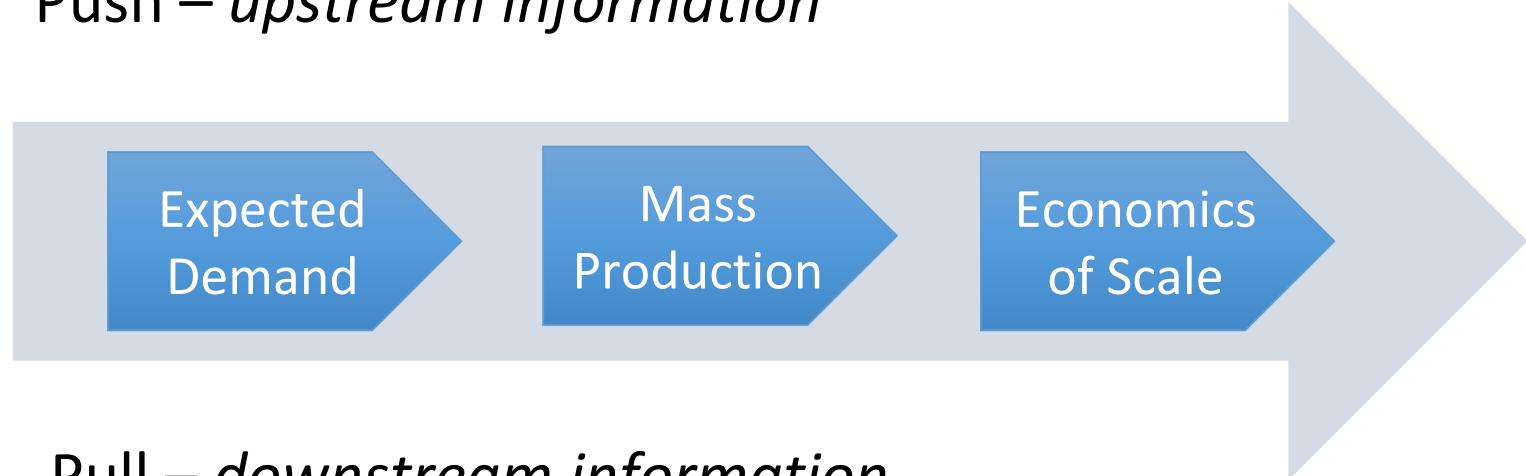


"You can have any color, as long as it's black"

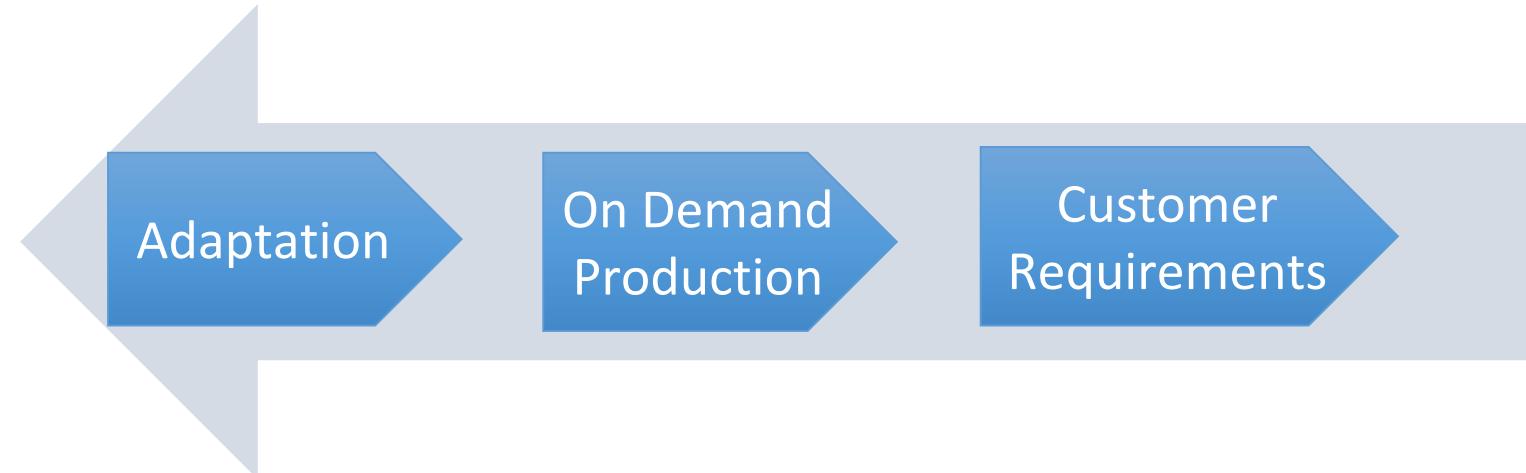
Henry Ford

Push and Pull Systems

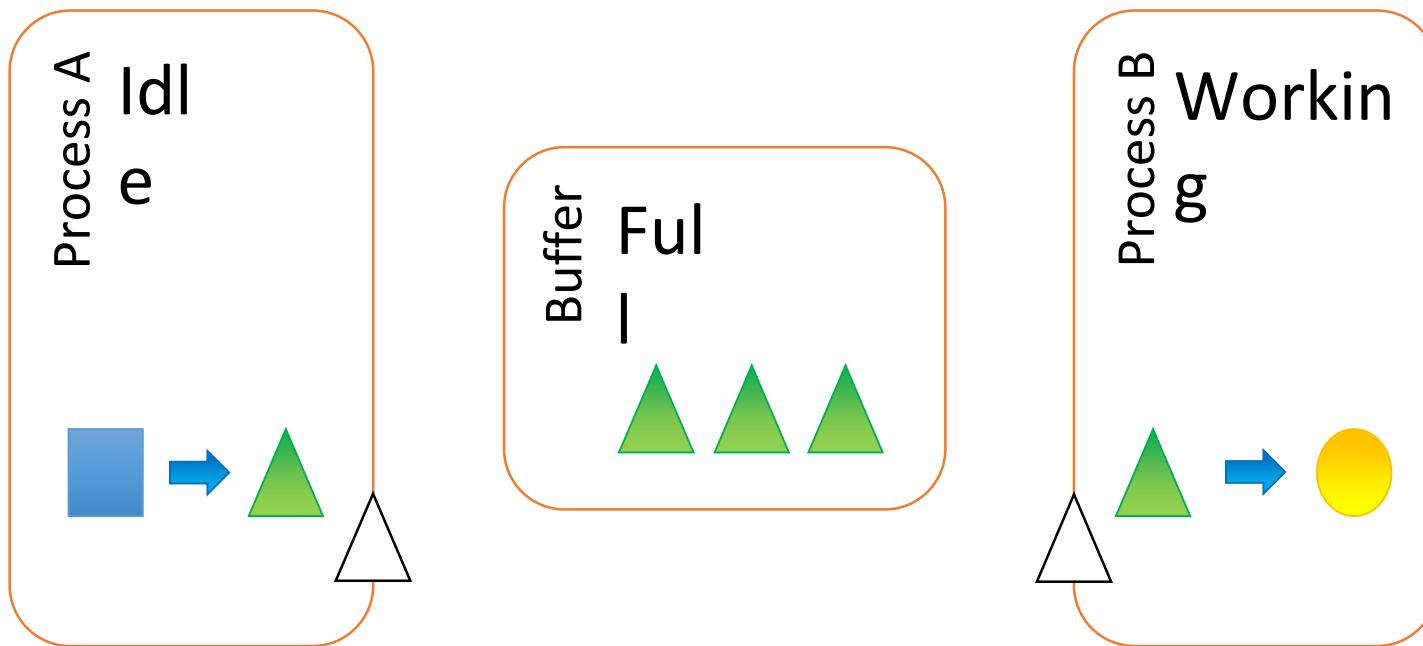
Push – upstream information



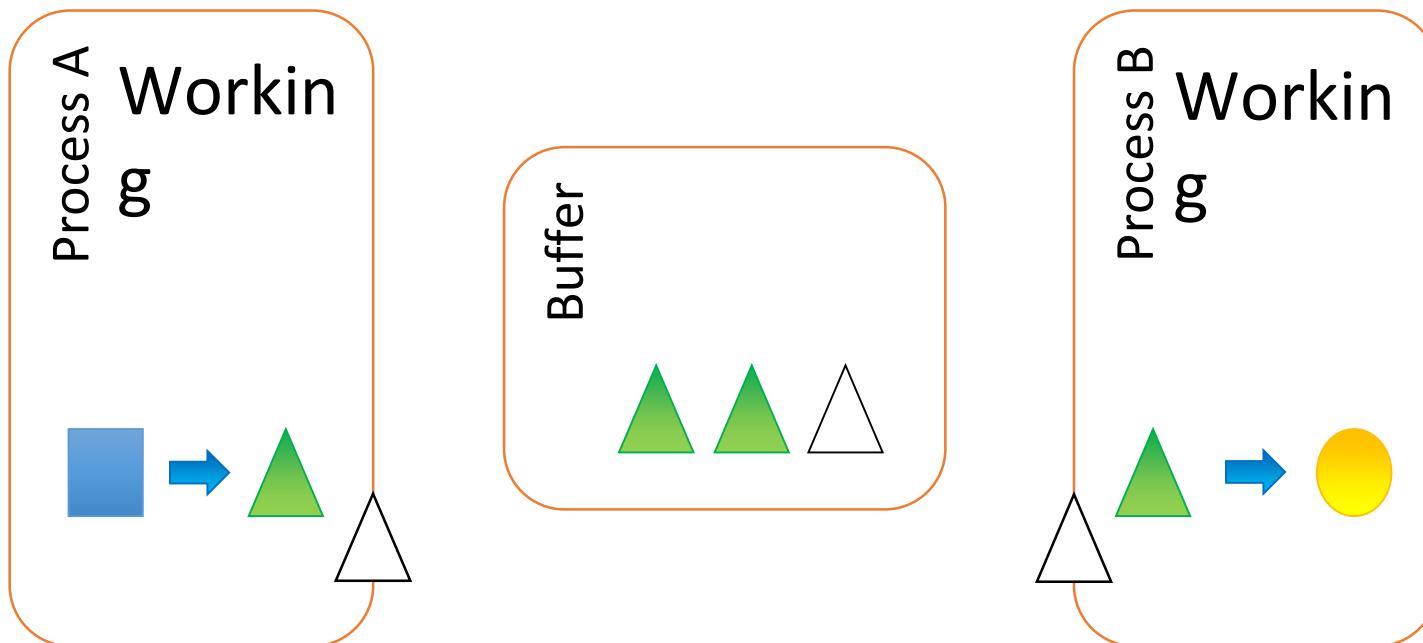
Pull – downstream information



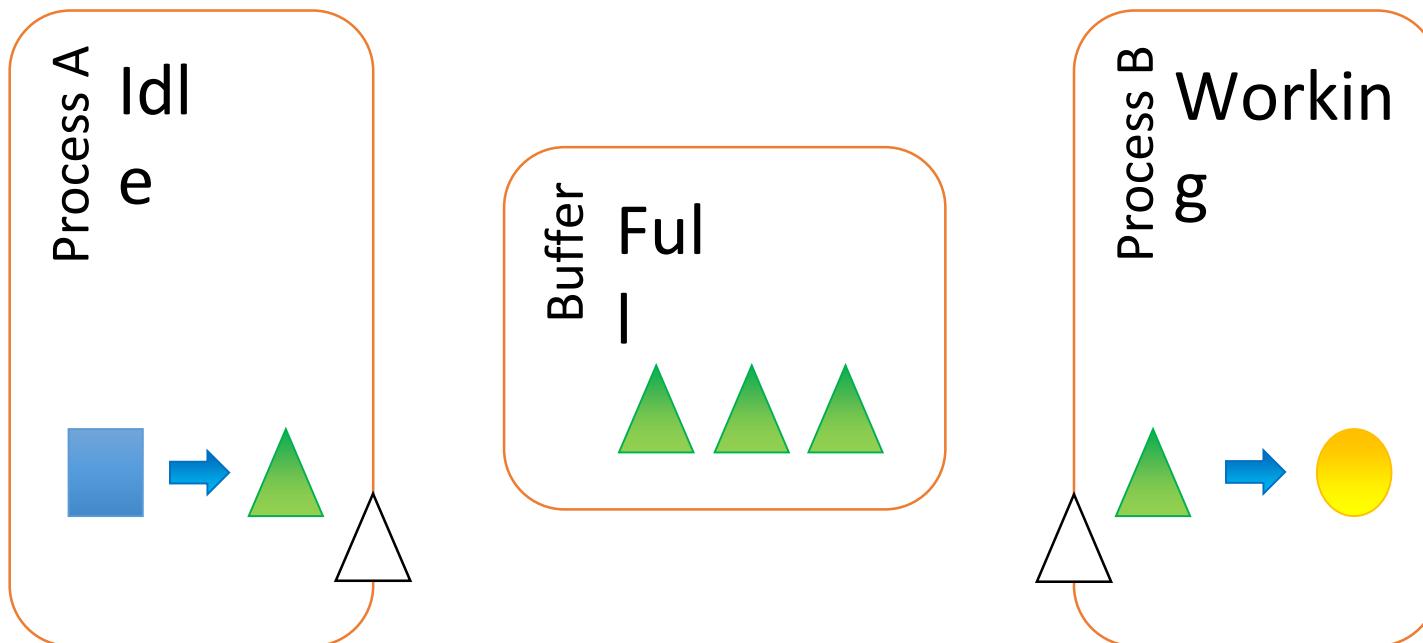
Kanban



Kanban



Kanban



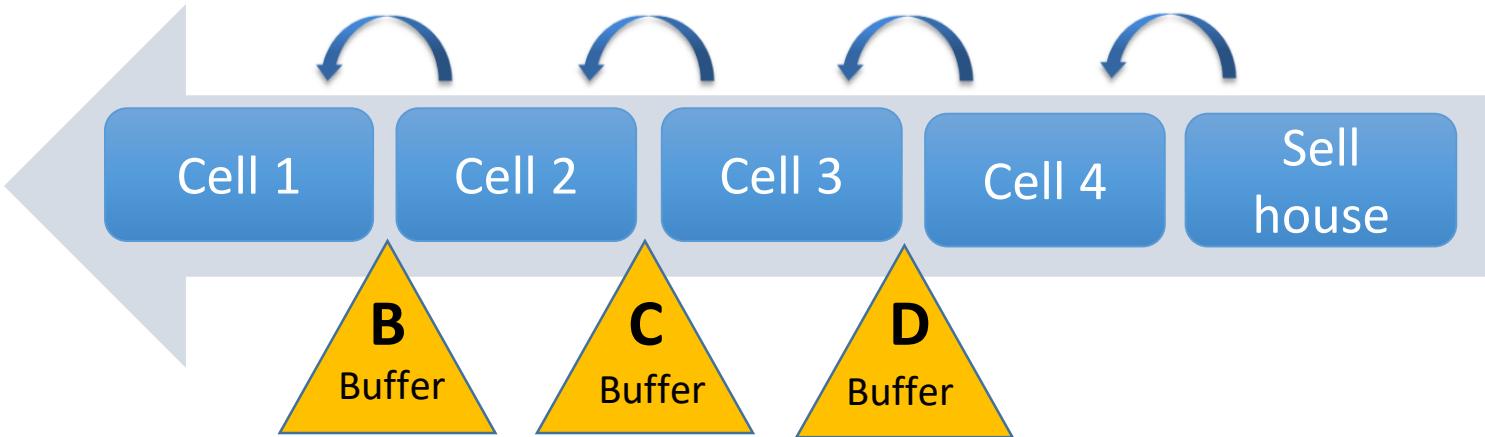
Kanban

Limits Work in Progress (WIP) on a pull system

Aids visual control →

Helps to get an even and maximized flow

Second Run – Pull Process



Pull System with Kanban

Demande vient du client

Items sont produit pour remplir le gap

Mettre un buffer dans les étapes intermediaires

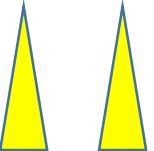
Second Run – Pull Process

1. Roles:
 1. Sales Manager
 2. Development team
2. FQ -Face Queue: One blank face
3. EQ- Eye Queue: Two sets of eyes,
rectangular and triangular
4. UFQ: Two uncompleted faces with eyes attached only.
5. MQ: Two mouths, one rectangular, one triangular
6. **Each queue is created on one piece of paper**
7. **Fill all queues before starting**
8. *Wait for my signal* ☺
4 minutes

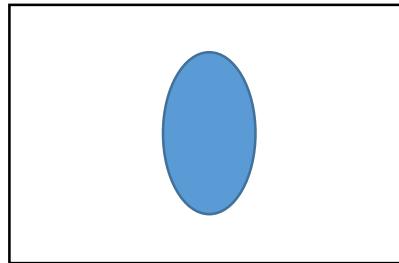
4 minutes



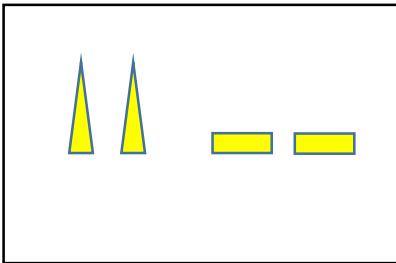
Happy Faces – The Models



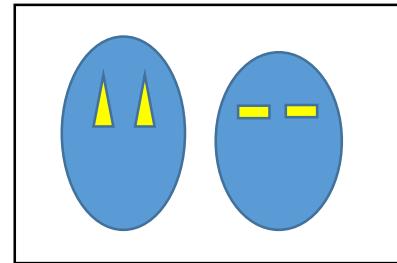
FQ



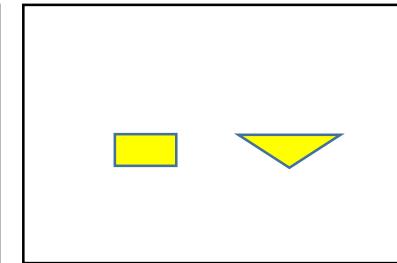
EQ



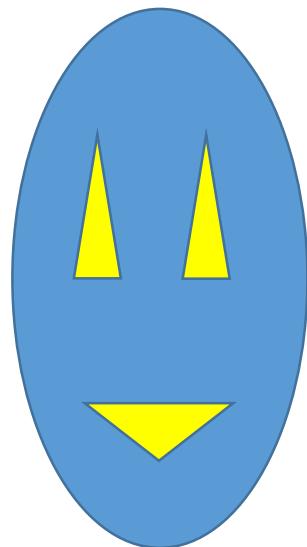
UFQ



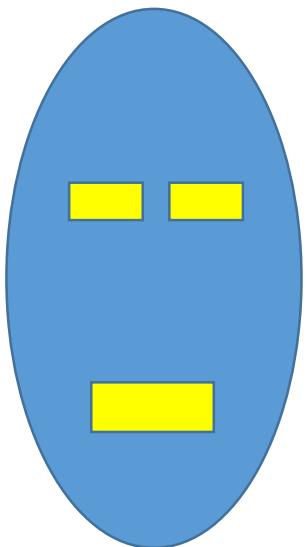
MQ



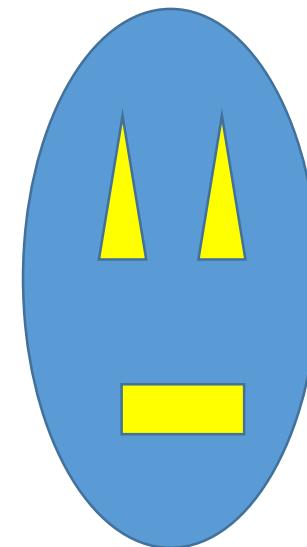
Happy Faces – The Models



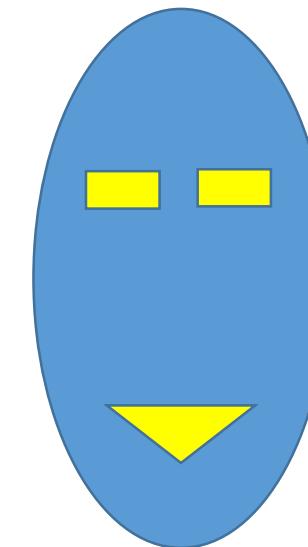
Adelaide



Laurent



Pierre



Sofie

Calculate profit and loss

Every sold face = +\$400

Every unsold complete face = -
\$200

Every unsold eye = -\$25

Every unsold mouth = -\$50

Every uncompleted face = -\$100



Retrospective



**What good things did we do?
What could have been better?
What can we improve?**

Lissage du processus ou production simplifiée

Rendre la ligne de production lisse

Heijunka (stop the line when needed) 平準化

Certaines personnes travaillent plus que d'autres

Réduire les travailleurs au ralenti

Rythme durable

3. Amplifier l'apprentissage

Expérimentation et collecte des feedback

Arreterez vous et réfléchissez

Partage des connaissances

L'apprentissage est essentiel



Kaizen = Continuous Improvement

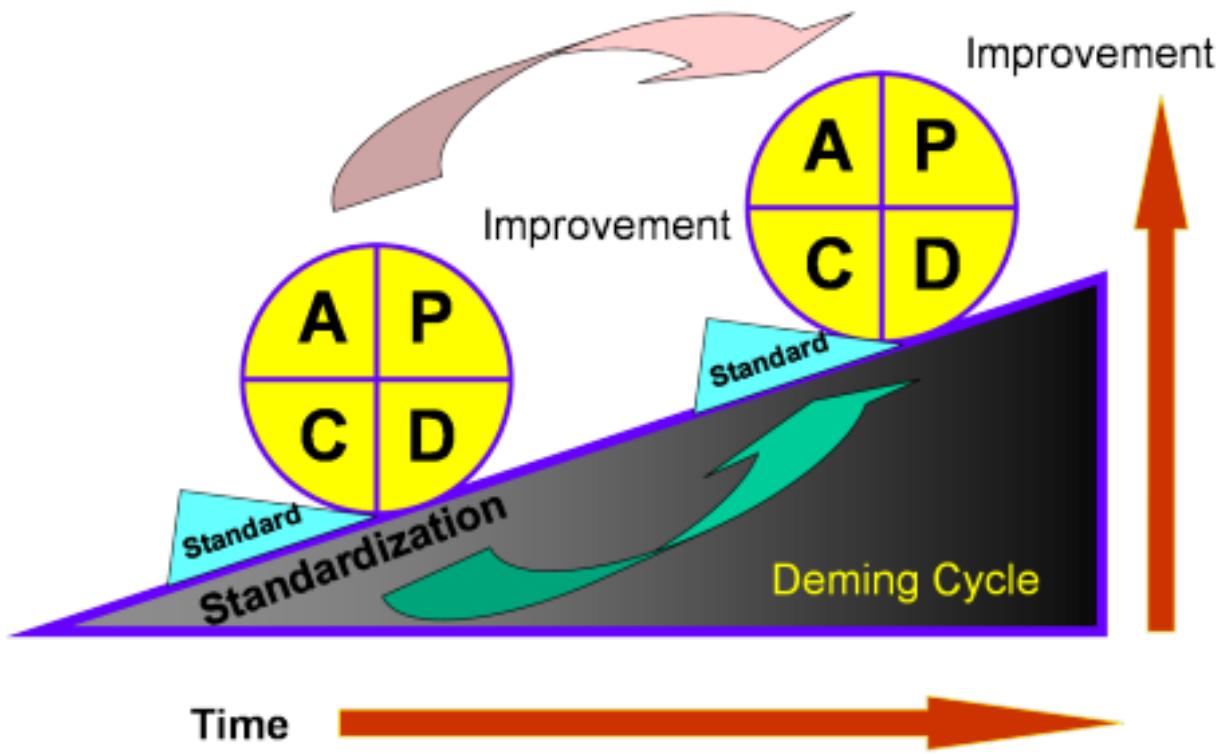
Reflèchire et adapter
Reflexion long terme
Deming circle P-D-C-A
Apprentissage par la standardisation

改善



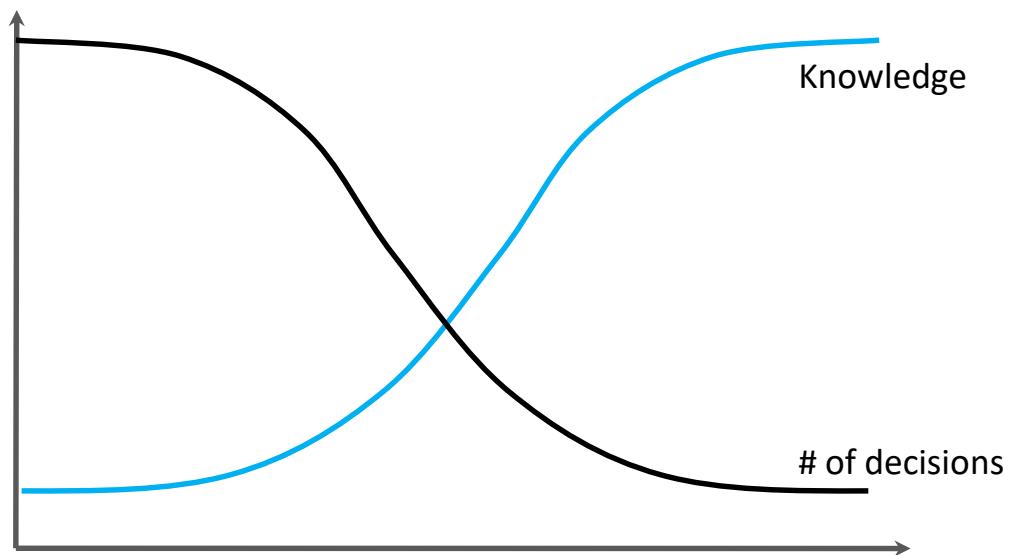
Kaizen = Improvement

改善



4. Retarder la décision

Decider quand la decision devrait etre prise



Abolir l'idée qu'il est judicieux de commencer le développement avec une spécification complète

Myth: La planification est un engagement

"In preparing for battle I have always found that plans are useless,
but planning is indispensable"

General Dwight D. Eisenhower

Un plan peut et va etre changer

Bienvenue!

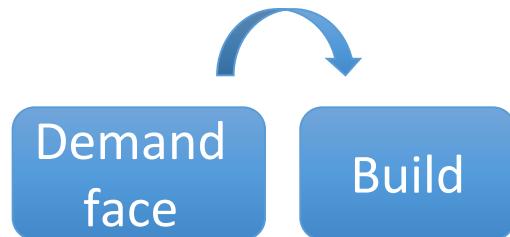
Diminuer les dépendances

L'architecture doit permettre l'ajout de toute fonctionnalité à tout moment.

Maintenir la possibilité d'options

L'architecture doit permettre l'ajout de toute fonctionnalité à tout moment.

Third Run – WorkCells



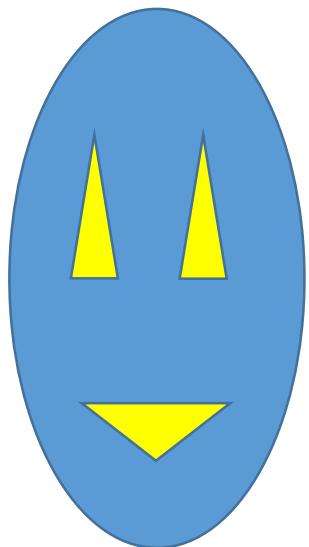
Work Cell

Each person builds a complete face

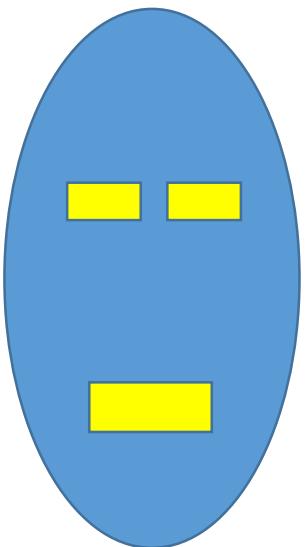
One round

4 minutes

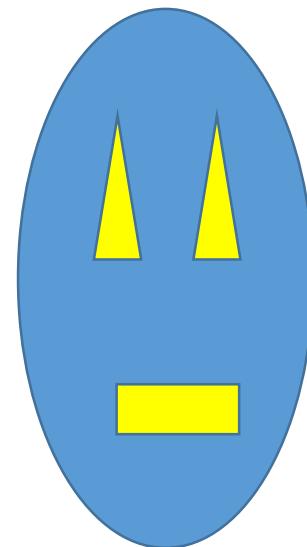
Happy Faces – The Models



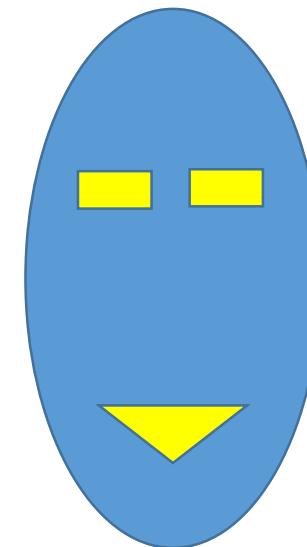
Adelaide



Laurent



Pierre



Sofie

Calculate profit and loss

Every sold face = +\$400

Every unsold complete face = -
\$200

Every unsold eye = -\$25

Every unsold mouth = -\$50

Every uncompleted face = -\$100



Retrospective



**What good things did we do?
What could have been better?
What can we improve?**

Forth Run – Improve the Process

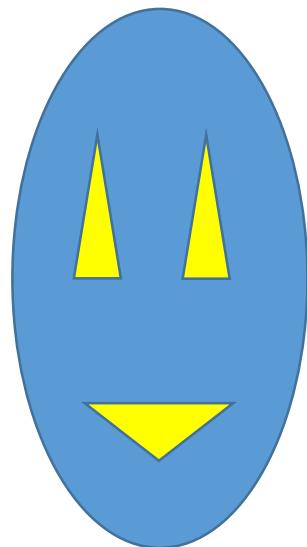


Discuss your own process (2 min)

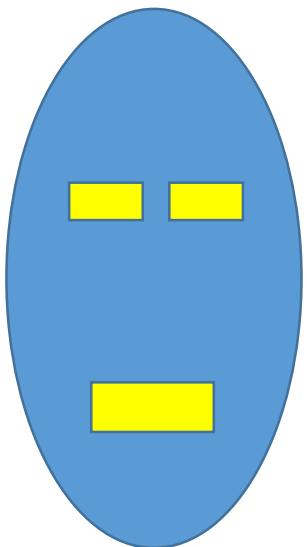
1 round

4 minutes

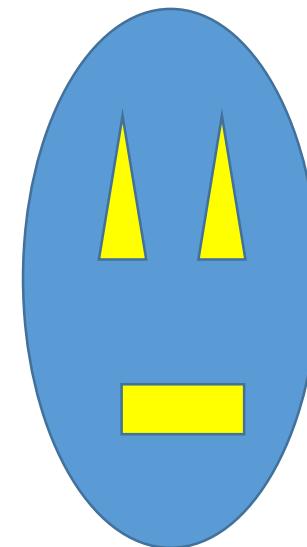
Happy Faces – The Models



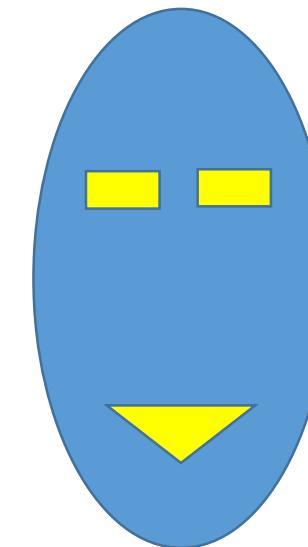
Adelaide



Laurent



Pierre



Sofie

Calculate profit and loss

Every sold face = +\$400

Every unsold complete face = -
\$200

Every unsold eye = -\$25

Every unsold mouth = -\$50

Every uncompleted face = -\$100



Retrospective



**What good things did we do?
What could have been better?
What can we improve?**

5. Livrer vite

Retour/feedback rapide

Liverer ce que veut le client maintenant



“Instead of having
many people spending a long time doing a big thing
you have
a small team spending short time doing a small thing”

6. Responsabiliser l'équipe

Les personnes engagées offrent un avantage en matière de développement durable.

Les équipes prospèrent sur la base de la fierté, de l'engagement, la confiance et les encouragements

L'équipe sait mieux comment les choses sont fais

Fournir des servant leaders

Effective teams have servant leaders who bring out the best in the team

Respect des collègues



7. Optimiser le système dans son ensemble

Optimiser dans l'ensemble

Travailler avec un objectif commun

Concentrez-vous sur l'ensemble du flux de valeur

Du concept à la rentabilité

de la demande client à la mise en production

Organisation croisé

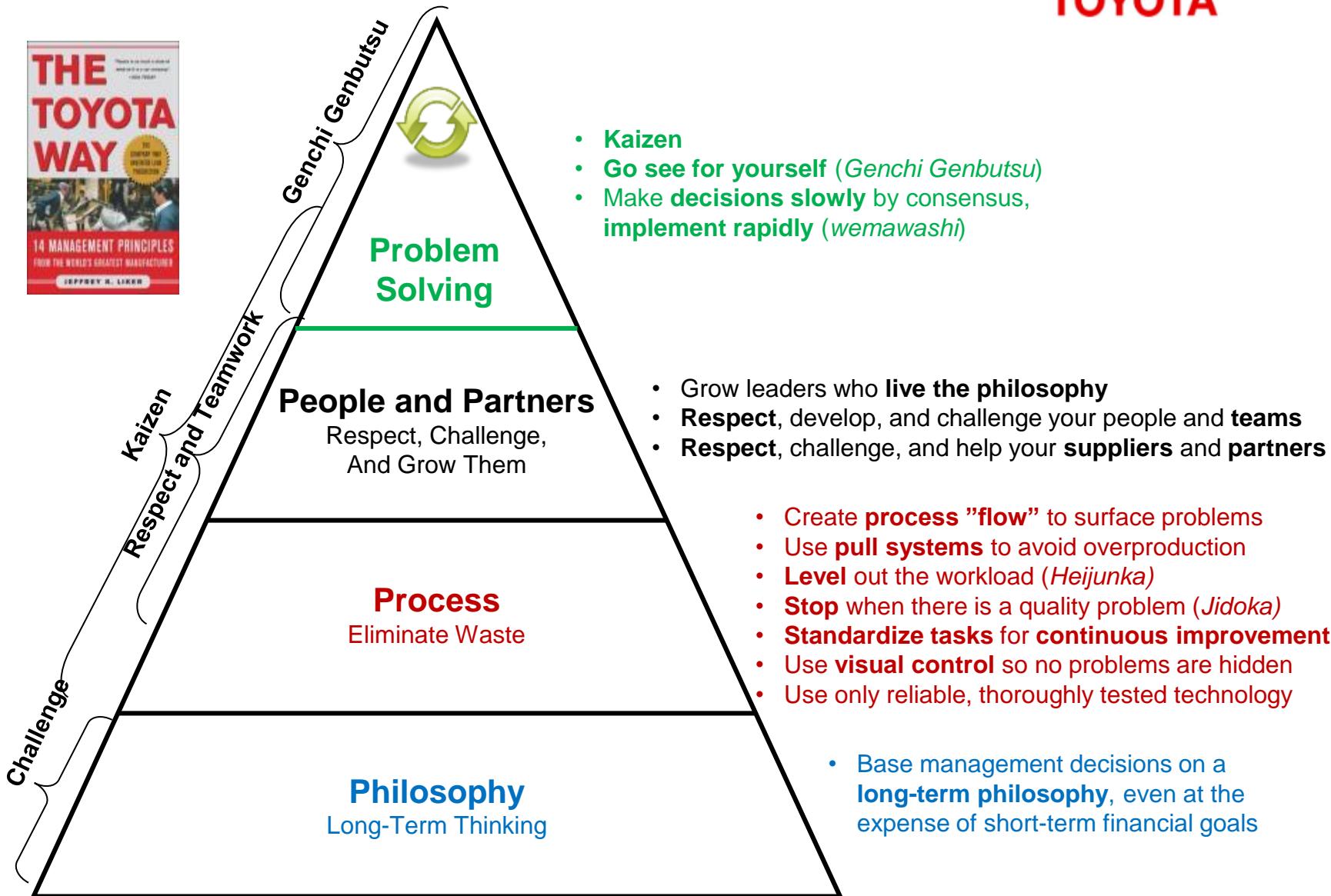
Livrer un produit complet



**Un produit complet est fourni par des équipes
qui se complètent**



Lean Principles of Toyota



Apply Lean principles

Adjust it to fit **YOUR** reality

Agenda

History and Background

Lean Principles

Agile Manifesto

Scrum

Excercise on Scrum

Summary



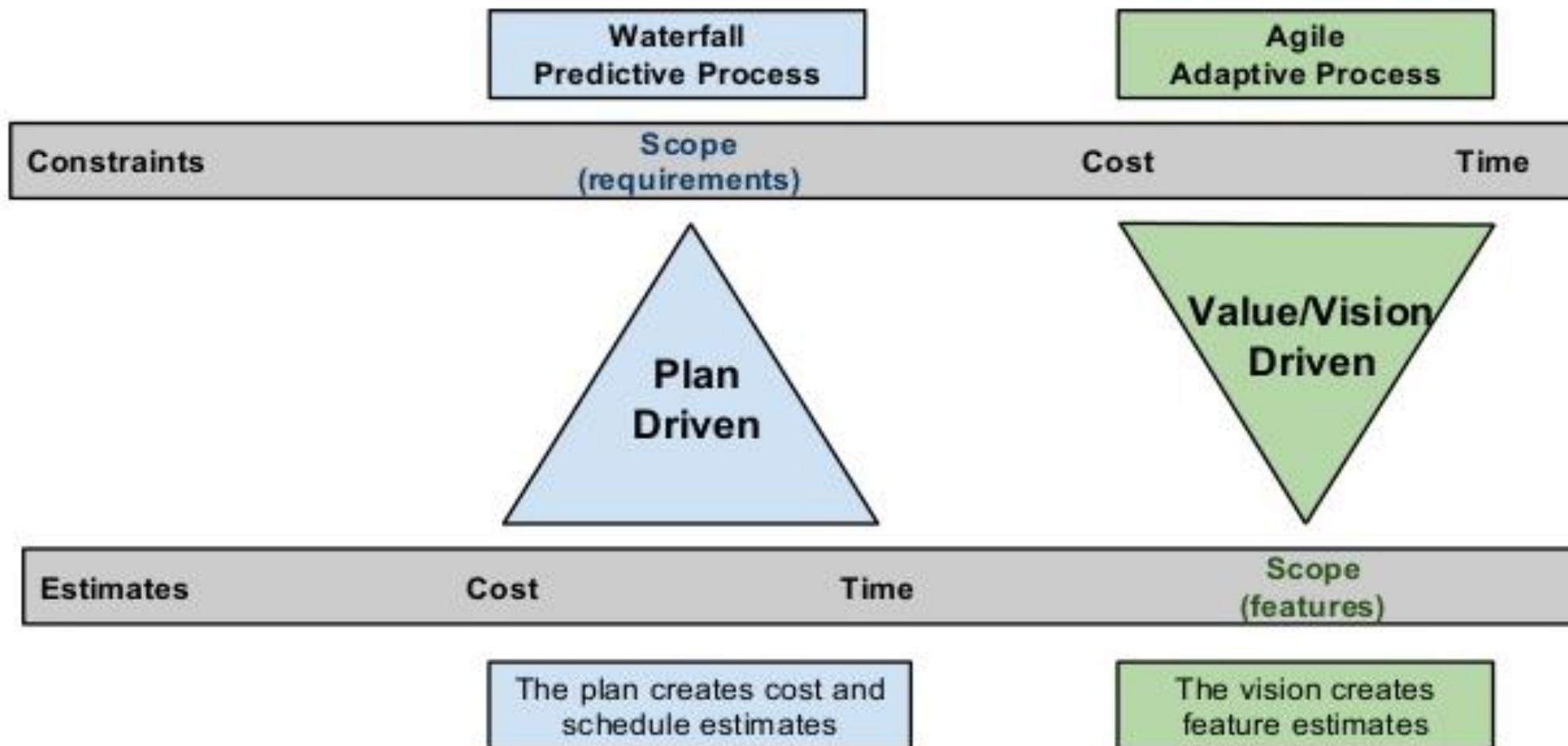
Agile



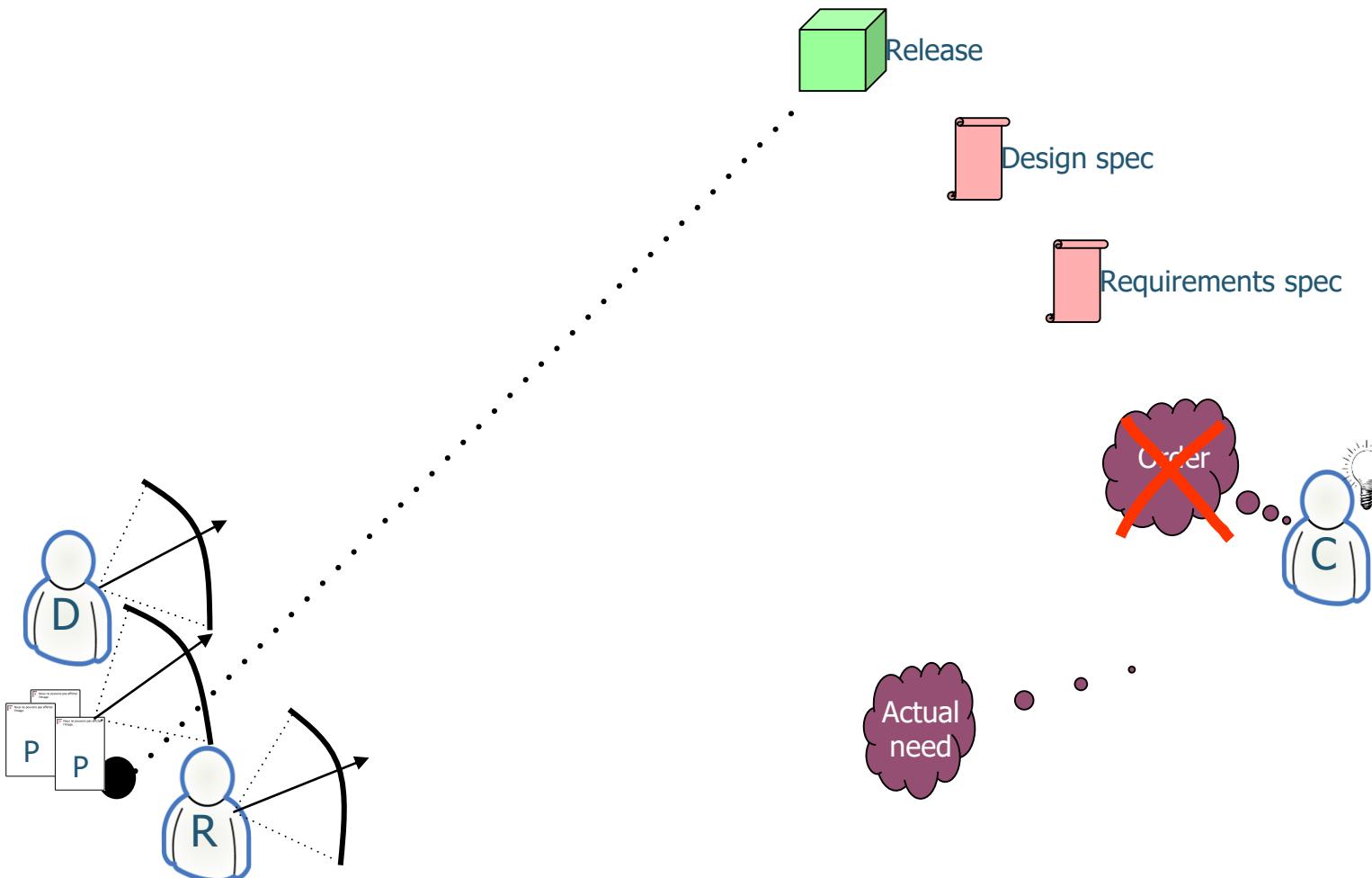
Agile

- 2001: « **Agile Alliance** » définit les méthodes Agiles
 - But:** augmenter le niveau de satisfaction du client tout en rendant le travail de développement plus facile
- 2 caractéristiques fondamentales:
 - «**Adaptatives** » plutôt que prédictives
 - Être favorable aux changements
 - Suivre un formalisme léger → Planification plus souple
 - Orientées vers des **personnes** plutôt que vers les processus
 - Adopter un **esprit collaboratif**
 - Travailler avec les **spécificités de chacun**

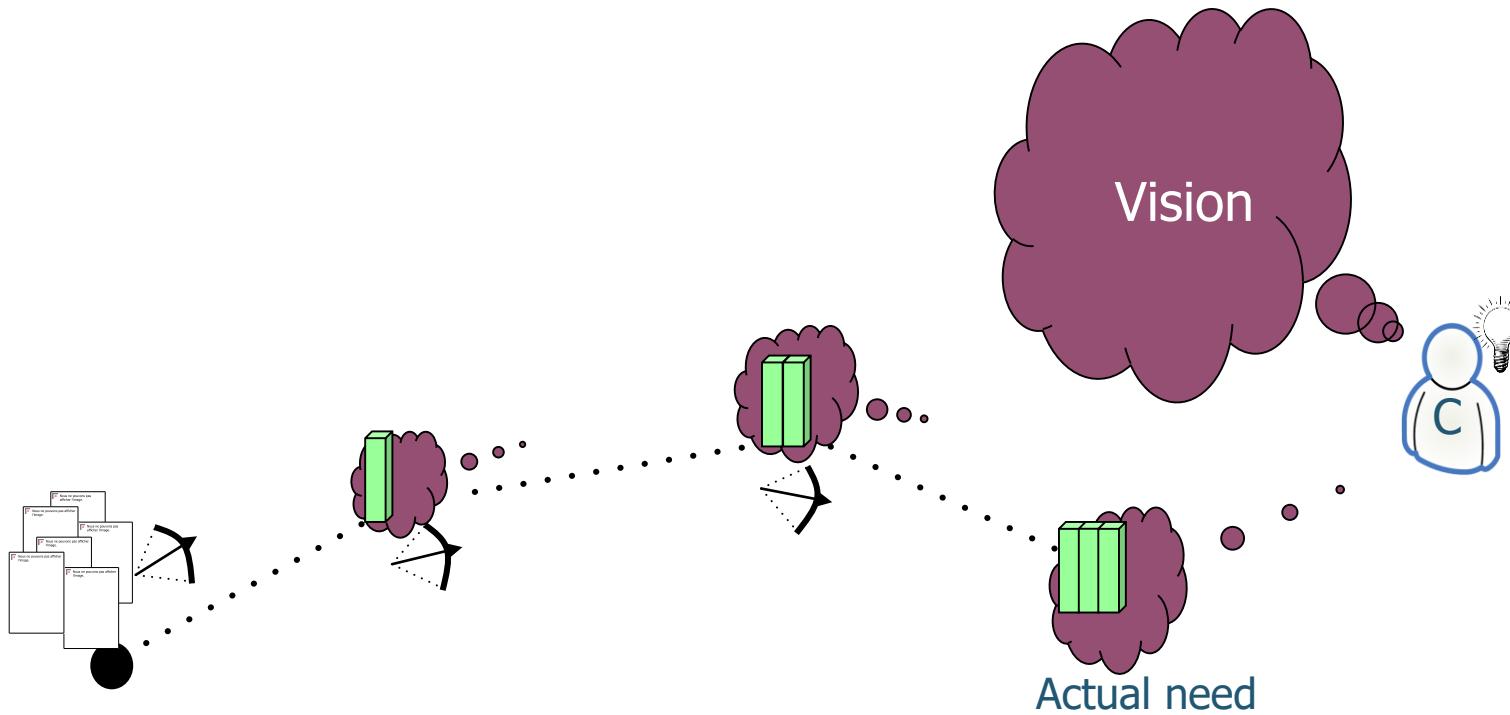
Prescriptive vs. Adaptive



Predictive approach

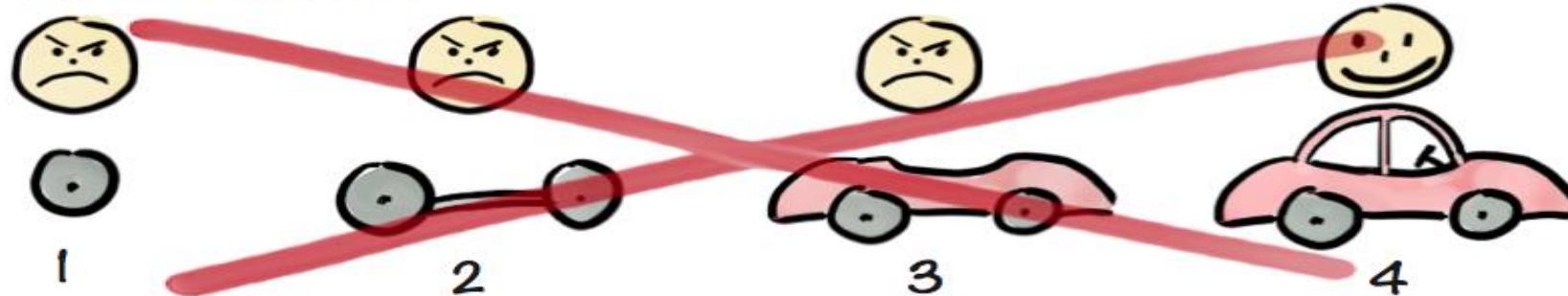


Adaptive approach

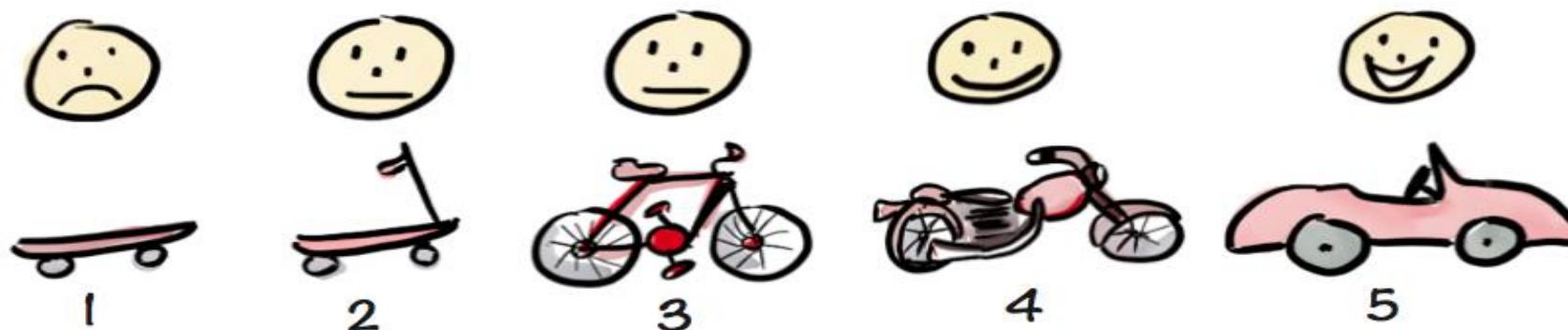


Adaptive approach

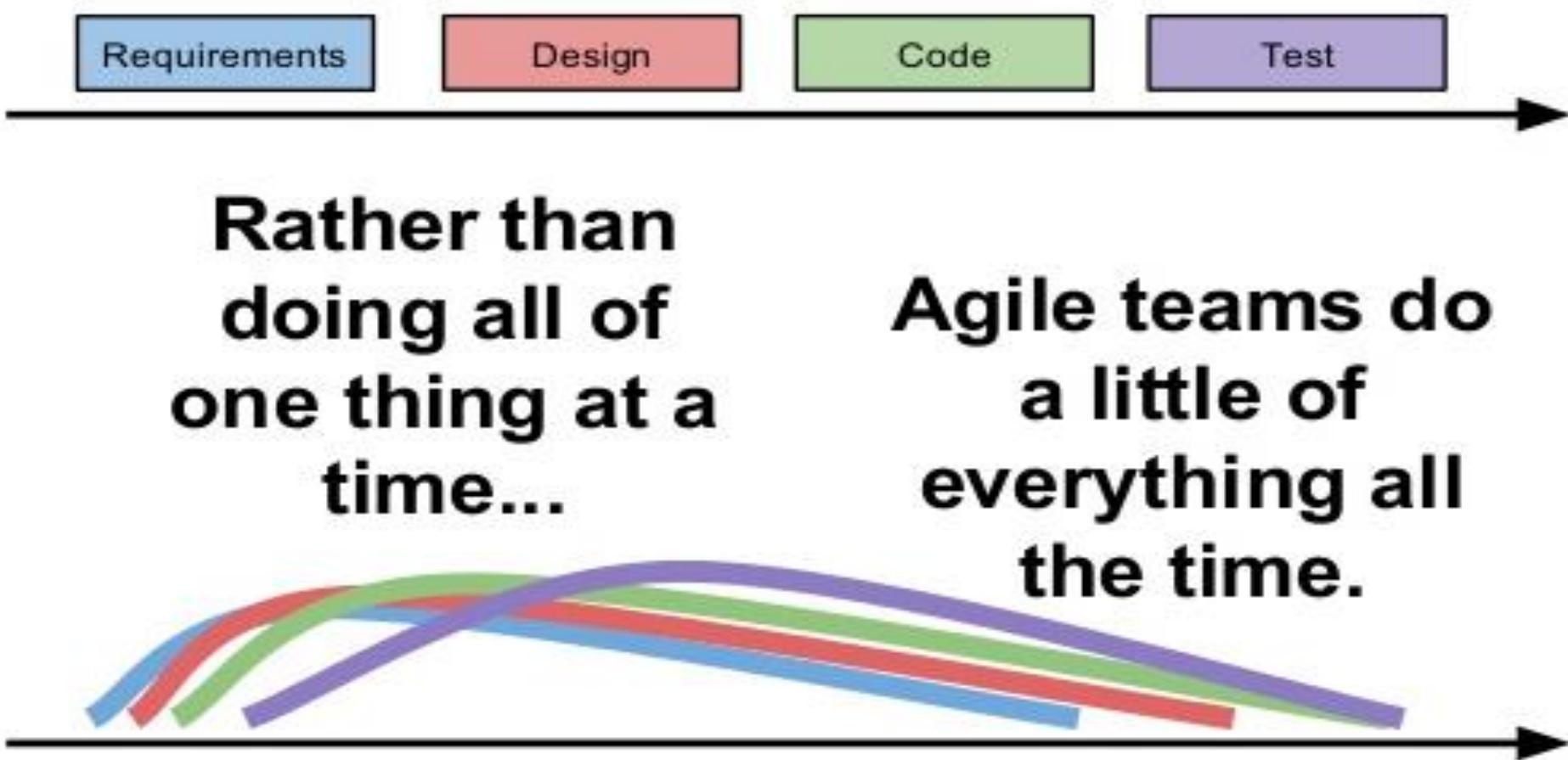
Not like this....



Like this!

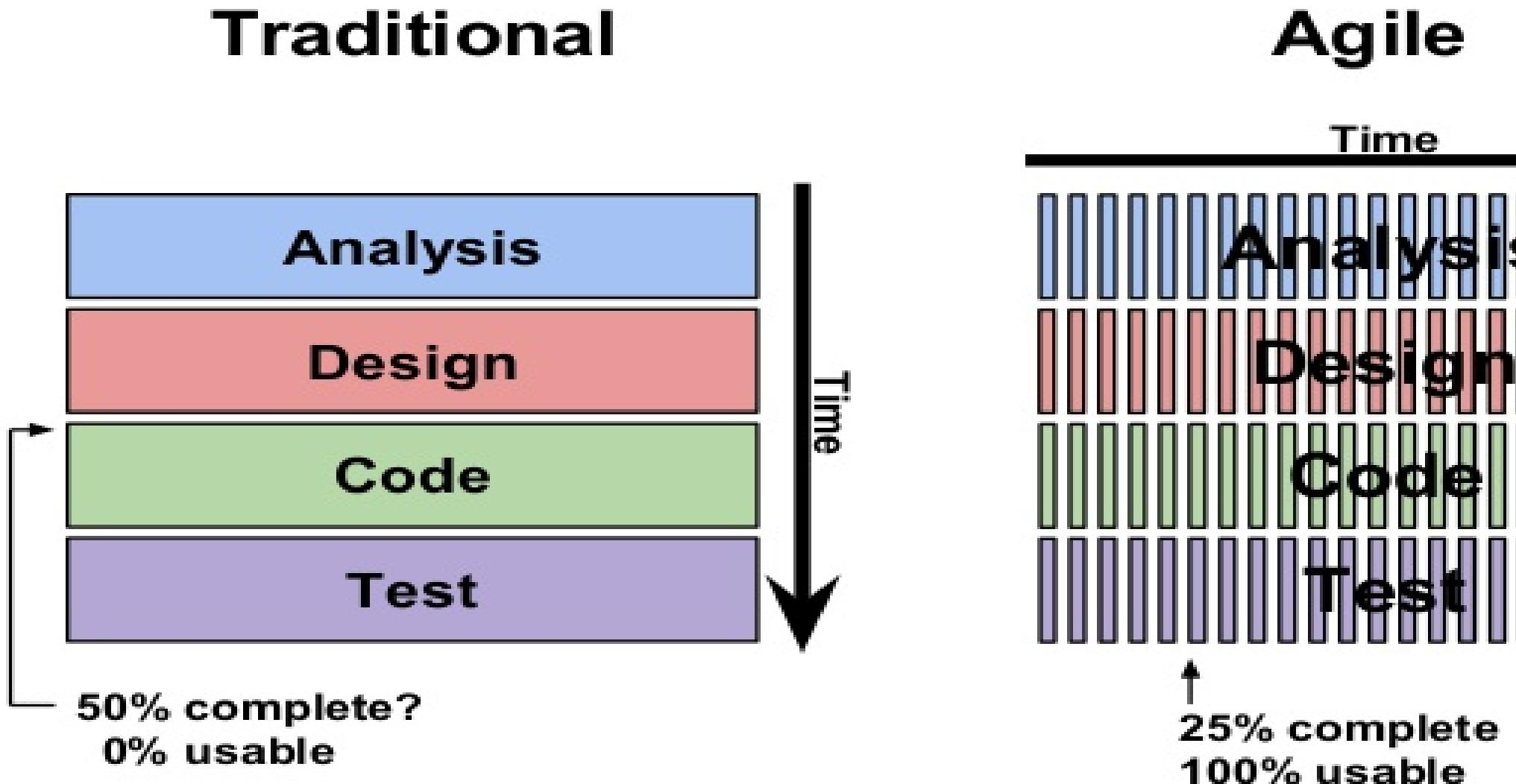


Sequential vs. Overlapping development



Source: "The New New Product Development Game" by Takeuchi and Nonaka. Harvard Business Review, January 1986.

Software development process



What is Agile?

The english term agile means

Light

Alerte

Quick

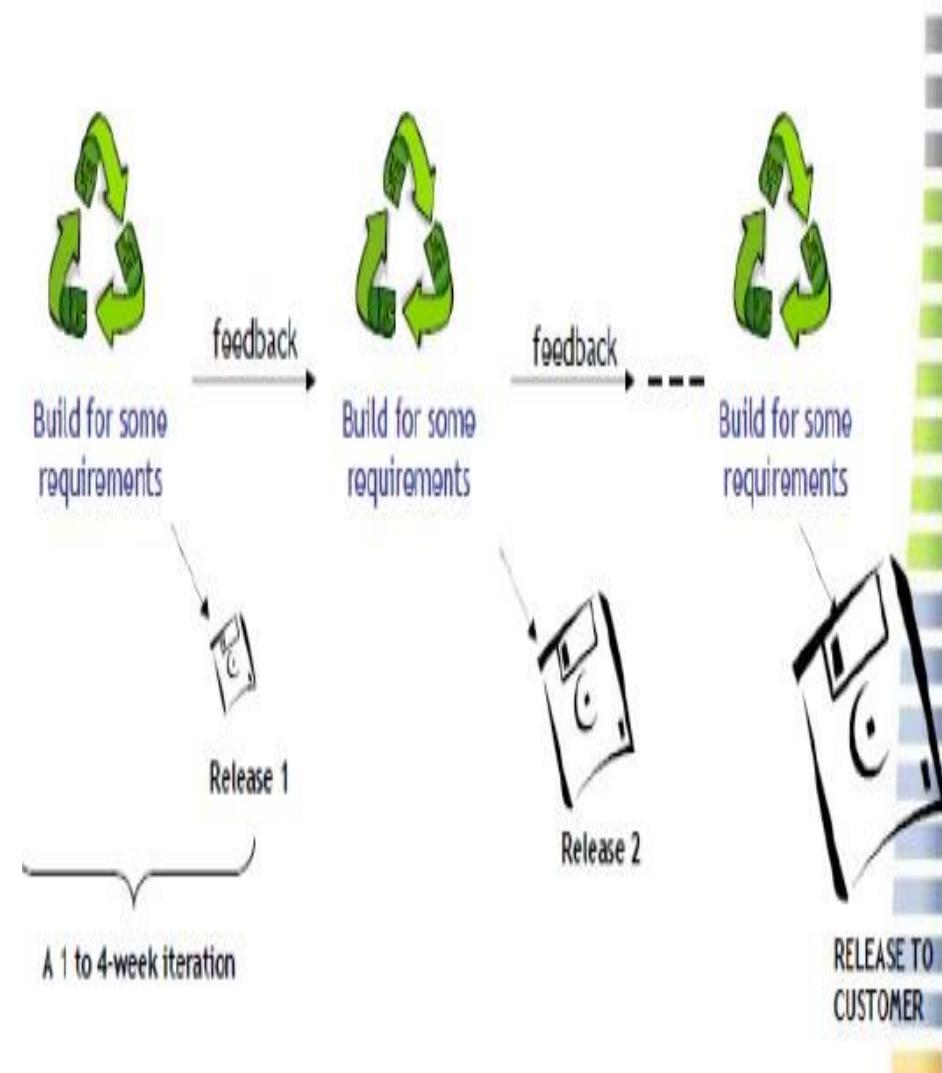
Ability to move or think easily and quickly

Agile software development

Has no definition

Umbrella term for similar projects

The term coined in 2001



Manifeste Agile

Nous sommes arrivés à préférer et favoriser:

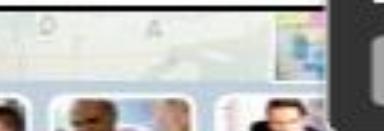
Les individus et leurs interactions plus **que les processus et les outils**

Du logiciel qui fonctionne plus **qu'une documentation exhaustive**

La collaboration avec les clients plus **que la négociation contractuelle**

L'adaptation au changement plus **que le suivi d'un plan**

Cela signifie que bien qu'il y-aït de la valeur dans les items situés à droite, notre préférence se porte sur les items qui se trouvent sur la gauche.



Les 12 principes du manifeste agile

Les 4 valeurs sont déclinées en 12 principes généraux

Satisfaire le client

Considérer comme naturel les changements d'exigences

Livrer fréquemment une application fonctionnelle

Fonctionnels et développeurs travaillent ensemble

Bâtir le projet autour de personnes motivées

L'échange d'information le plus efficace est en face à face

Un logiciel fonctionnel est la meilleure façon de mesurer l'avancement

Le rythme de développement doit être soutenable indéfiniment

Simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle

Architectures, spécifications et conceptions issues d'équipes auto-organisées

Vérifier en continue l'excellence des pratiques et techniques

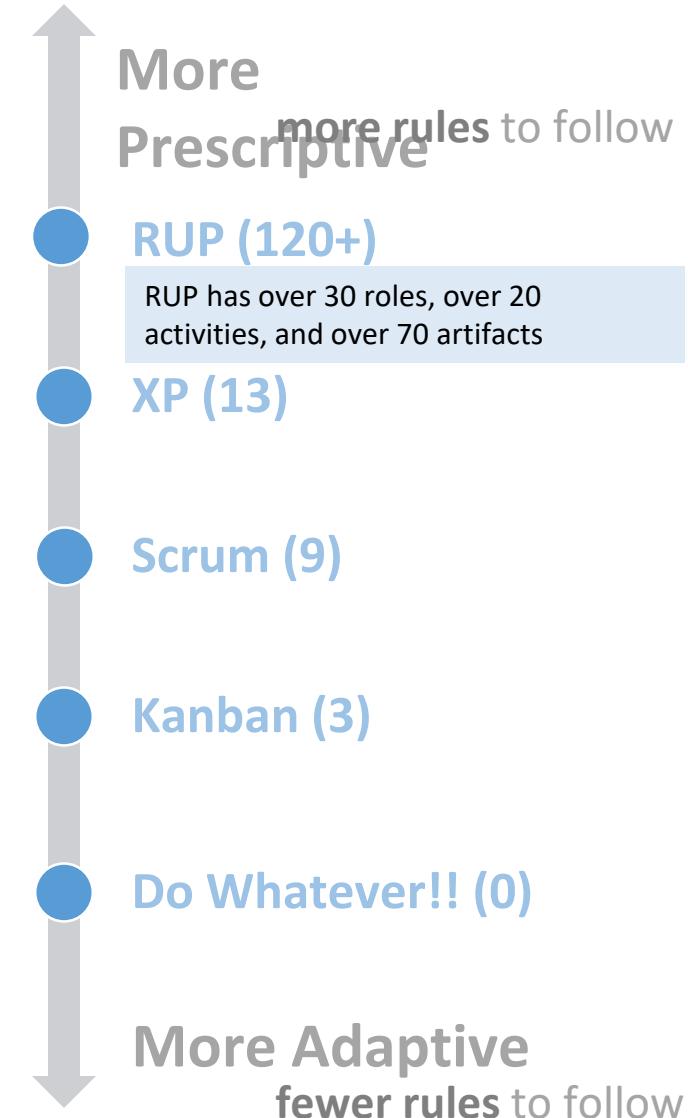
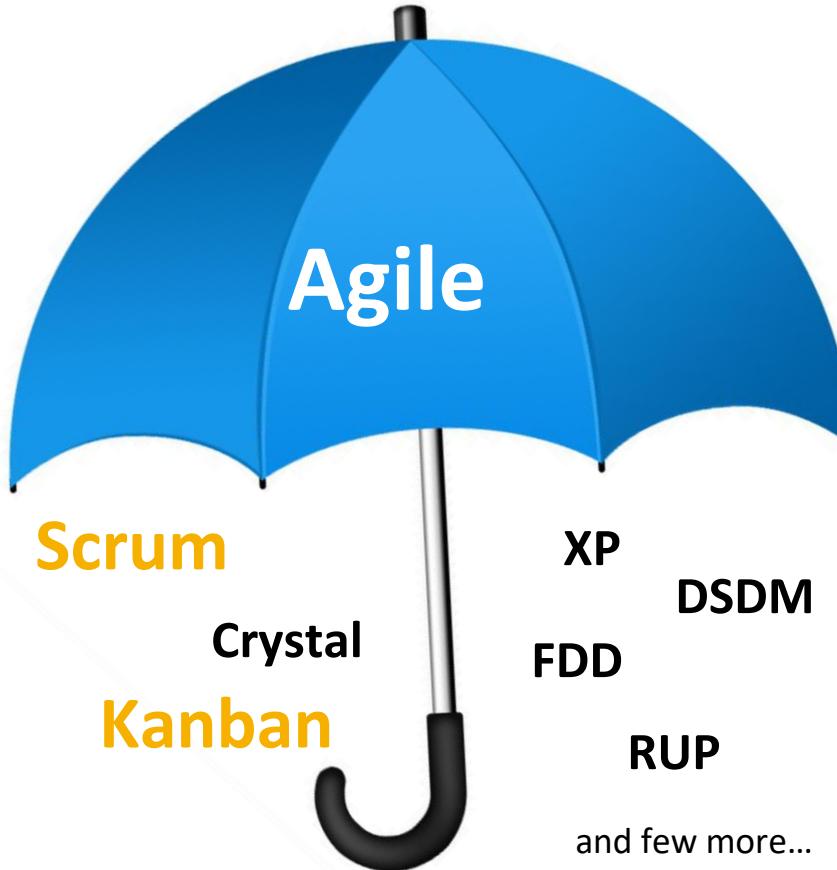
Régulièrement, réflexion de l'équipe pour être plus efficace !



Synthèse des différences fondamentales entre approche traditionnelle et approche agile

Thème	Approche traditionnelle	Approche agile
Cycle de vie	sans rétroaction possible, phases séquentielles	Itératif et incrémental
Planification	Préditive, basée sur des exigences définies dès le début du projet	Adaptive avec ajustement si nécessaire
Documentation	Produite en quantité importante comme support de communication	Réduite au profit d'incrément opérationnels
Équipe	Des ressources spécialisées dirigées par un chef de projet	Une équipe responsabilisée
Qualité	Contrôle à la fin du cycle de vie	Un contrôle continu
Changement	Résistance / opposition au changement	Accueil favorable des changements
Suivi d'avancement	Mesure de conformité aux plans initiaux	Le nbre de fonctionnalités implémentées, le reste à faire
Gestion des risques	Processus distinct, rigoureux de gestion des risques	Gestion des risques intégrée dans le proc global
Mesure de succès	Respect des engagements initiaux (coûts, délais et qualité)	Satisfaction client par la livraison de valeur ajoutée

Agile Umbrella



Etude agile

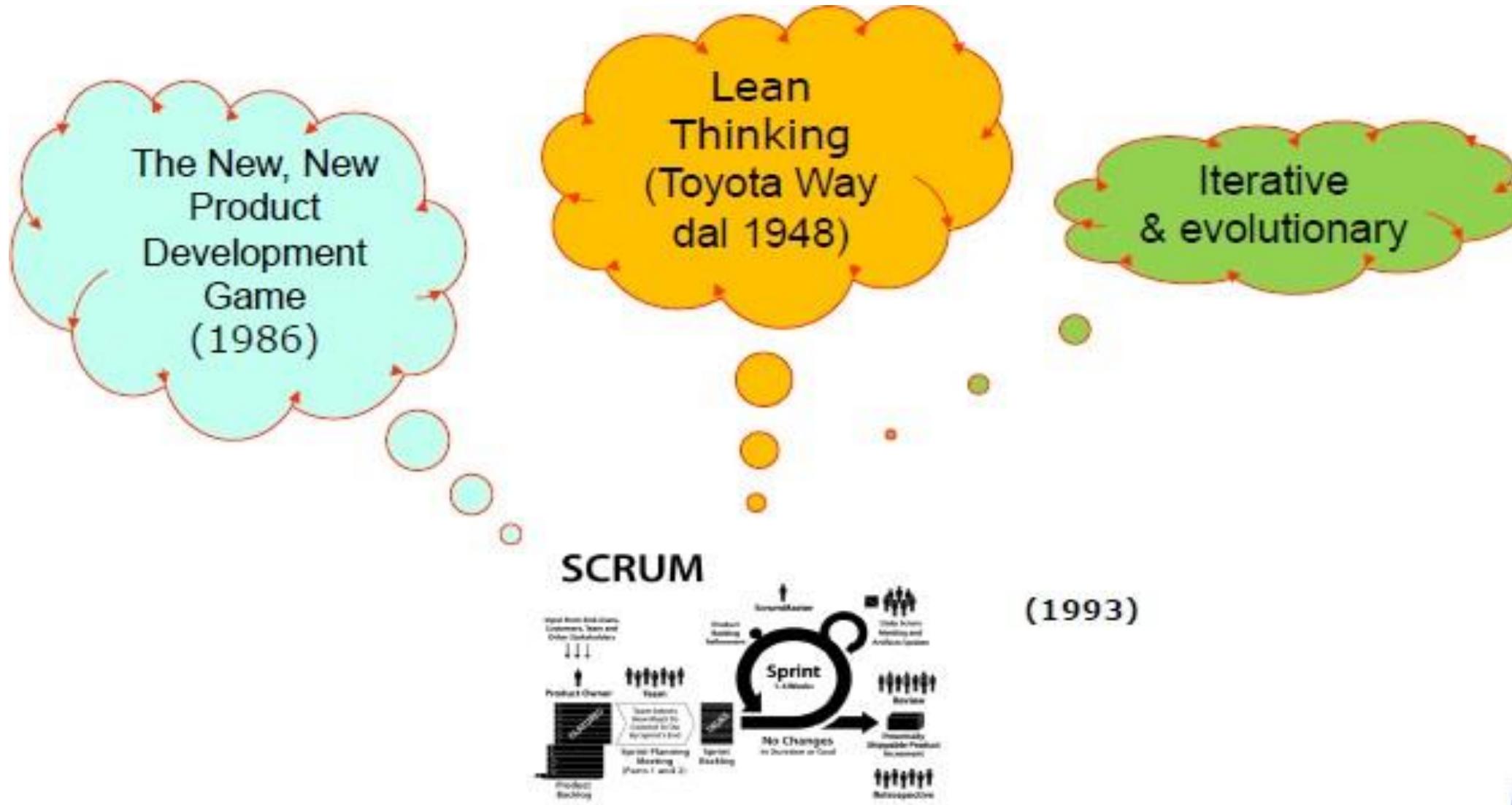
- Selon le bureau d'études Gartner, en 2012 :
 - **Plus de 60% des projets informatiques deviennent agiles**
 - **80% de ces projets utilisent la méthode Scrum**
- Ken Schwaber décrit Scrum comme un cadre (*framework*)
- Un spécialiste des processus parlerait, pour Scrum, de *pattern* de processus, orienté gestion de projet, qui peut incorporer différentes méthodes ou pratiques d'ingénierie.
- → **Scrum est une méthode agile pour la gestion de projets.**

Scrum



**SCRUM : produire au plus vite et
au plus près de vos priorités !!!**

SCRUM



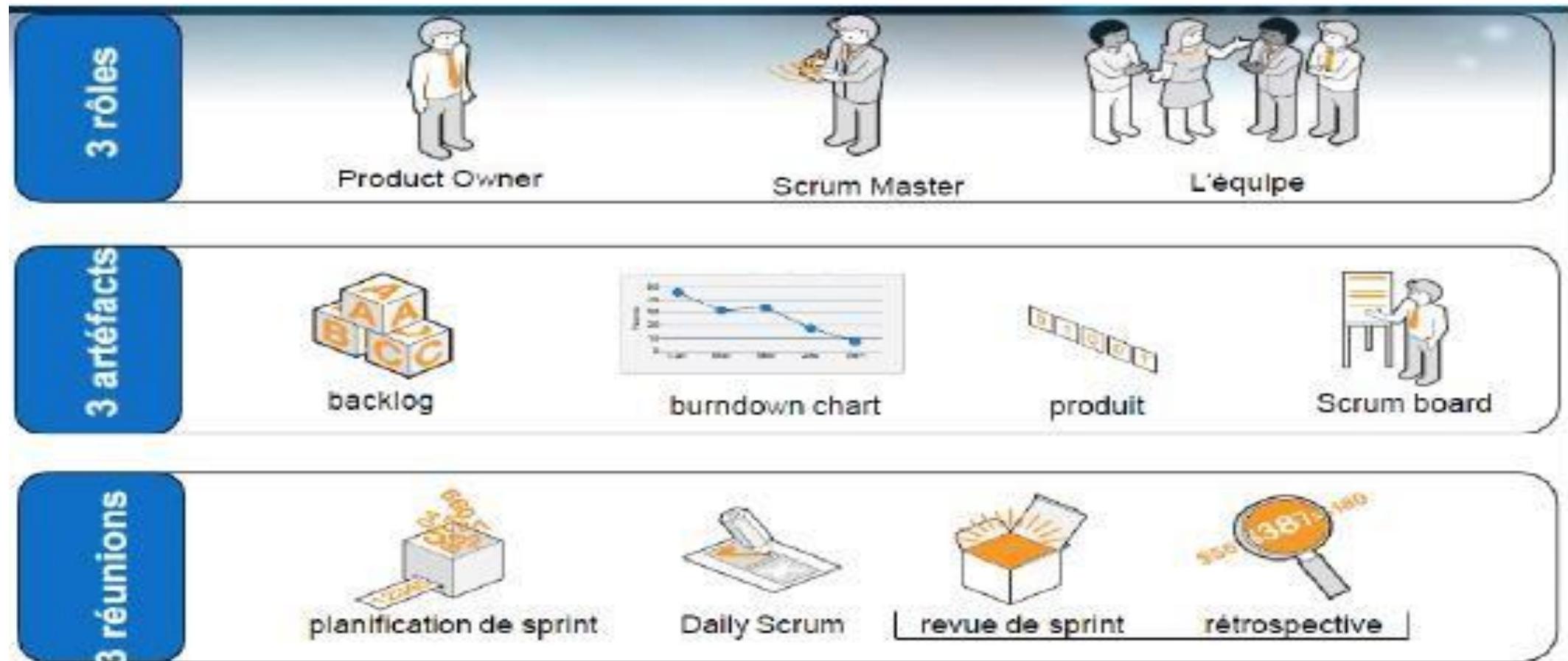
SCRUM

- Le terme Scrum est emprunté au rugby et signifie mêlée.
→ une équipe soudée, qui cherche à atteindre un but.

Scrum a été conçu pour améliorer la productivité dans les équipes auparavant paralysées par des méthodologies classiques - plus lourdes.

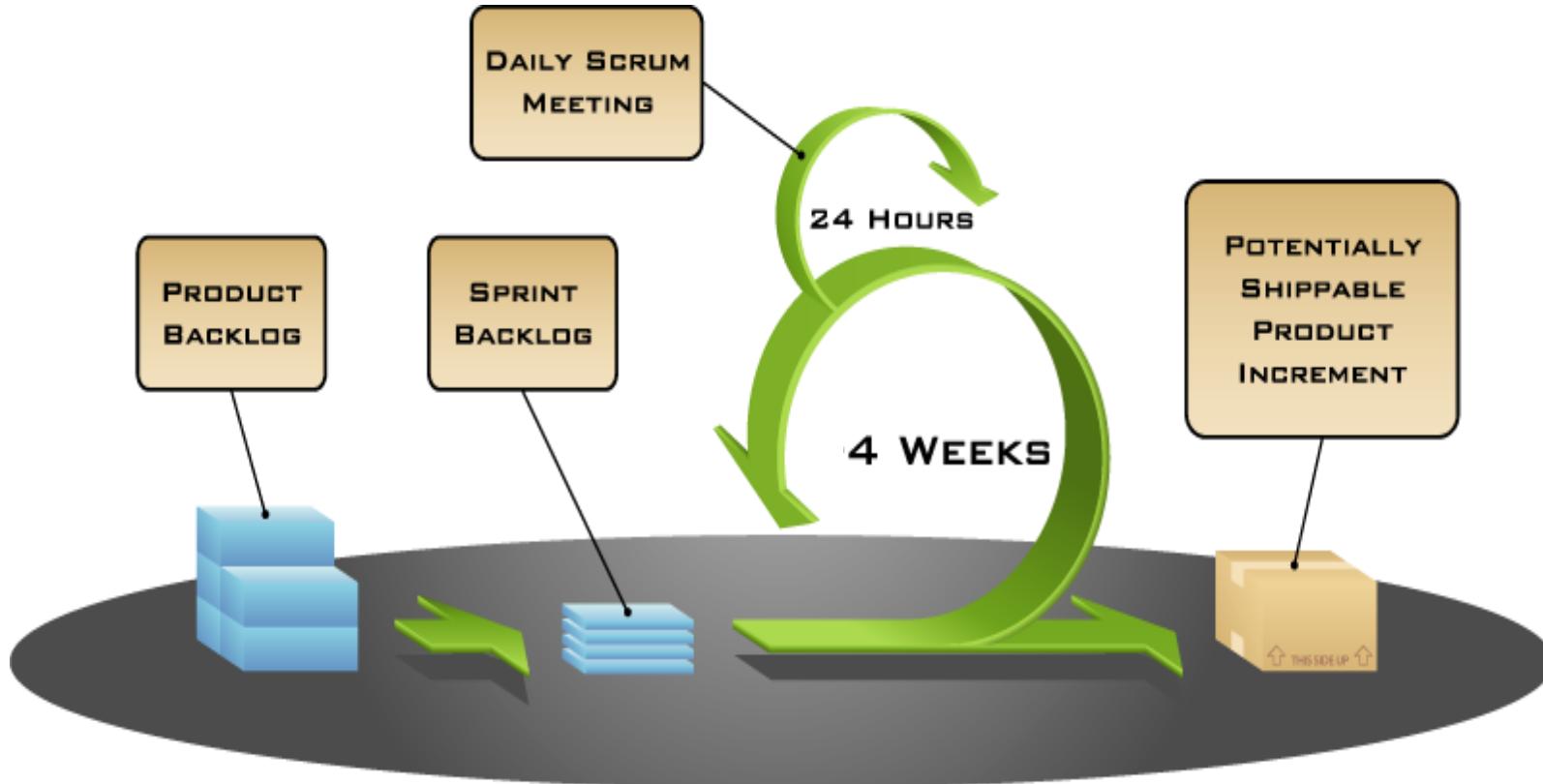


SCRUM Vue d'ensemble



Scrum

Process overview



COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

Scrum est

Transparence !
Inspection !
Adaptation !

Scrum est

- **Transparence**: garantit que tous les indicateurs relatifs à l'état du développement sont visibles par tous ceux qui sont intéressés par le résultat du produit.

Non seulement la transparence pousse à la visibilité mais ce qui est rendu visible doit être bien compris.

Par exemple, si un indicateur annonce que le produit est fini (ou une partie seulement du produit), cela doit être strictement équivalent à la signification de fini définie par l'équipe.

Scrum est

- **Inspection:** Les différentes facettes du développement doivent être inspectées suffisamment souvent pour que des variations excessives dans les indicateurs puissent être détectées à temps.
- **Adaptation:** Si l'inspection met en évidence que certains indicateurs sont en dehors des limites acceptables, il est probable que le produit résultant sera également inacceptable si on ne réagit pas ; le processus doit donc être ajusté rapidement pour minimiser les futures déviations.

Scrum est :

Il y a **trois points d'inspection et d'adaptation** dans Scrum :

- Le **scrum quotidien** permet d'inspecter la progression par rapport au but du sprint et de faire des adaptations qui optimisent la valeur du travail du jour suivant.
- La **planification et la revue de sprint** sont utilisées pour inspecter l'avancement du développement par rapport au but de la release et faire des adaptations sur le contenu du produit pour le prochain sprint.
- La **rétrospective** inspecte la façon de travailler dans le sprint pour déterminer quelles améliorations du processus peuvent être faites dans le prochain sprint.

Sprint /Release

- **Sprint (ou itération)** : un bloc de temps fixé aboutissant à créer un incrément du produit potentiellement livrable.
- **Release peut avoir plusieurs sens :**
 - Release comme version – Le dictionnaire du jargon informatique français définit une release comme suit : « version d'un logiciel effectivement diffusée, donc « Mise sur le marché ».
 - Release comme période de temps – Par extension, on appelle également release la période de temps qui permet de la produire.

Sprint /Release

- Différence entre incrémental & agile



- Une release est une série de sprints





* Sondage effectué en Mai 2009

Cycle de développement SCRUM

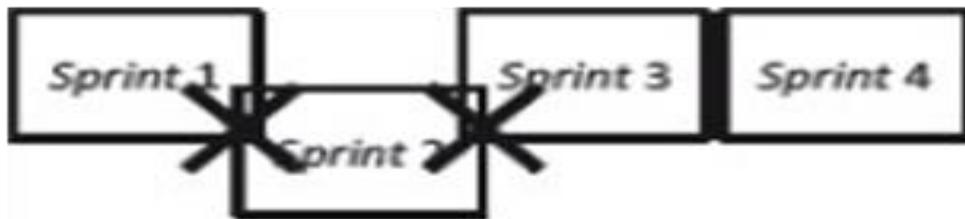
Une release est une série de sprints qui se termine quand les incrément successifs constituent un produit qui présente suffisamment de valeur à ses utilisateurs.

La durée des releases est définie par l'équipe et le Product Owner.

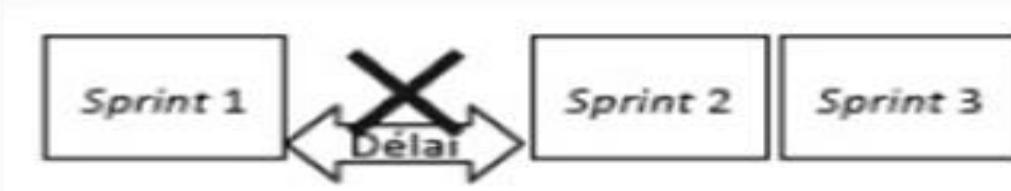
La tendance est à raccourcir ces durées : une release dure environ trois mois, avec des sprints de 2 ou 3 semaines.
idéalement, dérouler de 4 à 6 sprints dans une release.

Cycle de développement SCRUM

- Les sprints sont **séquentiels** : Il n'y a pas de chevauchements



- Les sprints s'enchaînent **sans délai** : le nouveau démarre immédiatement après le précédent.



Résultat d'un sprint

- Le résultat attendu = un incrément du produit final, qui est **potentiellement livrable**.
 - Le produit ne doit pas être seulement « potentiellement » utilisable à la fin d'un sprint, il doit simplement être utilisable.
 - Certes, le produit n'est pas complet par rapport à ce qui est prévu dans la release, mais il est livrable, au moins à des utilisateurs privilégiés.
- à la fin du sprint, déployer le produit potentiellement livrable, avec si nécessaire, la documentation permettant de l'utiliser.

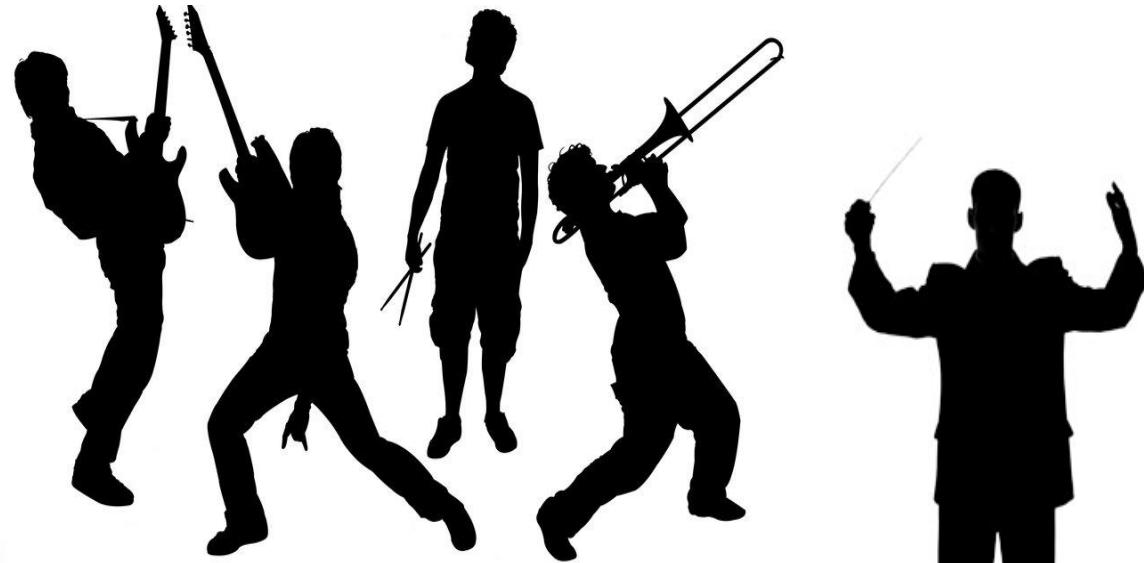
Résultat d'une release

- Le résultat attendu = un incrément du produit final, qui est **potentiellement livrable**.
 - Le produit ne doit pas être seulement « potentiellement » utilisable à la fin d'un sprint, il doit simplement être utilisable.
 - Certes, le produit n'est pas complet par rapport à ce qui est prévu dans la release, mais il est livrable, au moins à des utilisateurs privilégiés.
- à la fin du sprint, déployer le produit potentiellement livrable, avec si nécessaire, la documentation permettant de l'utilis₃e₆r.

Roles



Propriétaire
du produit



L'équipe

Scrum
master

Rôle dans Scrum



By Clark & Vizdos

© 2006 implementingscrum.com

La morale de cette histoire c'est que :

Vis à vis d'un projet, on n'a pas tous le même niveau d'investissement.

Rôle dans Scrum



Les Rôles de l'Équipe
Scrum

- L'équipe
- Le ScrumMaster
- Le Product Owner



Les Rôles
organisationnels

- Le Management
- Le Client
- Les Utilisateurs

Rôle dans Scrum

- A cause de cette histoire, on appelle souvent :
 - « **pig / cochon** » les personnes qui participent réellement au projet et qui s'y investissent
 - « **chicken / poule** » les personnes qui interagissent avec le projet mais qui ne sont pas réellement investies dans sa réalisation.

Le client

- **Sa fonction :**

- Il demande le produit
- Il contracte l'organisation pour le développement de son produit
- Typiquement, il s'agit d'un responsable qui achète un développement de produit par un sous-traitant.
- Dans les projets internes, il s'agit principalement du sponsor au projet, c'est à dire la personne validant le projet et le budget.

- **Sa mission :**

- Il commande le produit
- Il paie le développement du produit
- Il donne des **feed-back et des révisions**

Les rôles organisationnels - Scrum :

- Le client
- Le Manager
- Les utilisateurs finaux

} Stakeholders

Le manager

- **Sa fonction :**

- Le management, la gestion, est primordial dans tout projet Scrum. Il permet à l'Équipe de constituer un environnement optimal pour le déroulement du projet Scrum.
- Le manager donne de la structure et de la stabilité.
- Il travaille de concert avec le ScrumMaster pour réorganiser l'organigramme de la structure et donner de la guidance si nécessaire.

- **Sa mission :**

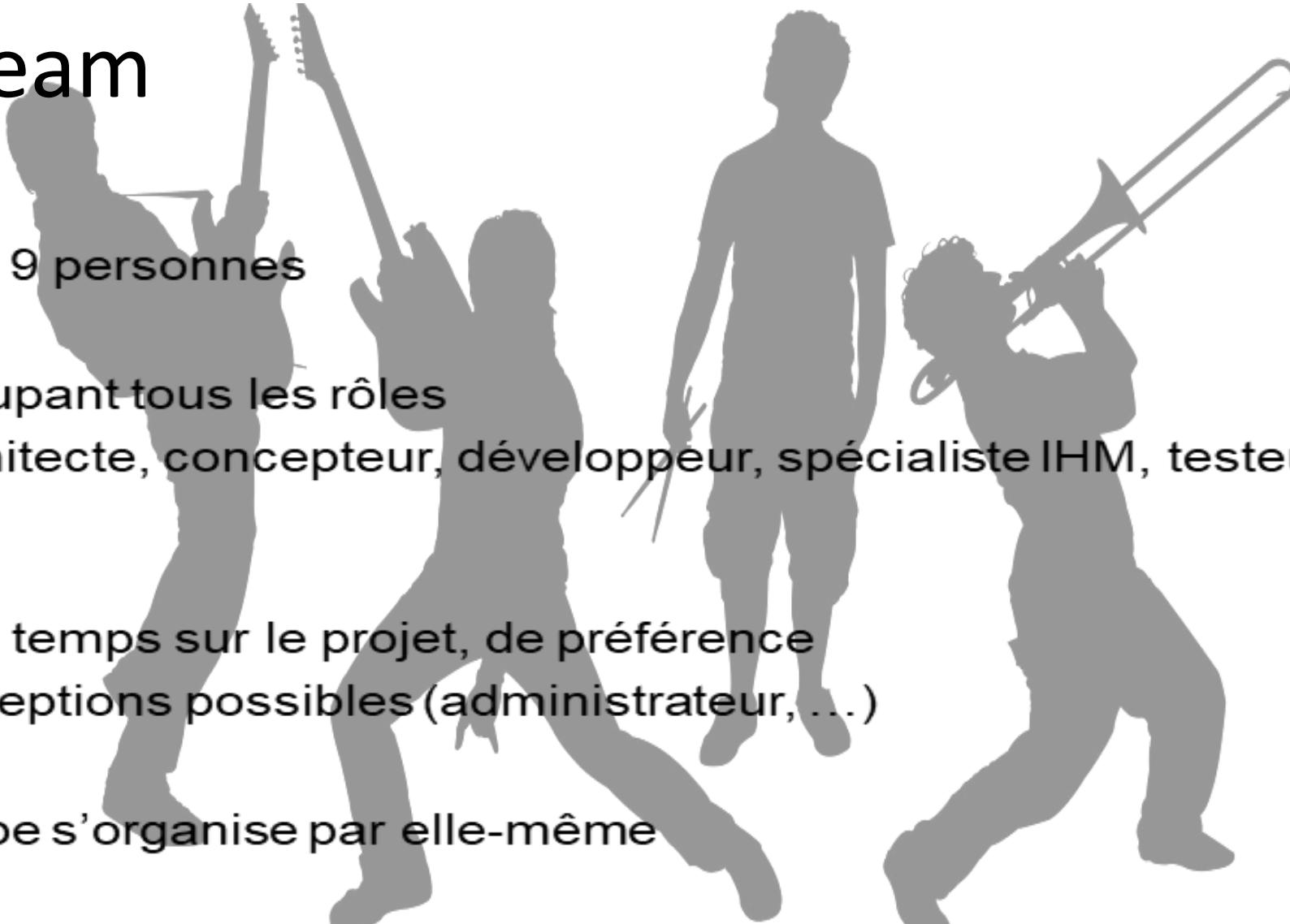
- Il s'assure que l'organisation puisse survivre en cas de défaillance.

L'utilisateur finale

- **Sa fonction :**
 - Ce rôle peut être joué par un grand nombre de personnes.
 - L'utilisateur final est celui qui connaît les besoins et avec cette connaissance, il définit le produit en disant à l'équipe ce dont il a besoin comme fonctionnalités.
- **Sa mission :**
 - Il connaît ses besoins et ses exigences
 - Il donne son **feed-back lors des revues**
 - Il participe, aussi, au Sprint Planning 1

The team

- De 5 à 9 personnes
- Regroupant tous les rôles
 - Architecte, concepteur, développeur, spécialiste IHM, testeur, etc.
- A plein temps sur le projet, de préférence
 - Exceptions possibles (administrateur, ...)
- L'équipe s'organise par elle-même
- La composition de l'équipe ne doit pas changer pendant un



The team

- Le rôle de **l'équipe de réalisation (E.R.) est essentiel** : c'est elle qui va réaliser le produit, en développant un incrément à chaque *sprint*.
- Dans Scrum, l'équipe s'organise elle-même et doit avoir toutes les compétences nécessaires au développement du produit.
→ une équipe Scrum est multifonctionnelle.
- C'est l'équipe qui définit elle-même la façon dont elle organise ses travaux, ce n'est pas le ScrumMaster (ni le Product Owner).
- Chaque membre de l'équipe apporte son expertise, la ^{syn}ergie améliorant l'efficacité globale.

Feature Team



Scrum Master



Facilitator and leader

For the team and the Product Owner (PO)

Ensures that the process is followed

Remove *Impediments*

Shields the team from external interferences

Sees to that manual processes are automated

Improves the productivity of the team

Improves the engineering, practices and tools

Keeps information about progress up to date and visible

Scrum Master



- **Pas de chef de projet dans Scrum ! Le rôle est éliminé.**
- Le travail et les responsabilités d'un chef de projet ne disparaissent pas pour autant dans les projets Scrum.
→ une grande partie est dévolue au Product Owner, une autre partie est laissée à l'équipe.
- Un des principes de Scrum est l'auto-organisation
→ Pas besoin d'un chef qui assigne le travail à faire à l'équipe.
- **ScrumMaster n'est donc pas un nouveau nom pour chef de**

Scrum Master



- Le ScrumMaster a pour responsabilité essentielle d'aider l'équipe à appliquer Scrum et à l'adapter au contexte.
- Il a une grande influence sur la façon de travailler, sur le processus, comme le **Product Owner** en a une sur le produit.
Le **ScrumMaster** pourrait être qualifié de **Process Owner** par équivalence.

Scrum Master



veiller à la mise en application de Scrum

encourager l'équipe à apprendre, et à progresser

faire en sorte d'éliminer les obstacles qui pourraient freiner l'avancement

Faire en sorte que les différentes réunions aient lieu et qu'elles se fassent dans le respect des règles

inciter l'équipe à devenir autonome.

Protéger l'équipe des interférences extérieures pendant le déroulement d'un sprint



Scrum Master

Avant le premier sprint :

- Le S.M. est souvent la première personne désignée dans l'équipe.
- Il peut donc être impliqué dans la constitution de l'équipe.
- Il arrive qu'il **participe au choix du Product Owner**.
- Il s'assure que la logistique est adaptée aux pratiques de travail en équipe.

Tâches périodiques pendant les sprints

- Il met Scrum en application.
- Il organise et anime les réunions qui constituent le cérémonial d'un sprint.
- Il fait en sorte que ces réunions aient lieu et qu'elles soient efficaces.
- Il y joue un rôle de facilitateur, littéralement «celui qui facilite les choses»

Scrum Master



Quelques traits permettent de déceler un ScrumMaster :

- la capacité à percevoir les émotions dans l'équipe,
- la curiosité et l'envie d'apprendre,
- l'inclination à penser que les gens font de leur mieux dans leur travail,
- l'envie de changer les choses qui ne vont pas bien,
- l'orientation vers le collectif,
- la prise de risques.

Product Owner



1. Fournir une vision partagée du produit :

- Le P.O. est responsable de définir l'objectif du produit et de le partager avec l'équipe qui le développe.
- Sa vision consiste typiquement à définir :
 - l'énoncé du problème que le produit veut résoudre,
 - une position du produit qui soit claire pour tout le monde,

Product Owner



2. Définir le contenu du produit :

- Le P. O. définit le contenu du produit
 - identifier les fonctionnalités requises + les collecter dans une liste, appelée **le backlog de produit**.
- Le P.O. est responsable du backlog et y contribue de façon régulière.
- Comme il est moteur pour établir ce que doit faire le produit, il est logique qu'il fournit aussi ses **conditions de satisfaction**, qui permettront de s'assurer que ce qu'il demande est bien réalisé
→ il est donc impliqué dans les **tests d'acceptation**.
- En plus de son travail pour le sprint courant, il s'occupe aussi des éléments du backlog prévus pour les sprints suivants.

Product Owner



3. Planifier la vie du produit :

- Le P. O. définit l'ordre dans lequel les parties du produit seront développées.
→ Il doit alimenter l'équipe avec les fonctionnalités à développer, selon ses **priorités** définies en fonction de la valeur qu'elles apportent.
- L'ordre de réalisation définit le cycle de vie du produit. Cette vie est constituée d'une succession de versions (les releases).
- Le P.O. définit l'objectif d'une release et prend les décisions sur le contenu et la date de mise à disposition du produit (durant le

Product Owner



- La personne idéale pour jouer ce rôle devrait posséder les compétences suivantes :

Bonne connaissance du domaine métier

Maîtrise des techniques de définition de produit

Capacité à prendre des décisions rapidement

Capacité à détailler au bon moment

Esprit ouvert au changement

Aptitude à la négociation

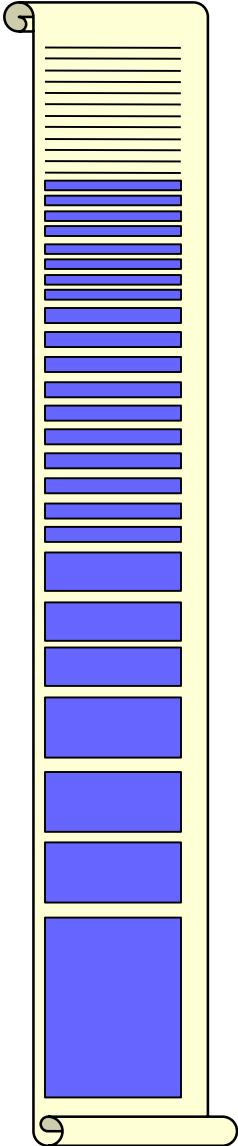
- Quelqu'un qui a été Analyste Métier (Business Analyst) est un bon candidat pour ce rôle.

Product Owner



- On peut se baser sur les compétences souhaitées du Product Owner déjà présentées.
- En effet, cette personne doit être :
 - Une personne disponible :
 - disponibilité continue,
 - participation aux différentes cérémonies Scrum,
 - implication régulière : MAJ le backlog, ajuster les priorités, répondre aux questions, définir et faire les tests d'acceptation
 - Une personne motivée pour ce rôle

Product backlog



- Au départ, la difficulté fondamentale est de transformer l'idée de départ en quelque chose d'utilisable par l'équipe de réalisation (ER).
- Dans les projets traditionnels, cette transformation se fait entièrement au début du projet et se concrétise dans un document, qui décrit :
 - ce que va faire le produit,
 - quelles sont ses fonctions
 - quel est son comportement.

deviner et imaginer les détails du comportement du produit avant d'entreprendre aucune action

Product backlog

- Un backlog = un référentiel des exigences
- Un backlog = Liste ordonnée de toutes les choses à faire.
- Les éléments du Backlog sont appelés des stories



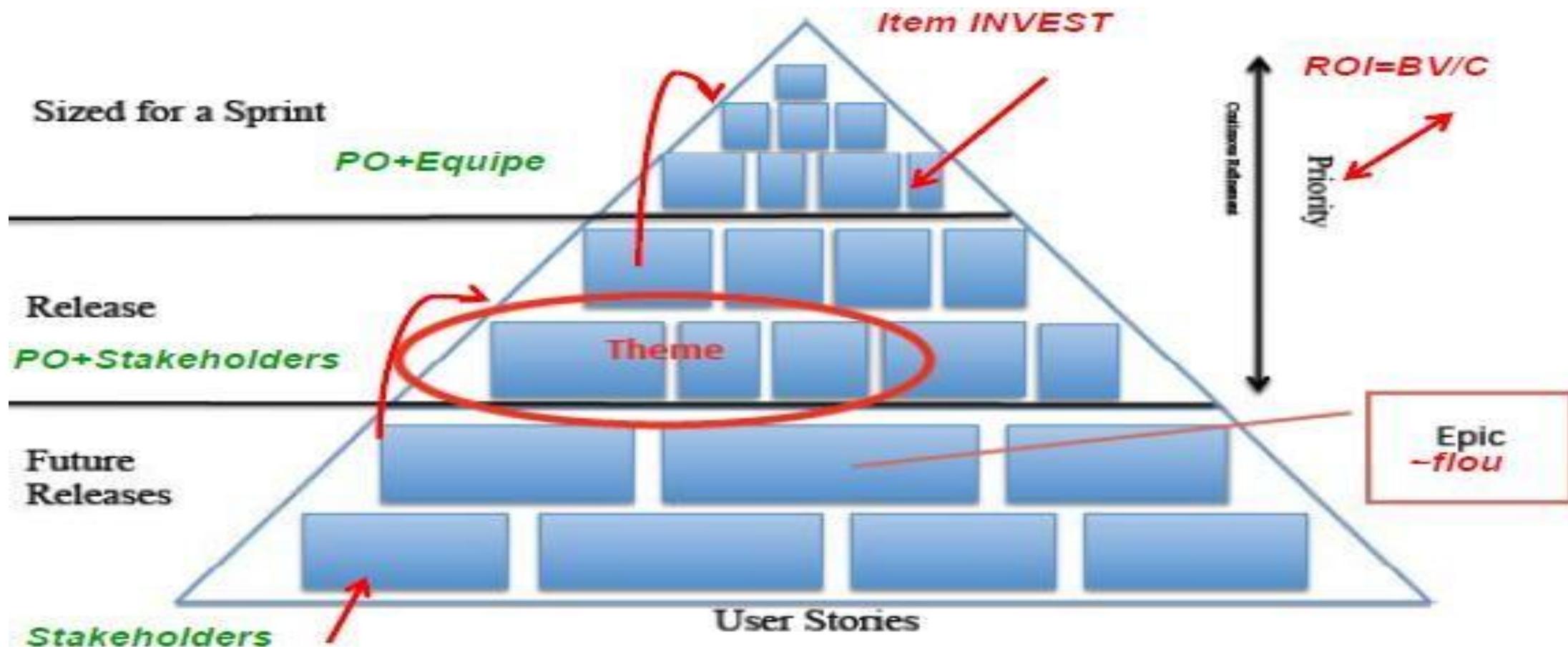
- On distingue :
 - le backlog de produit qui énumère les exigences avec le point de vue du client
 - le backlog de sprint qui contient les tâches de l'équipe.

Product backlog

- Les exigences
- Une liste de tout ce qui va entraîner du travail au projet
- Chaque élément doit apporter de la valeur aux utilisateurs ou clients du produit
- Les priorités sont définies par le directeur produit
- Les priorités sont revues à chaque sprint



Product backlog



Product backlog

- **I : Independant**

Ne dépend de rien (réduire les liens entre items)

- **N : Negociable**

Je n'ai pas une solution technique figée

- **V : Valuable pour le client**

En tant que Scrum trainer, je peux

- **E : Estimable**

Estimation en compléxité ou en temps idéal

- **S : Small**

A définir en interne (2/3 jours)

- **T : Testable**

Validation de l'item

User story

- Une User story est une exigence du système à développer, formulée en une ou deux phrases dans le langage de l'utilisateur.
- Les User Stories émergent au cours d'ateliers de travail menés avec le Product Owner ou les Utilisateurs.
- Quelques Exemples :
 - “**En tant que recruteur, je peux déposer des offres d'emploi”;**
 - “**En tant que jeune diplômé, je peux créer un CV”;**
 - “**En tant que jeune diplômé je peux modifier un CV”;**
 - “**En tant que jeune diplômé je peux sélectionner des offres”...**

Product backlog

- Même si c'est le Product Owner qui en est responsable et qui définit les priorités, le backlog est partagé entre toutes les personnes de l'équipe.
- Le backlog est également visible des personnes extérieures à l'équipe qui sont intéressées par le développement du produit : clients, utilisateurs, managers, marketing, ...
favoriser la **transparence et faciliter le feedback**, qui se concrétise par l'ajout de nouvelles stories.

Product backlog

- Le backlog suit la vie du produit qu'il décrit, il **évolue** avec lui ; il peut donc avoir une durée de vie très longue, courant sur de nombreuses releases.
- **Émergence progressive**
 - Plutôt que d'essayer de tout figer dans le détail au début, le contenu du backlog est décomposé **progressivement**.
 - Dans un développement agile, le changement est **possible** et même **encouragé**. Il ne coûte presque rien, tant qu'il porte sur un élément pour lequel on a fait peu d'effort.

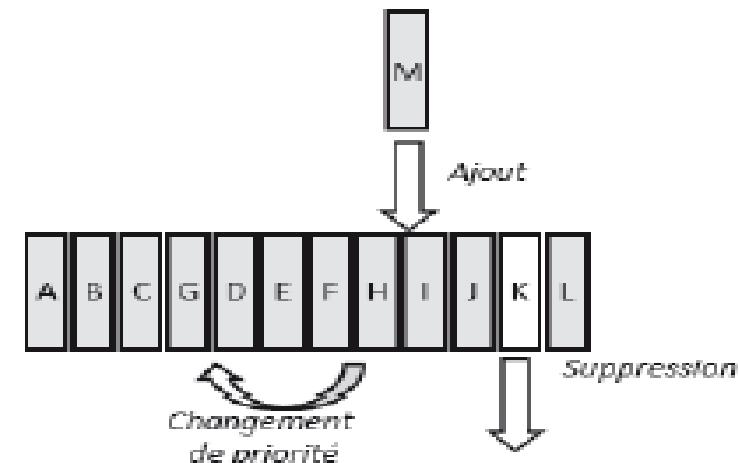
Product backlog

- **Changements continuels**

- Le backlog n'est pas un document figé, il n'est jamais complet ou fini tant que le produit vit encore,
- il évolue continuellement : des éléments sont ajoutés, des éléments sont supprimés, des éléments sont décomposés ou les priorités ajustées.

- **Attention :**

- ce changement continu n'a pas d'impact sur l'équipe qui développe ;
- pendant un sprint, la partie du backlog sur laquelle l'équipe travaille est gelée.



Product backlog

- **Utilisation continue**

- Le backlog est élaboré dans une forme initiale avant le lancement du premier sprint : il permet de planifier la 1ère release.
- Il sert pendant les sprints pour connaître les stories en cours de développement.
- **Par exemple**, lors de la réunion de planification en début de sprint, il est utilisé pour décider du sous-ensemble qui sera réalisé pendant le sprint.

Product backlog

- Le backlog est la liste unique de tout ce qui est à faire, ce qui donne beaucoup d'importance à la notion de **priorité**.
- Cette **priorité** permet de constituer le flux de stories qui va alimenter l'équipe. L'ordre peut changer tant que l'équipe n'a pas commencé à développer la Story
- Exemple :
 - Dire que la story A est plus prioritaire que la story B signifie que A sera réalisée avant B.
 - Les priorités sont utilisées pour définir l'ordre de réalisation

Product backlog

Parmi les critères qui poussent à donner une grande priorité à une story :

– **Le risque qu'elle permet de réduire**

- L'objectif est de réduire l'exposition au risque le plus rapidement possible.
- Des stories permettant de valider des choix techniques sont toujours prioritaires.

– **L'incertitude sur des besoins des utilisateurs qu'elle permettra de diminuer**

- Quand un utilisateur désire une fonctionnalité mais ne sait pas de quelle façon le service doit être rendu, la solution est de lui montrer rapidement une version pour obtenir du feedback → s'offrir une occasion d'améliorer le produit.

– **La qualité à laquelle elle contribue**

- Les travaux visant à garantir la qualité du produit devraient être prioritaires.

– **Les dépendances entre stories**

- Si une story A ne peut être développée que si une autre story B existe, la story B doit être plus prioritaire que A.

Les éléments du backlog

- Les principaux attributs

Story
<ul style="list-style-type: none">• Nom• Identifiant• Description• Type (<i>user, technique, défaut</i>)• État• Taille

- Les types d'éléments :

- **User story** : Décrit un comportement du produit visible pour les utilisateurs et qui leur apporte de la valeur ou de l'utilité.
- **Story technique** : Elle est invisible pour les utilisateurs mais visible par l'équipe de développement. Elle est nécessaire pour pouvoir développer certaines user stories, son utilité est donc *indirecte*.
- **Défaut** : Il est relatif à un comportement visible des utilisateurs mais qui enlève de la valeur au produit. En développement de logiciel, on parle couramment de **bug**.

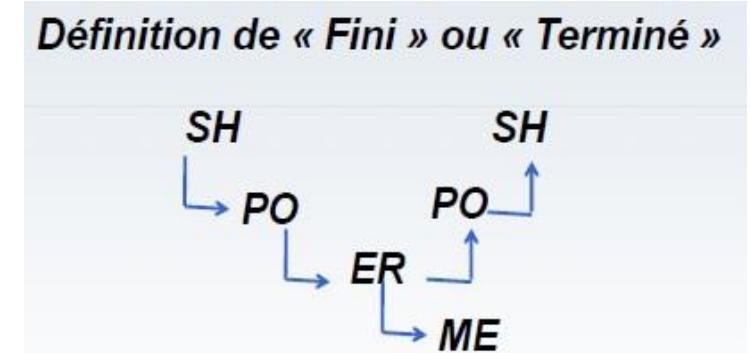
User story

- User Stories :
 - As a <actor>, I would like to <action>, so that <value>.
- Attributs :
 - Taille (points , journées idéales), Business Value (\$, H / M);
 - Conditions de satisfaction.
- La taille d'une User Story
 - Influencée par :
 - La difficulté de la Story
 - La charge de travail
 - Une valeur relative
 - Les points n'ont pas d'unités :
 - Suite de Fibonacci (0, 1, 2, 3, 5, 8, 13, 21....)

User Story



- **Créé** : par n'importe qui
- **Accepté** : par le Product Owner
- **Estimé** : par l'équipe dans une séance de planning poker
- **Planifié** : associé à un sprint futur lors de la planification de release
- **En cours** : développé dans le sprint courant
- **Fini** : terminé, selon la signification de fini



DFD

Pour l'équipe

- Le Code est conforme aux normes. Il doit être :
 - Propre
 - Refactoré
 - Testé unitairement
 - Validé (checked in)
 - Intégré (Built)
 - Dispose d'une suite de tests unitaires qui lui est appliquée.
- Pour arriver à cela, l'environnement de développement est constitué :
 - D'une bibliothèque de code source
 - De codes standards,
 - Build automatisé,



DFD

Pour Scrum

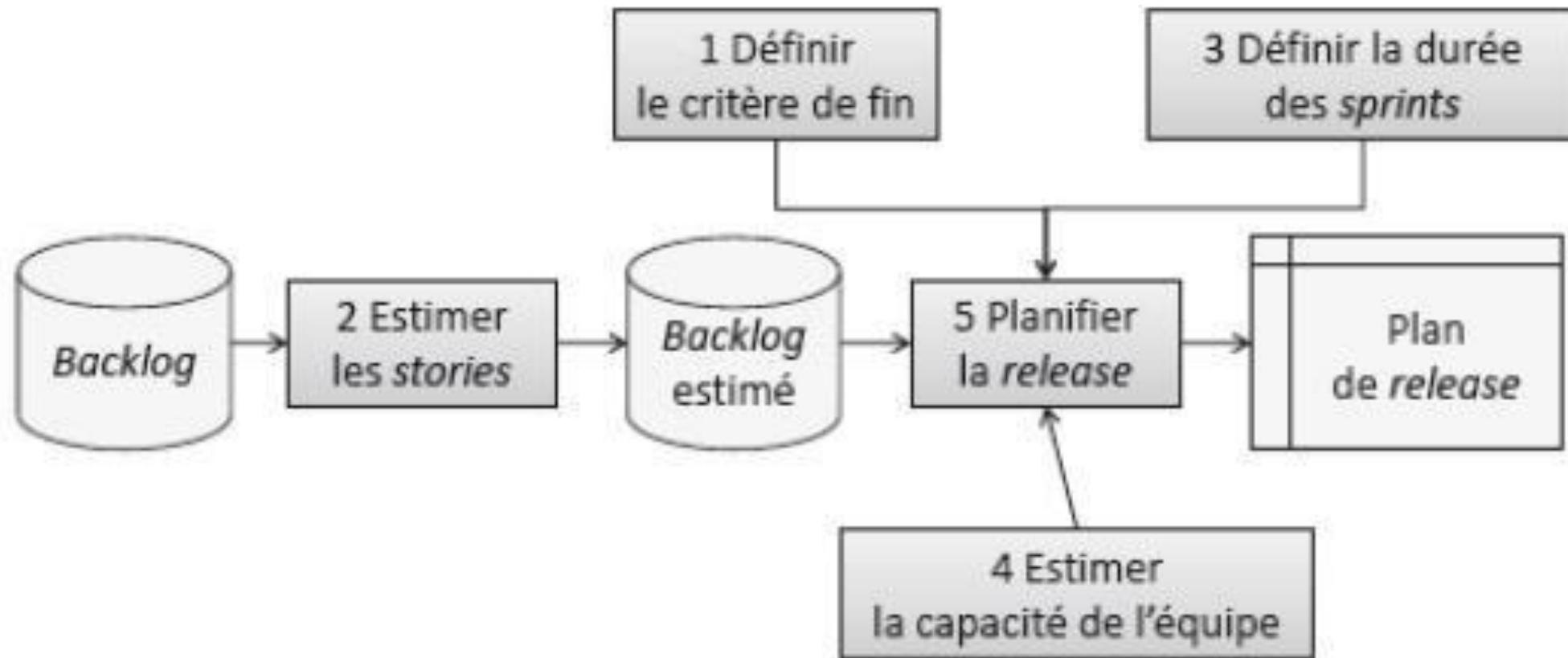
- Une Story/Item est « Finie / Done » lorsque l'équipe a déclaré qu'elle est « Terminée »
- Le Sprint/Iteration est “done” lorsque
 - tous les items sont “done”
 - et que le Sprint atteint son objectif
 - et que les critères d'acceptation sont adressés.
- La Release est “done”
 - “done” pour l'intégration
 - “done” pour la production

Sprint Planning Meeting

- Une **release** est la période de temps constituée de sprints utilisée pour planifier à moyen terme.
- Un **burndown chart** est une représentation graphique du reste à faire dans une période, actualisé aussi souvent que possible et permettant de montrer la tendance :
 - Un *burndown chart de sprint* est mis à jour **quotidiennement**,
 - Un *burndown chart de release* est **actualisé à chaque sprint**.

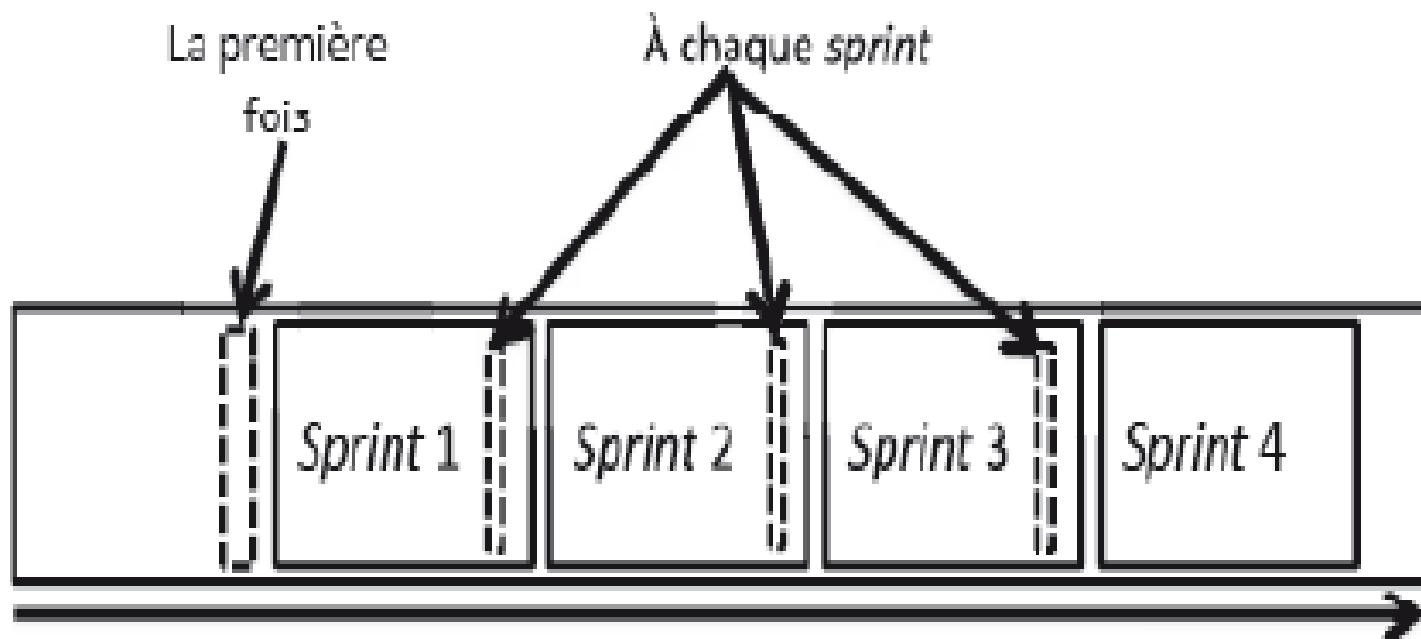


Processus de planification de release



Processus de planification de release

- Quand faire la planification de release ?



Processus de planification de release

Étape 1 : Définir le critère de fin de la release

- Pour décider de l'arrêt des sprints et de la fin d'une release, il existe deux possibilités :
 - Finir quand le backlog est vide (sans succès !!!???)
 - Fixer la date à l'avance :
 - elle donne un objectif précis, ce qui motive plus l'équipe ;
 - elle demande obligatoirement une réflexion poussée sur les priorités des éléments du backlog par le Product Owner ;
 - des éléments du backlog présentant finalement peu d'intérêt ne seront pas intégrés dans la release ;
 - on passe généralement moins de temps à estimer et planifier puisque la date de livraison est connue.
 - une organisation s'habituerà à cette fréquence, qui cadence le travail de l'équipe mais aussi celui des utilisateurs et de leurs représentants

Processus de planification de release

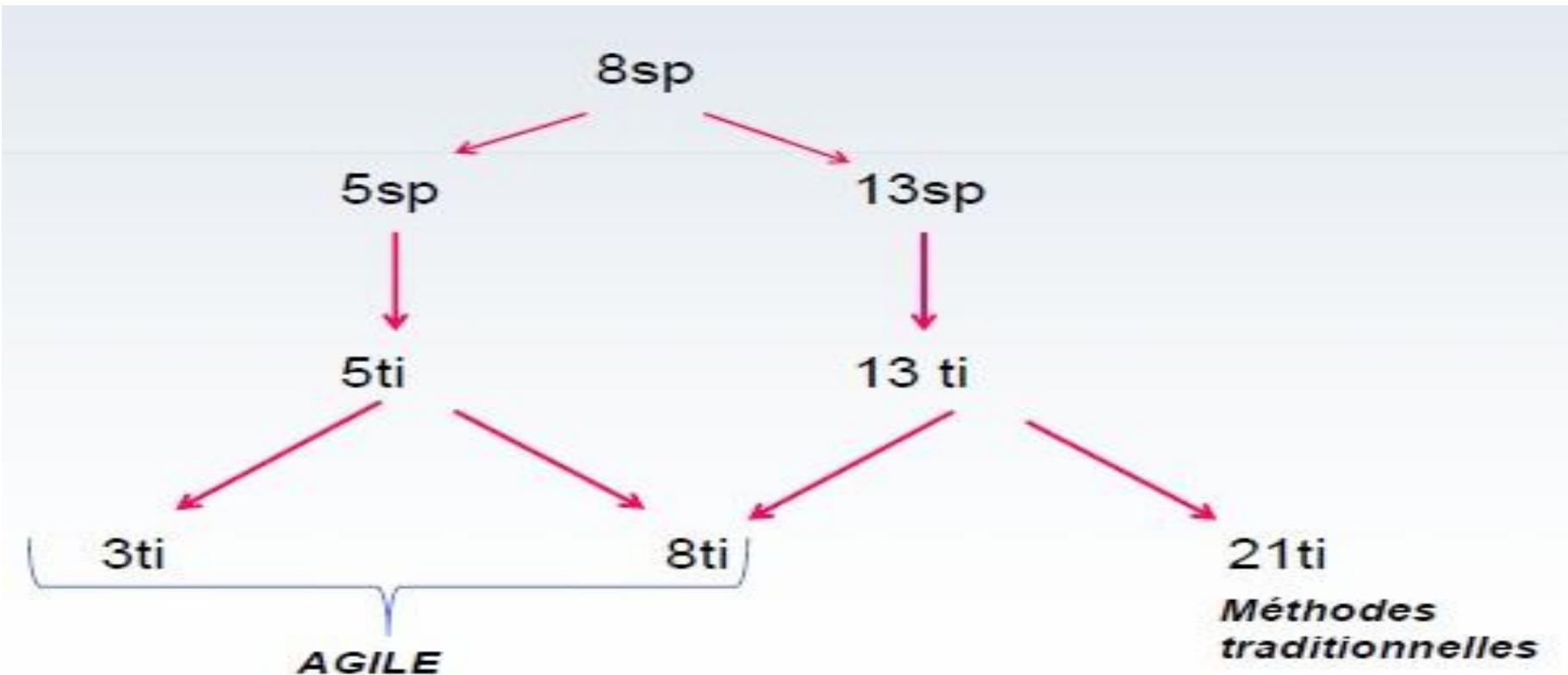
Étape 2 : Estimer les stories du Backlog

- Chaque story du backlog doit être estimée si on veut en tenir compte dans la planification.
- L'estimation dont il est question ici est celle relative à l'effort à fournir pour la développer.
- 3 techniques sont envisageables :
 - Triangulation (individuelle ou collective)
 - Avis d'experts
 - Planning Poker suivant la suite de Fibonacci
- L'usage le plus fréquent est de faire une estimation collective au cours d'une séance appelée **planning poker** et d'estimer la taille plutôt que la durée.

Sizing: Temps idéal

- Le TI varie en fonction de la suite de Fibonacci
- Réduire le Facteur d'impact:
 - motivation
 - Complexité
- LAPS: effort
 - on quantifie l'effort ssi on a une personne assignée

Sizing: Temps idéal



Sizing: Temps idéal

5 8 13 21

1. On demande à 5 pourquoi c'est si simple
En tant que SM, il doit faire une étude poussée
2. On demande à 21
3. Le SM demande aux autres membres de trouver une autre manière de faire ça
4. Le PO doit choisir entre plusieurs options:
 - Op1: 21: tout est ok mais c'est dur
 - Op2: 5 : tout est ok mais c'est assez simple
 - Op3: 8 : ~

Sizing: Temps idéal

5 8 13 21

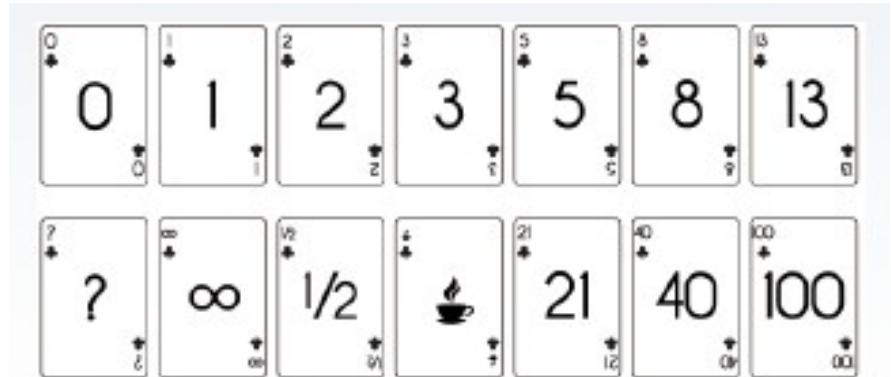
Le PO doit faire une analyse business:

- 21: 500dt
- 5: 300dt
- 8: 450dt

Le PO doit calculer le ROI en terme financier: $450/8$

Le planning poker

- Le « poker planning » est une pratique Scrum qui permet d'évaluer la complexité d'un élément du Backlog Produit.
- Il s'agit de réunir l'équipe de réalisation, le Product Owner et le Scrum Master dans une même salle et de leur donner à chacun un jeu de cartes :



l'influence des autres membres de l'équipe.

Le planning poker

Objectifs du planning Poker :

- L'objectif de la réunion est d'estimer la complexité de chaque élément du backlog produit relativement à un *élément de référence généralement simple et dont la signification est bien maîtrisée par l'équipe.*
- De façon arbitraire, l'élément de référence reçoit la valeur 3 ou 5. Ces points correspondent à un effort de réalisation pour l'équipe.

Le planning poker

Déroulement du planning Poker :

- Lecture d'un élément du backlog par le Product Owner
- Discussion rapide de cet élément
- Vote (chaque participant sélectionne une carte et tous les participants montrent la valeur en même temps)
- Discussion sur les écarts (« J'ai mis 2 et toi 13 ... pourquoi penses-tu que cet élément est si compliqué ? »)
- Nouveau vote
- En cas d'accord, le Product Backlog est mis à jour avec la valeur sur laquelle l'équipe s'est accordée
- S'il n'y a pas unanimité, une discussion est entamée. Une compréhension différente du besoin ou de la solution à adopter peut être la cause des divergences d'opinion. On répète le processus d'estimation jusqu'à l'obtention de l'unanimité.

Le planning poker

Planning Poker :

- Signification des cartes (unité = « story points »):
 - 0 → « cette story est déjà faite » or « cette story c'est pratiquement rien, juste quelques minutes de travail »
 - Tasse à café → « Je suis trop fatigué pour penser. Faisons une courte pause. »
 - 1, 2, 3, 5, 8, 13 → estimation de l'effort (suite de Fibonacci)
 - 21, 40, 100 → grande stories à décomposer
 - ? → « Je n'ai aucune idée. Vraiment aucune. »
 - ∞ → « A mon avis, la décomposition s'avère indispensable »

Processus de planification de release

Étape 3 : Définir la durée des sprints

- Au lancement, une des premières questions à laquelle il faut répondre concerne la durée des itérations.
- Il n'y a pas de réponse universelle. Toutefois, toutes les itérations doivent être de même durée et qu'une itération est un bloc de temps fixé.
- La pratique actuelle dans les développements avec Scrum recommande des sprints de deux ou trois semaines

Processus de planification de release

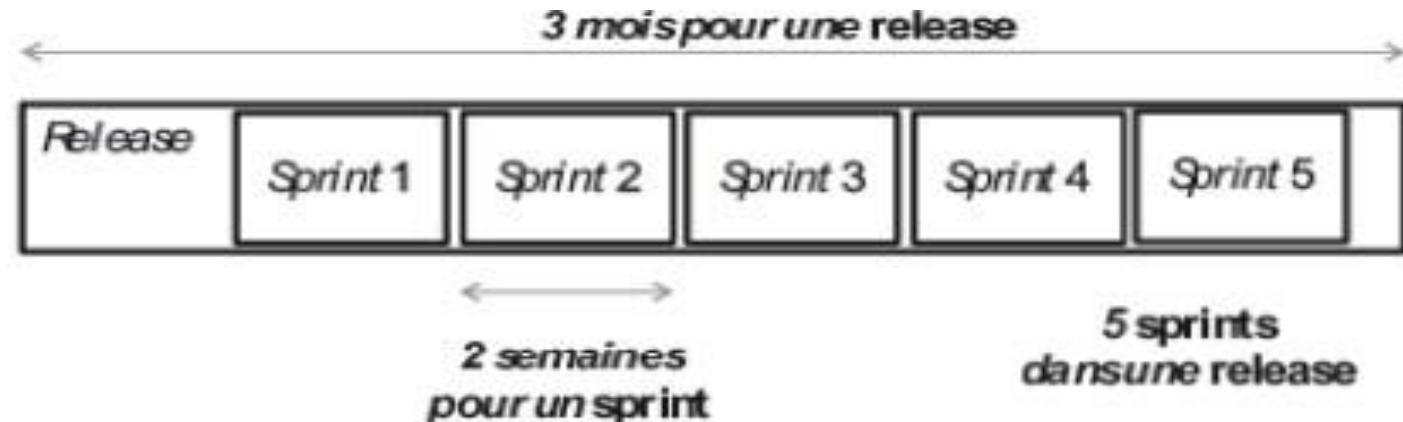
Étape 3 : Définir la durée des sprints

Les critères à retenir pour définir la bonne durée sont :

- **L'implication des clients et utilisateurs**
- **Le coût supplémentaire engendré par le *sprint***
 - préparer le produit partiel, faire les tests de non-régression, préparer la démonstration et pour les revues de fin et de début de *sprint*
- **La taille de l'équipe**
 - Plus il y a de personnes dans l'équipe, plus il faudra de temps pour se synchroniser.
- **La durée maximum pour prendre en compte un changement**
- **La date de fin de la *release***
- **Le maintien de la motivation de l'équipe**

Processus de planification de release

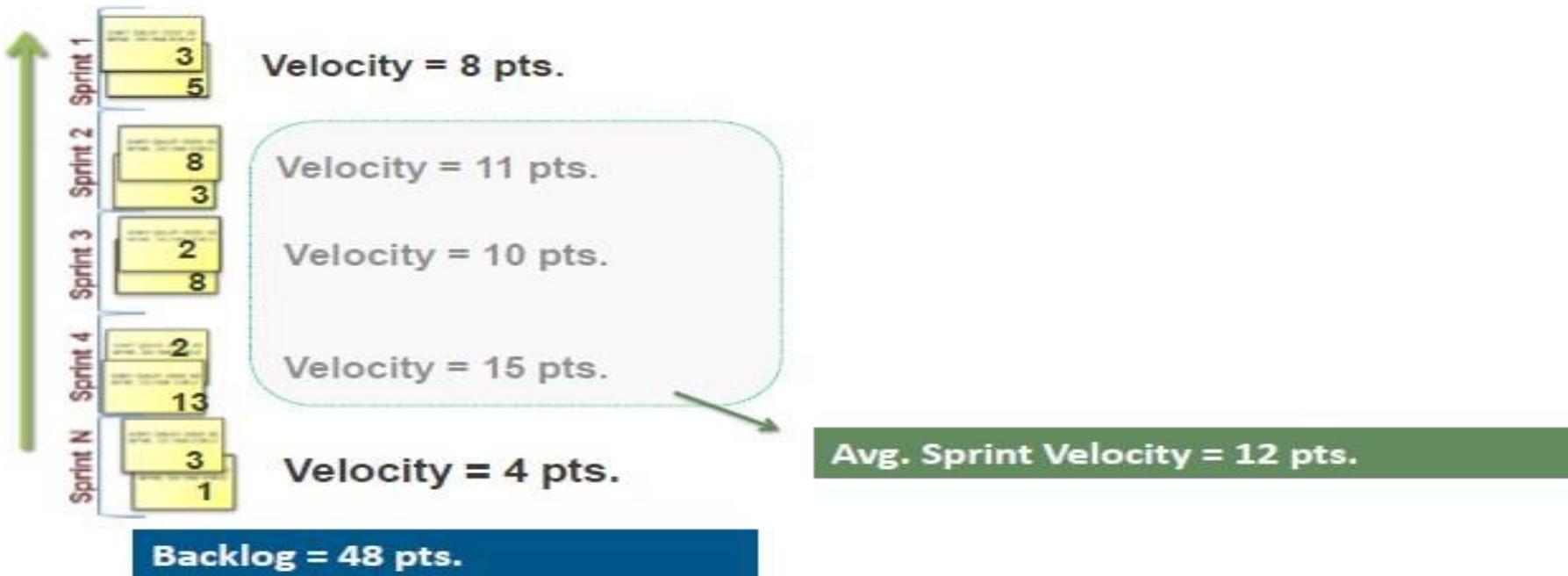
Étape 3 : Définir la durée des sprints



Durées de releases et de sprints usuelles

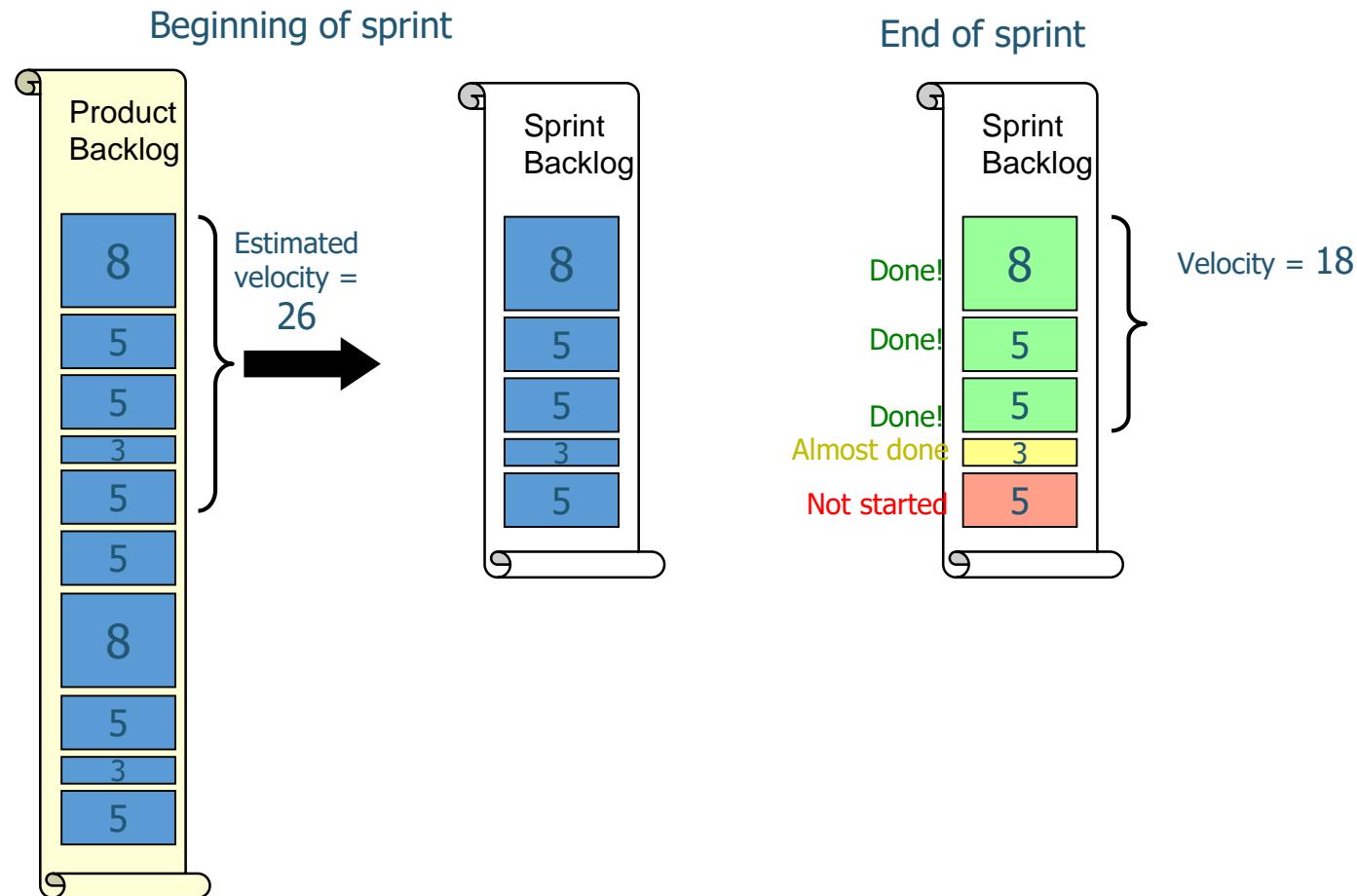
Processus de planification de release

Étape 4 : Estimer la capacité de l'équipe - Vélocité



Vélocité réelle= somme(des SP des US terminées)

Velocity



Processus de planification de release

Étape 5 : Produire le plan de release

- Considérer le backlog priorisé et estimé.
- Commencer par le premier sprint de la release.
- Y associer les stories en commençant par les plus prioritaires.
- Continuer dans ce sprint en additionnant la taille en points des stories jusqu'à arriver à la capacité de l'équipe (vitesse)
- Quand on y arrive, passer au sprint suivant.

Sprint planning

- Éviter que certains chefs de projet font la planification tout seuls, ils identifient des grandes tâches, les « chiffrent » et les affectent aux personnes de leur équipe.

Si les tâches n'avancent pas comme prévu, le chef de projet aura beau râler, les équipiers diront que les estimations n'étaient pas réalistes.

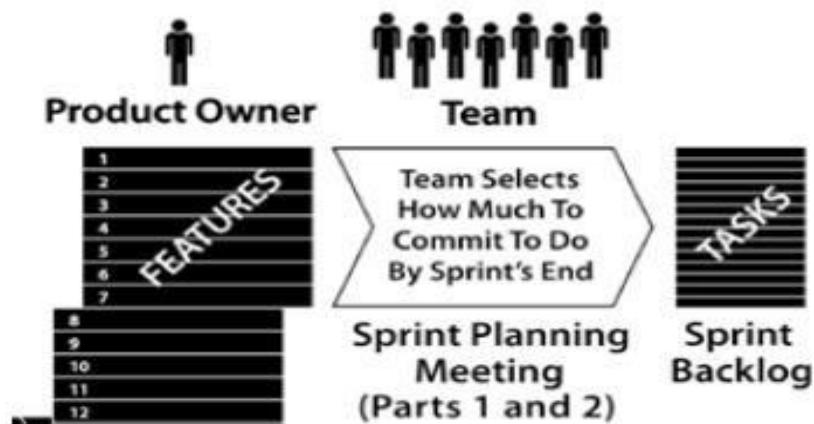
- La planification de sprint est basée sur l'idée qu'on ne peut pas prévoir de façon précise au-delà d'un certain horizon
→ L'horizon pour la planification détaillée correspond au sprint (2 - 4 semaines).

Sprint planning

- Cette réunion met en évidence, peut-être encore plus que pour la planification de release, le rôle essentiel de l'équipe dans l'élaboration des plans.
- Le travail du sprint appartient à l'équipe : ce n'est pas un chef qui définit ce qu'il y a à faire, c'est l'équipe qui s'organise elle-même.
- Au-delà de sa fonction première de planification, la réunion est un rituel qui prépare l'équipe à travailler de façon collective pendant le sprint.

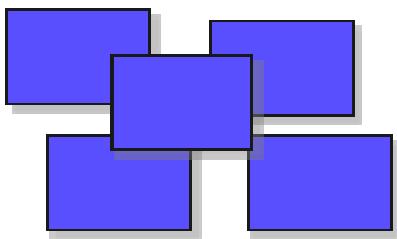
Sprint planning

- **Durée:** 1 heure – 4 heures
- **Participants:** PO, SM, Equipe
→ Le Backlog du produit au mur, définition de « Done »
- **Objectif:** extraction des items du Backlog du produit pour le sprint

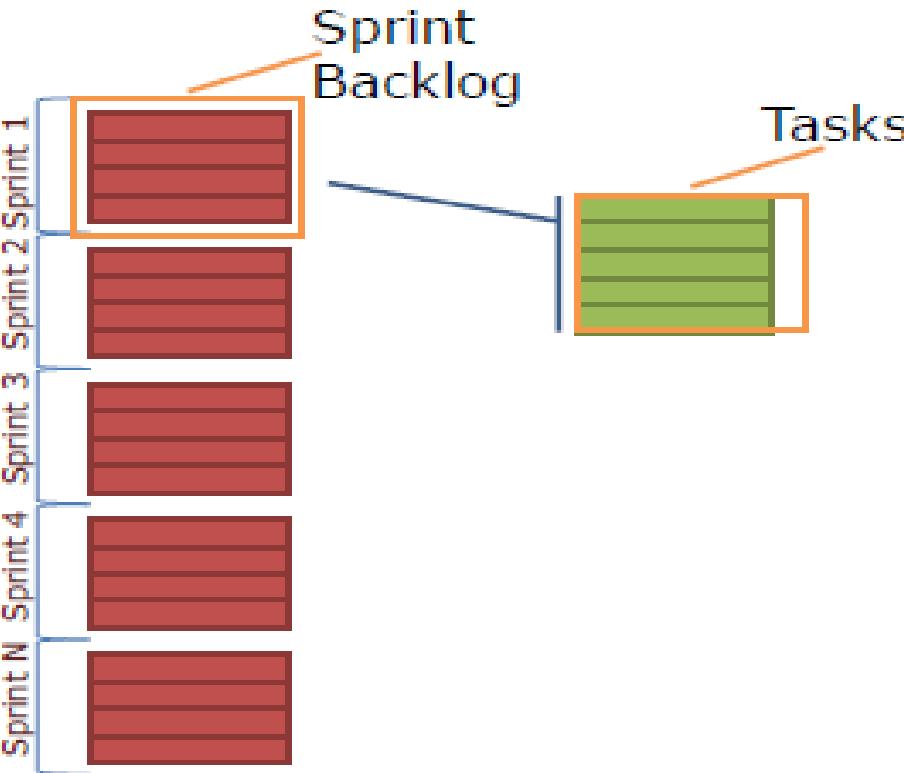
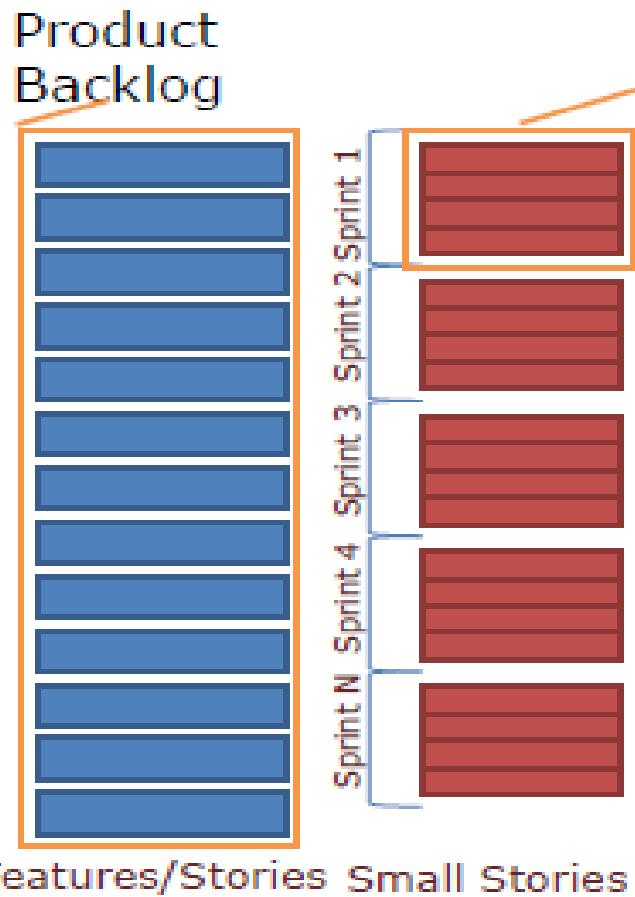


Sprint planning

Product owner identifica gli ambiti su cui lavorare



Themes/Feature Groups



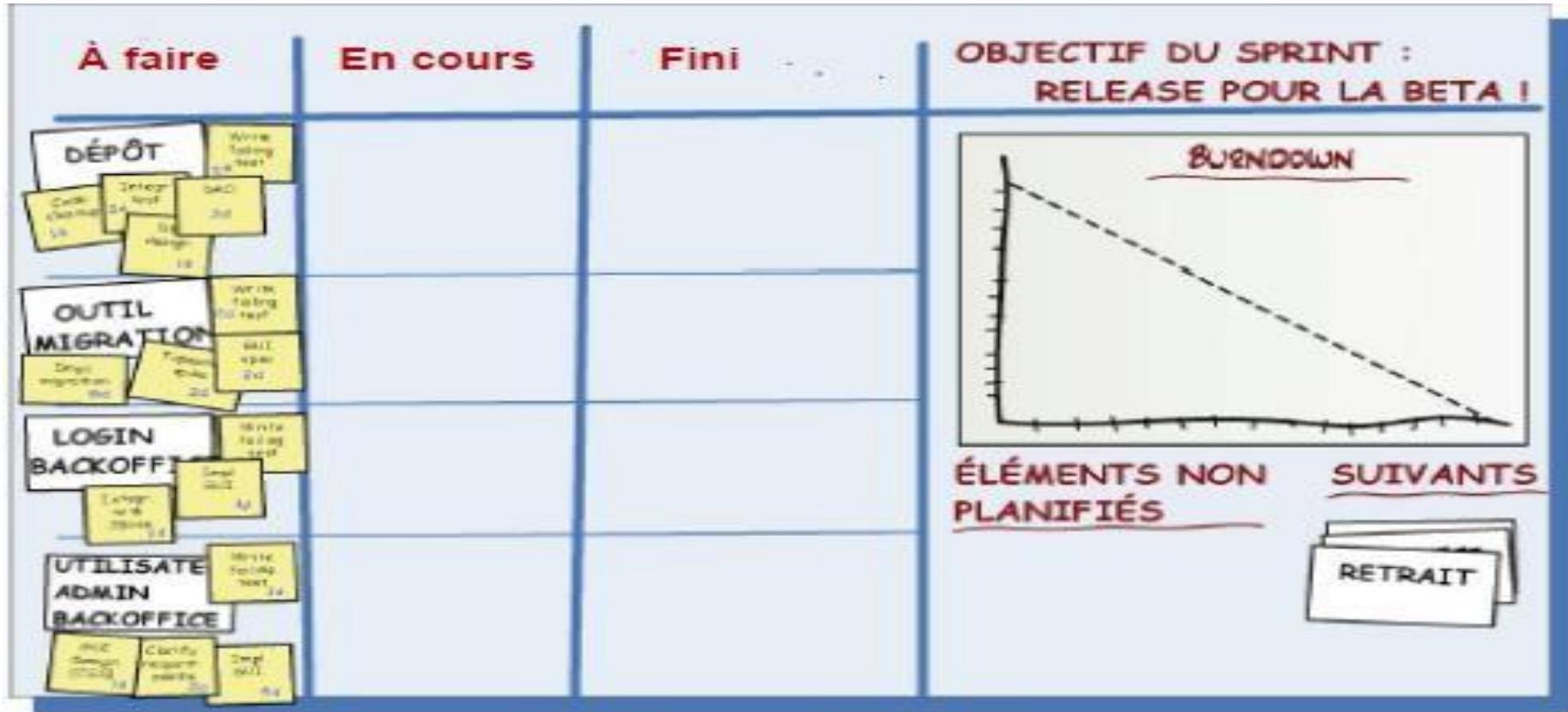
Features/Stories Small Stories

Sprint planning

- Il est élaboré lors de la réunion de planification du sprint.
- Pour chaque story sélectionnée, l'équipe identifie les tâches correspondantes.
- Sur le tableau, les stories et les tâches sont placées avec des **Post-it**.
- L'état des tâches est reconnu selon la place de la tâche dans des zones représentant chaque état : **à faire, en cours et finie**.



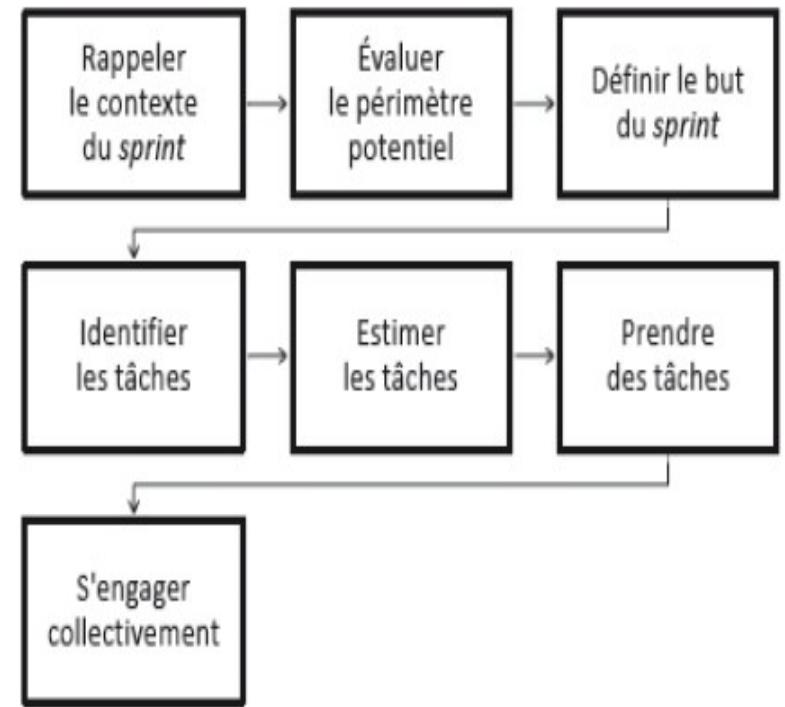
Sprint planning



Sprint planning

- La planification de sprint est une séance de travail collectif, limitée dans le temps.
- La durée de la réunion de planification de sprint est à ajuster en fonction de la durée du sprint en question:
→ limiter à **2*n heures**, où, n étant le **nombre de semaines dans le sprint**.

Par exemple : Pour un sprint de deux semaines, la réunion a une limitation à 4 heures.



Sprint planning

- **Étape 1 : Rappeler le contexte du sprint**

- Le Product Owner rappelle la place de ce sprint dans la release en cours (chaque sprint a un numéro séquentiel qui lui est affecté) ;
- il annonce la date de fin, en fonction de la durée usuelle.

- **Étape 2 : Évaluer le périmètre potentiel**

- Il s'agit de préciser le périmètre envisagé pour ce sprint, c-à-d les éléments du backlog de produit qui vont être réalisés.

Règle:

- C'est le Product Owner qui définit les priorités et donc l'ordre des stories candidates à être dans le sprint.
- C'est l'équipe qui est la seule à décider du périmètre, c-à-d à arrêter la liste des stories candidates.

Sprint planning

- forcer toute l'équipe à discuter pour éclaircir des points de solution par rapport à cette story,
- demander si nécessaire au Product Owner des précisions sur le comportement attendu.

- La liste des tâches se construit progressivement avec l'examen des stories sélectionnées → tâches déduites des stories
- puis en complétant avec des tâches indépendantes des stories
- storyless

Sprint planning

- **Étape 5 : Estimer les tâches**

- L'estimation du temps à passer sur une tâche est faite collectivement, par l'équipe.
- Les tâches sont estimées en *heures*. Il est conseillé d'avoir des tâches suffisamment petites pour qu'elles soient finies en moins de 2 jours de travail
- La liste des tâches constituée lors de la réunion de planification n'est pas figée :
 - des tâches peuvent être ajoutées,
 - d'autres supprimées
 - et d'autres décomposées pendant le *sprint*.

Sprint planning

- **Étape 6 : Prendre les tâches**

- Les membres de l'équipe qui prennent eux-mêmes les tâches.
- Il n'est pas obligatoire d'aboutir à l'attribution de toutes les tâches : il suffit que chacun ait du travail pour les premiers jours du sprint ; l'affectation des autres tâches est différée.
- Il est fréquent de revoir le périmètre après la décomposition en tâches :
 - Avec une idée plus précise du travail à faire, l'équipe peut décider d'en faire plus ou moins que le périmètre évalué en début de réunion.
 - le Product Owner doit assister à toute la réunion.

Sprint planning

- **Étape 7 : S'engager collectivement**

- Pour finir la réunion, l'équipe s'engage à réaliser les stories sélectionnées.
- L'engagement collectif est important pour motiver l'équipe.
 - Peut permettre de déceler des réticences de certains, qu'il est préférable de prendre en compte avant de finir la réunion.
- Avant de demander l'engagement, le *ScrumMaster* annonce la capacité prévue pour ce sprint en la calculant à partir des stories dans le périmètre.
- Même s'il peut y avoir de légères variations, il est important que cette capacité reste dans une fourchette raisonnable par rapport à la vitesse moyenne des derniers sprints.

Sprint planning

- Préparer le backlog de produit en anticipation
- Laisser l'équipe décider du périmètre
- Laisser l'équipe identifier les tâches
- Décomposer en tâches courtes
- Prendre un engagement raisonnable
- Garder du mou dans le plan de sprint
 - Pour les incertitudes dans les estimations sur les tâches.
 - Pour le travail en anticipation sur le *sprint suivant*.
 - Pour les impondérables susceptibles de se produire.
- prévoir entre 10 et 30% de mou (temps non affecté)
- Faire de la conception tout au long des sprints

Role du PO important

Que faire si le P.O. persiste à dire qu'il n'a pas le temps de participer aux réunions de planning de sprint ?

- Essayer d'aider le P.O. à comprendre pourquoi sa participation directe est cruciale, et espérer qu'il change d'avis.
- Essayer de convaincre quelqu'un de l'équipe de se porter volontaire pour servir de représentant du P.O. pendant la réunion.

Par exemple, dire au directeur de produit : « Puisque que vous ne pouvez pas venir, nous laisserons **xxx** ici présent vous représenter. Il aura les pleins pouvoirs pour changer la priorité et la portée des histoires pendant la réunion. Je vous suggère de vous synchroniser avec lui le mieux possible avant la réunion. Si **xxx** ne vous convient pas comme représentant, merci de suggérer quelqu'un d'autre, à condition que cette personne puisse rester avec nous pour toute la durée de la réunion. »

- Essayer de convaincre l'équipe de direction de désigner un nouveau P.O.
- Repousser le démarrage du sprint jusqu'à ce que le P.O. trouve le temps de participer à la réunion. En attendant, refuser de s'engager sur une quelconque livraison. Laisser l'équipe consacrer chaque jour à ce qui lui paraît le plus important ce jour-là.

Attention !!!

- Durant les sprints, la capacité de l'équipe **doit augmenter**.
→ bon SM, bon PO, équipe motivée et engagée
- La capacité peut diminuer à cause des **dettes techniques**:
 - **Bug** → problème au niveau d'une tâche:
 - Ne doit pas être réintégrer dans le BP
 - Ne s'estime pas
 - L'équipe assume!
 - **Dysfonctionnement** → problème au niveau d'une ou plusieurs US
 - Il faut l'intégrer dans le BP
 - Sa complexité sera plus importante par rapport à sa première estimation

Daily

- Le scrum quotidien est un point de rencontre où tous les membres de l'équipe répondent à trois questions simples et actualisent le plan de sprint :
 - **Présenter ce qui a été fait** : **Qu'as-tu fait depuis le dernier scrum?**
 - **Prévoir ce qui va être fait** : **Que prévois-tu de faire jusqu'au prochain scrum?**
 - **Identifier les obstacles** : **Quels sont les obstacles qui te freinent dans ton travail ?**
- Son but principal est d'optimiser la probabilité que l'équipe atteigne les objectifs du sprint.
- Les moyens pour atteindre ce but consistent en :
 - Éliminer les obstacles nuisant à la progression de l'équipe.
 - Garder l'équipe concentrée sur l'objectif du sprint.



Daily

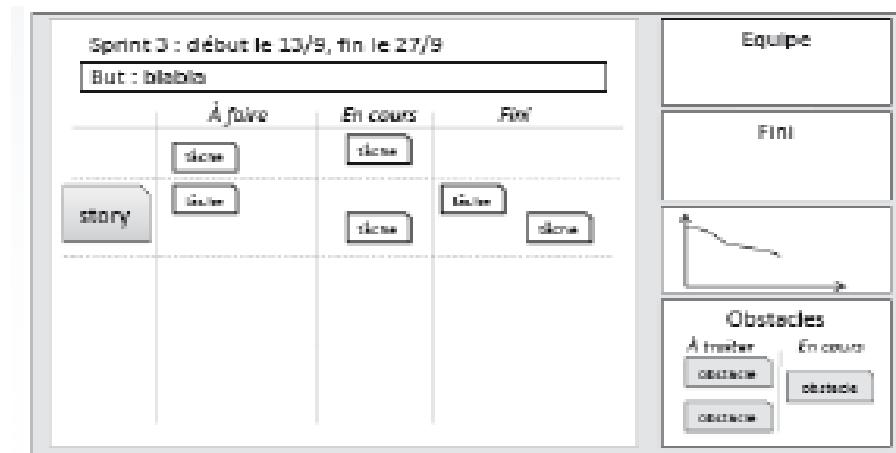
- Un obstacle empêche une tâche (ou plusieurs) de se dérouler normalement.
- La personne doit penser à ce qui l'a gêné dans les tâches qu'elle a faites.
- C'est au ScrumMaster de rechercher les vrais obstacles derrière les différentes formulations potentielles :
 - « *difficulté à communiquer avec le Product Owner !* ».
 - « *j'ai proposé deux façons de procéder pour l'arrangement des boutons sur la fenêtre de validation de la story 22 et j'attends toujours la réponse du Product Owner !* ».

Daily

- Si un obstacle est identifié, trois situations sont envisageables:
 - Si un des membres de l'équipe connaît déjà la solution pour l'éliminer, il est bien évident qu'il la donne immédiatement.
un gain de temps considérable
 - si des personnes ont seulement des pistes de solution, une discussion peut s'engager, mais ce n'est pas le lieu adéquat pour la prolonger.
Le ScrumMaster arrête la discussion et propose aux personnes intéressées de la repousser à plus tard, en dehors du scrum.
 - Si aucun membre de l'équipe n'a une idée sur la solution du problème posé, la tâche est mise en attente et c'est au Scrum Master de rechercher les solutions

Daily

- C'est la responsabilité du ScrumMaster de classer les obstacles par priorité et de faire en sorte qu'ils soient éliminés au plus vite.
- Un obstacle possède les attributs suivants :
 - son nom,
 - son état (identifié, en cours de résolution, résolu),
 - son impact (défini par les tâches qui sont bloquées ou freinées),
 - la date d'identification.



Daily

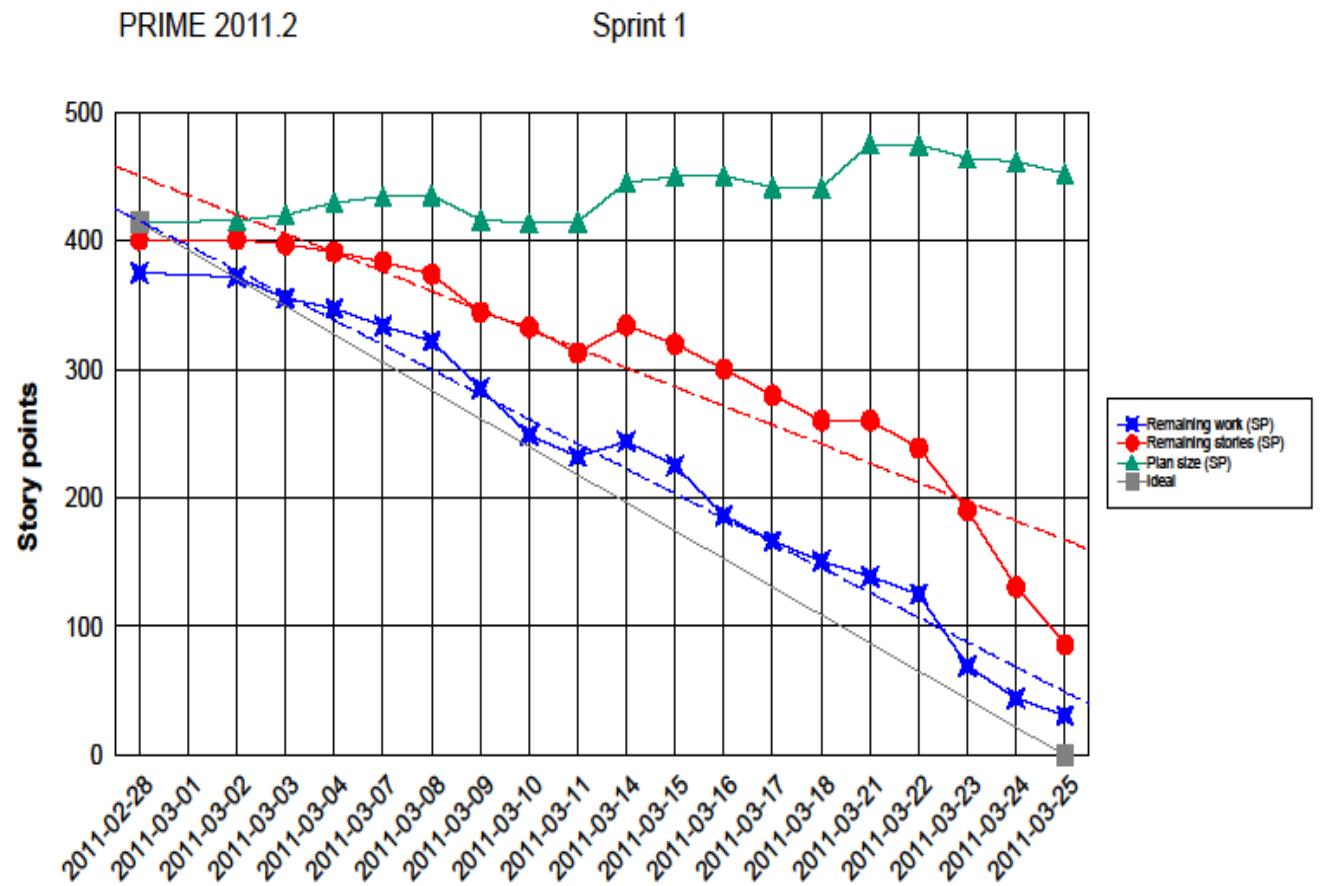
- S'en tenir à un **quart d'heure**
- Tout le monde est debout
- Ne s'intéresser qu'au reste à faire, pas au temps passé
- Faire le suivi des tâches avec les états plutôt que les heures
- Veiller à finir les stories
- Organiser des variations dans le déroulement du scrum :
 - la prise de parole se fait un coup de gauche à droite, un coup de droite à gauche ;
 - la réponse aux trois questions se fait une fois à la suite pour chaque personne, une autre fois par trois tours avec une question à chaque fois.
 - de la musique pour annoncer le scrum, qui change à chaque *sprint*
 - un ballon de rugby passé de main en main pour montrer qui a la parole.

Sprint backlog – Scrum board



Burndown

- Un **burndown** de release est un indicateur graphique basé sur la mesure de ce qui reste à faire.
- Un point dans le graphe est ajouté pour chaque sprint.
- Pour l'obtenir il faut donc une liste de ce qui reste à faire et une mesure de la taille de chaque élément, ce qu'on trouve dans le backlog de produit.
- À quoi sert le burndown chart de release ?
 - à montrer l'avancement réel, en tout cas le meilleur qu'on ait, puisqu'il est basé sur la distinction entre ce qui est complètement fini et ce qui reste.



Sprint Review Meeting

- L'un des principes des méthodes agiles :
« pour connaître l'avancement d'un développement, il vaut mieux se baser sur du logiciel fonctionnel plutôt que sur de la documentation ».

- Dans un développement de logiciel, on va montrer du code qui marche → démonstration de l'incrément de produit réalisé pendant le sprint.

→ favoriser les notions de visibilité, inspection et adaptation.



Sprint Review Meeting

Le but de la revue est de montrer ce qui a été réalisé pendant le *sprint* afin d'en tirer des enseignements pour la suite du projet.

Parmi les caractéristiques de cette réunion :

- **La revue accueille de nombreux invités**
 - Toute l'équipe Scrum participe à la réunion: l'équipe étendue, avec le ScrumMaster et le Product Owner.
 - Toutes les personnes qui sont parties prenantes (*stakeholders*) du projet y sont invitées et leur présence vivement encouragée.
 - C'est la réunion à laquelle assistent le plus grand nombre de personnes.
→ C'est l'occasion de faire collaborer tous les intervenants sur

Sprint Review Meeting

- Durée de la réunion
 - La revue de sprint a lieu le dernier jour du sprint, elle sera suivie de la **rétrospective**.
 - La durée de la réunion de planification de sprint est à ajuster en fonction de la durée du sprint en question → limiter à **1*n heures**, où, **n** étant le nombre de semaines dans le sprint.

Par exemple : Pour un sprint de deux semaines, la revue a une limitation à 2 heures.

- La revue nécessite une préparation, au moins pour accueillir le public et être en capacité de leur faire la démonstration.
- Le temps de préparation global ne devrait pas excéder une heure, car il n'y a pas de présentation formelle à faire : pas de slide préparé

Sprint Review Meeting

- **La revue montre le produit**
 - La revue de sprint porte sur le résultat du travail de l'équipe pendant le sprint : le produit partiel potentiellement livrable.
C'est cette version opérationnelle qui est présentée.
 - Dans le cas d'un développement de logiciel, il a la forme d'un build incluant les stories finies, accompagné de ce qui est nécessaire pour le faire fonctionner (tests, documentation, scripts...) et déployé sur un environnement de test.

Sprint Review Meeting

- **La revue montre le produit**
 - La revue de sprint porte sur le résultat du travail de l'équipe pendant le sprint : le produit partiel potentiellement livrable.
C'est cette version opérationnelle qui est présentée.
 - Dans le cas d'un développement de logiciel, il a la forme d'un build incluant les stories finies, accompagné de ce qui est nécessaire pour le faire fonctionner (tests, documentation, scripts...) et déployé sur un environnement de test.

Retrospective

- L'un des principes des méthodes agiles :
«À intervalles réguliers, l'équipe réfléchit à comment devenir plus efficace, puis adapte et ajuste son comportement en conséquence».
- Cette réunion permet à l'équipe d'optimiser son auto-évaluation en effectuant une rétrospective du sprint
 - faire ressortir les éléments qui ont bien fonctionné et ceux qu'il reste à améliorer.
 - des actions concrètes sont alors proposées pour le sprint suivant

Retrospective

- La rétrospective constitue un moment particulier où l'équipe s'arrête de produire, prend le temps de réfléchir et parle de ses expériences, avec l'objectif de :
 - capitaliser sur les pratiques qui ont marché,
 - éviter de refaire les mêmes erreurs,
 - partager différents points de vue,
 - permettre au processus de s'adapter aux nouvelles avancées dans la technologie utilisée pour développer.
- Elle dure en moyenne **une heure** et a lieu dans la même demi-journée que la revue de *sprint*.
- La réunion est, généralement, animée par le ScrumMaster

Retrospective

- Une rétrospective représente pour les participants une possibilité d'apprendre comment s'améliorer.
- L'idée est que ceux qui réalisent sont les mieux placés pour savoir comment progresser.
- L'objectif est l'apprentissage, et non la recherche de fautes.
- L'équipe est impliquée dans la mesure où chacun doit :
 - comprendre le besoin d'amélioration,
 - concevoir les améliorations,
 - s'approprier les améliorations,
 - être motivé pour s'améliorer.

Retrospective

- Le résultat de la rétrospective est la liste des actions qui ont été décidées. On peut classer les actions selon leur orientation :
 - celles dirigées vers les personnes et leur façon d'appliquer les pratiques,
 - celles orientées vers les outils à installer ou à adapter,
 - celles axées sur l'amélioration de la qualité.
 - Par exemple, les actions impactant les personnes sont :
 - limiter le scrum à 15 minutes,
 - faire une heure de travail en binôme tous les jours,
 - modifier la façon de prendre en compte les bugs...
- Exemple :** la limitation des scrums quotidiens à un quart d'heure est l'amélioration choisie par l'équipe.
- Les actions identifiées sont :
- apporter un minuteur,
 - Annoncer le temps écoulé,

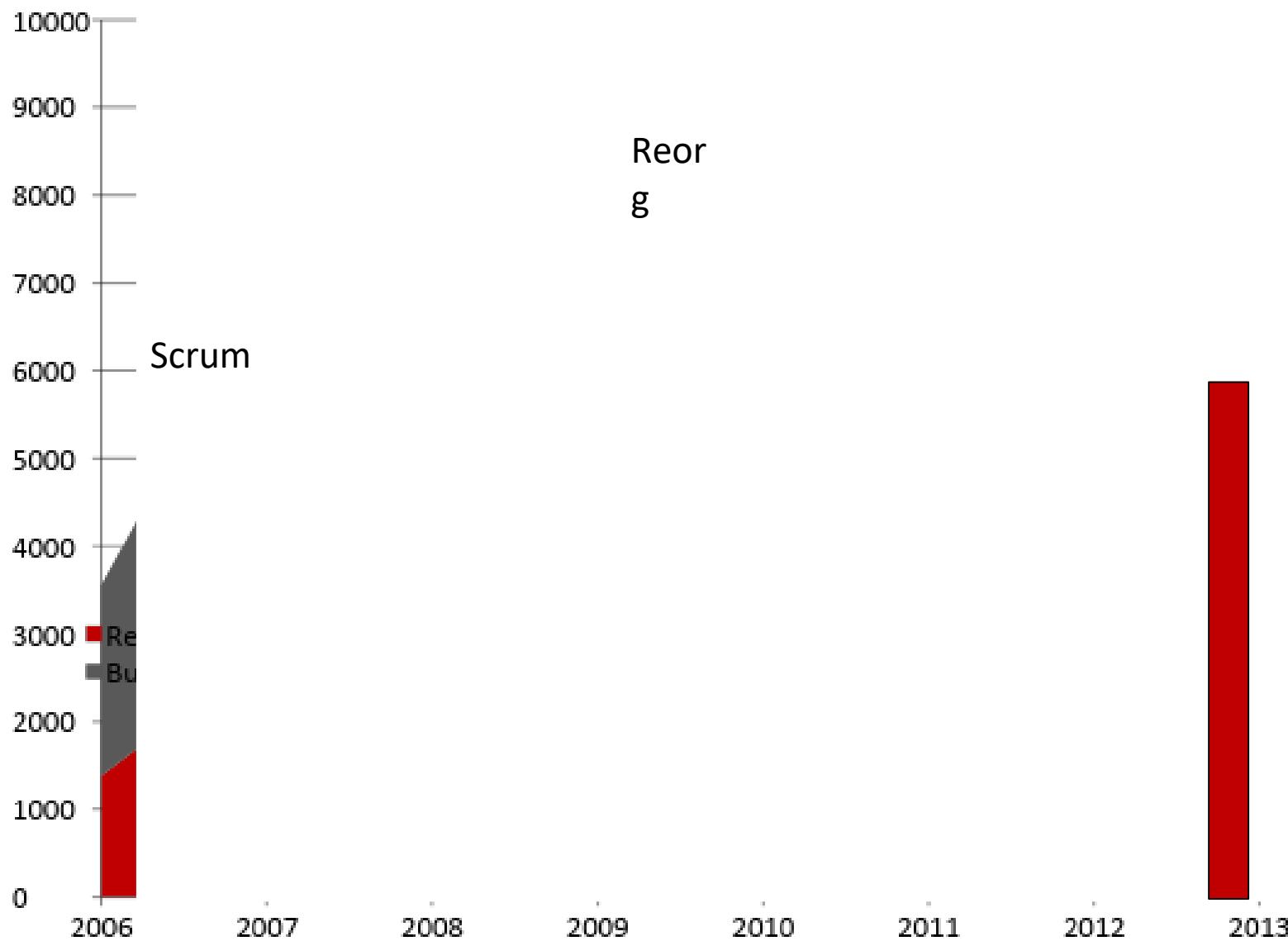
Retrospective

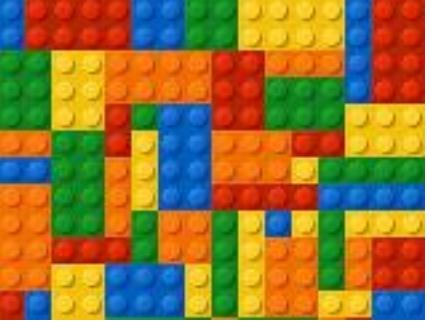
- Ne pas en faire une séance de règlement de comptes
- Parler de ce qui va bien
- Faire aboutir les actions des rétrospectives précédentes
- Se concentrer sur une amélioration
- Mener des rétrospectives plus poussées en fin de release
- Utiliser un facilitateur externe lorsque le Scrum Master n'y arrive pas

Engineering Practices

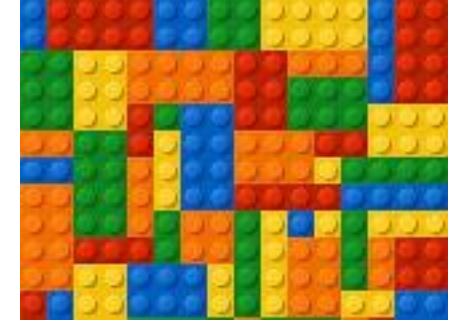
- Not prescribed, **but necessary!**
- Automatic Testing
- Pair Programming / Code Reviews
- Emergent Architecture
- Xtreme Programming is a good source

Agile at XXX— a true story





Lego for scrum



Les règles du jeu

Le matériel nécessaire



Des boîtes de Lego



Des Post-it



Des marqueurs



La disposition de la salle



1 table = 1 ville



Pour chaque table, 4 panneaux au mur

- Backlog
- Estimation
- Planning
- Rétrospective



Pour chaque table, 1 product Owner et 1 Scrum master



le déroulement du jeu : le contexte

Chaque équipe a pour mission de **construire une ville** (ville médiévale, ville écolo, ville flottante, etc.) aux caractéristiques bien précises.

Pour la construction de ces villes, les équipes vont devoir opérer en **mode Agile**, avec la méthode Scrum.

Les rôles des participants

-  **Le Product owner** : il représente le client. C'est lui qui définit les contours du projet, les priorités.
-  **Le Scrum master** : il est l'un des membres de l'équipe. Il fait le lien avec le Product owner pour s'assurer que le projet est bien défini. Il veille au respect du cadre de travail Scrum.

Les développeurs : ce sont les membres de l'équipe. Ils réalisent le projet et



Backlog / estimation



Backlog : ici, il est représenté par un grand panneau, sur lequel le Product Owner colle des post-it qui représentent les détails du projet (les livrables), classés par ordre de priorité.



Estimation : les équipes trient chaque livrable en fonction de la difficulté qu'il représente. Ils les classent alors sur un tableau entre 1 ; 2 ; 3 ; 5 ; 8 (suite de Fibonacci)

LE déroulement du jeu : les étapes



1ère étape :

Auto-organisation des équipes qui choisissent leur Scrum master



2ème étape :

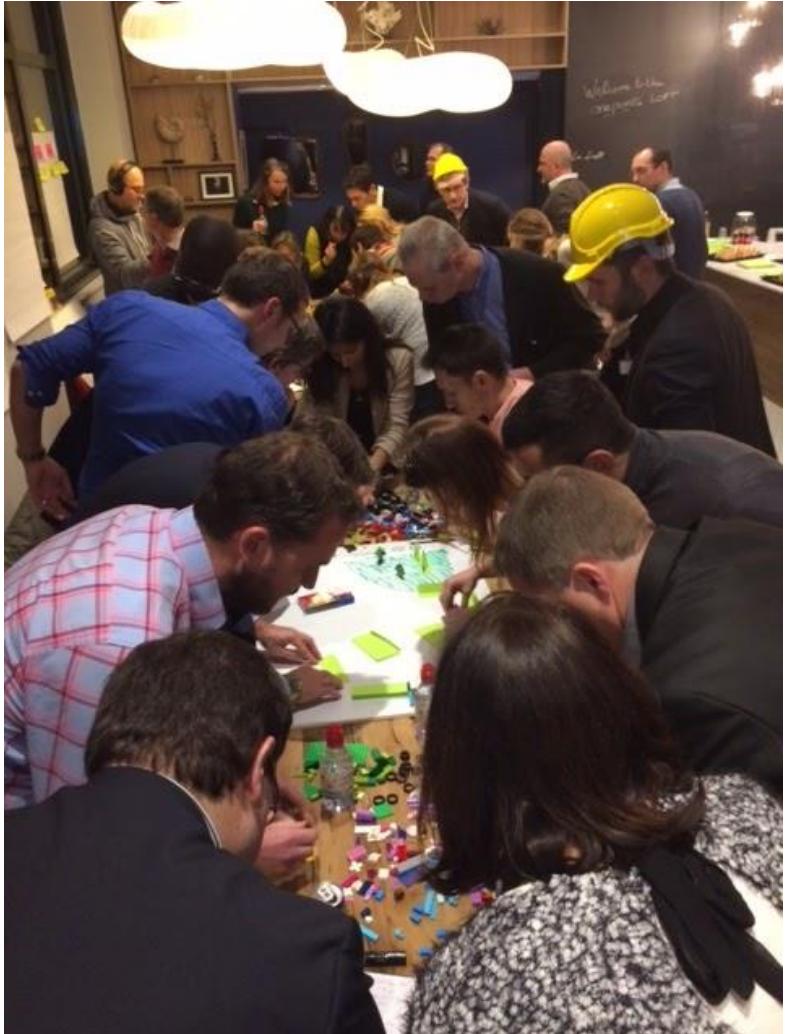
Réalisation du Backlog à l'aide du Product Owner et estimation de la difficulté des livrables.



3ème étape :

3 sprints décomposés ainsi : Planification, Production, Revue, Rétro

Revue / rétro



A la fin du sprint, le Product owner fait la **Revue** des éléments réalisés, les valide ou non.

Lors de la **Rétro**, les participants échangent sur la méthode et l'organisation pour s'améliorer lors du sprint suivant.

Le déroulement du jeu

