Agile

Méthodologies

CONSULTING BSTORM

Méthodologies

Qu'est ce qu'une méthodologie ?

Cadre établi afin de

1. **structurer**
2. **planifier**
3. **contrôler**

le processus de développement d'un système d'information.

Les disciplines du développement de software

1. **Analyse des exigences**
   - Comprendre les besoins du client.
2. **Conception** (Design)
   - Définir la solution technique.
3. **Développement**
   - Implémenter la solution.
4. **Validation** (Testing)
   - S'assurer que la solution répond adéquatement aux besoins.
5. **Déploiement**
   - Intégration globale et mise en production.
6. **Maintenance**

Différentes méthodologies

Les différences entre différentes méthodologies de développement résident essentiellement dans :

1. L'importance donnée à chaque activité.
2. La séquence permise entre chaque activité.

Large variété de méthodologie: en cascade, en spirale, incrémentale, agile...

Waterfall

Fordisme

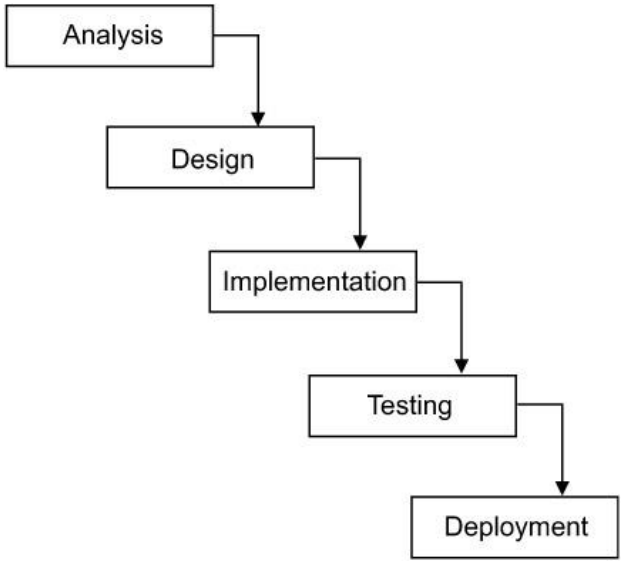
Au début des années 1900, **Henry Ford** a popularisé le fait que pour construire des produits complexes, il faut que les travailleurs se spécialisent. Ford révolutionna le secteur de l'automobile en créant le **Fordisme** où le travail est dit séquentielle.

- Augmentation de la productivité.
- Très efficace pour les produits ou services standardisés.

Méthode en cascade - Waterfall

Depuis les années 70, le secteur de l'IT utilise la méthode dite en cascade qui se caractérise par des **phases séquentielles**, qui se succèdent après la validation des livrables produits lors de la phase précédente.

- Chaque phase dépend des **livrables** de la phase précédente et correspond à une **spécialisation** des tâches.
- Le chef de projet de s'engager sur un **planning détaillé de réalisation**, prévoyant les jalons de début et fin de phases, et les activités à mener.



Réflexion

1. Quels sont les **avantages** et les **inconvénients** d'une telle méthode?
2. Quels sont les **critères favorable** / **défavorable** pour cette méthode ?

Avantages

1. **Approche simple**
   - Facile de créer planning et budget pour des tâches séquentielles.
   - Facile à suivre. Il suffit de suivre le planning et les instructions des phases précédentes.
2. **Economies d'échelle**
   - La spécialisation des équipes et le travail séquentielle permet des économies d'échelle.
3. **Pérennisation de la connaissance du projet**
   - Comme les équipes ne travaillent pas ensemble, il est nécessaire de fournir une documentation aussi importante que le code.

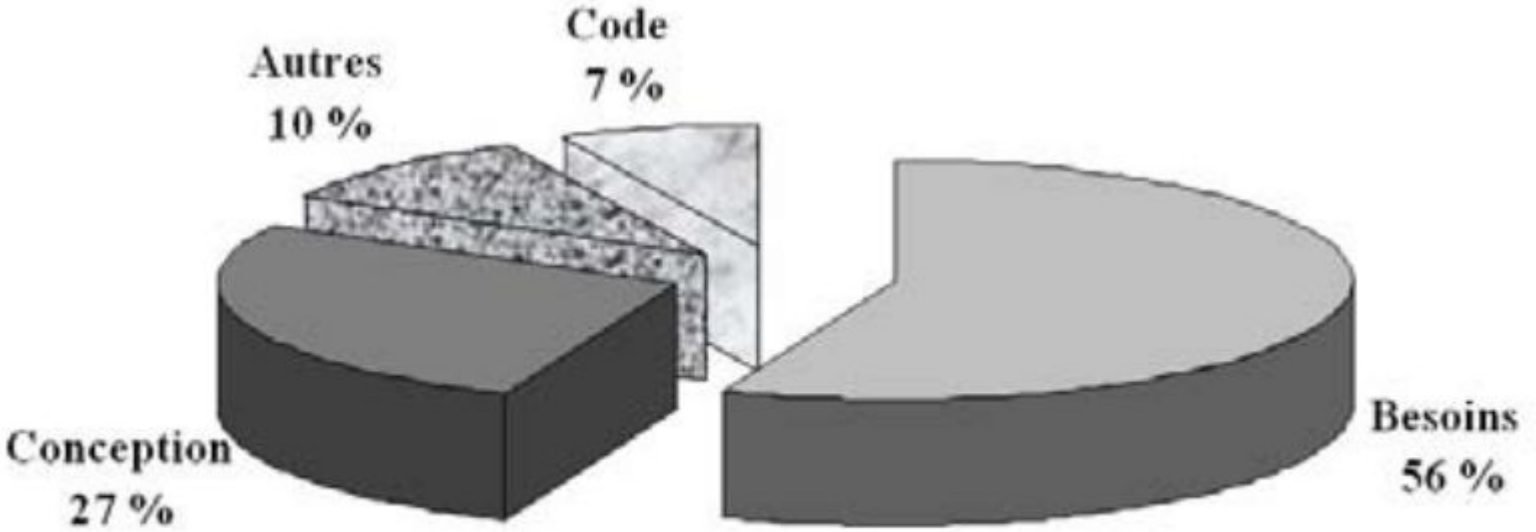
Inconvénients

1. **Rigidité de l'approche prédictive**
   - Pas toujours possible au début de projet d'anticiper tous les éléments d'un projet.
   - Difficulté à respecter le planning et budget.
   - La préoccupation majeure du chef de projet devient alors de coller au plus près au plan car tout retard ou imprévu est perçu comme échec, incompétence.

2. **L'effet tunnel**
   - Peu de possibilité pour un client de donner un feedback. Lorsqu'il peut le faire, il est souvent trop tard pour réaliser une modification. Hors il est souvent très difficile pour un client de décrire correctement son besoin du 1er coup.

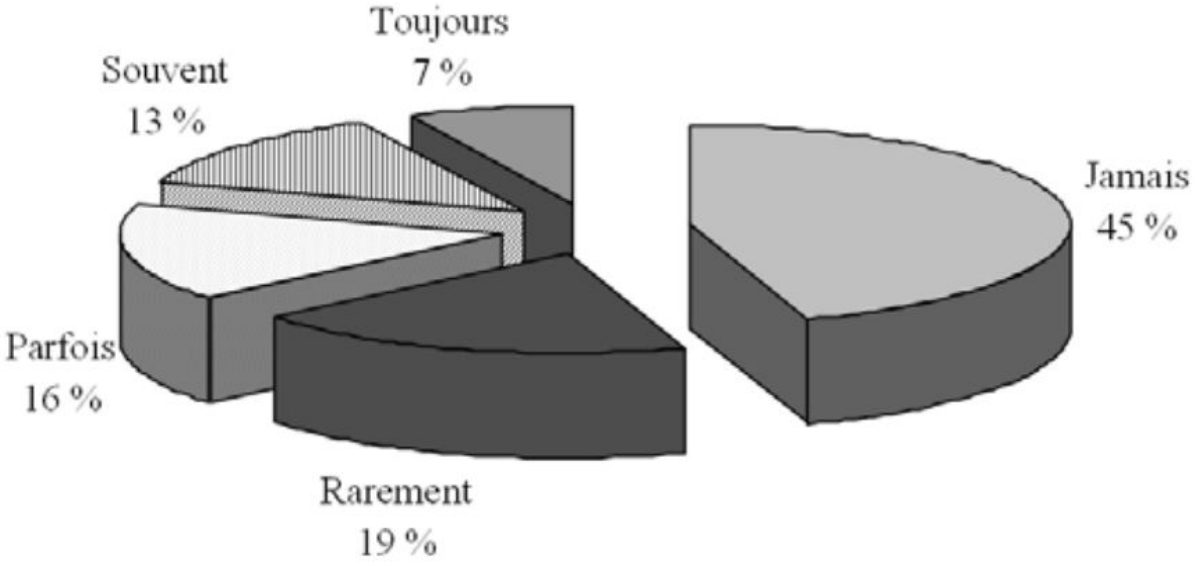
3. **Difficile d'apporter des modifications en cours de projet**
   - Il faut retourner dans les phases antérieures où les équipes sont probablement affecté à un autre projet.

Approche prédictive : origines des défauts logiciels



Source : J Johnson, Keynote speech, XP2002 (Sardaigne, Italie)

L'effet tunnel : % de fonctionnalités utilisées



Source : J Johnson, Keynote speech, XP2002 (Sardaigne, Italie)

Inconvénients

4. **Une mauvaise communication**
   - Peu importe la méthodologie, il y a toujours de problèmes de communication entre toutes les parties prenantes d'un projet. Puisque les équipes ne travaillent pas ensemble, ces difficultés sont exacerbées.
5. **L'implication tardive des développeurs**
   - Les développeurs n'interviennent que très tard dans les projets. Il est impossible pour eux d'apporter des modifications dans la conception.
6. **La levée tardive des facteurs à risques**
   - Les phases d'intégration, de tests et de présentation au client ne se font qu'à la fin du projet.
7. **Une documentation pléthorique**
   - On passe parfois plus de temps à documenter l'application qu'à la développer.

Exigences pour la méthode en cascade

1. Les exigences sont **claires** et correctement **définies**.
2. L'environnement de travail **stables**.
3. La technologie est bien **connue** et **mature**.
4. Il n'y a rien de nouveau ou d'inconnu dans le projet (**prévisibilité**).
5. De nombreux projets semblables ont déjà été exécutés avant.
6. Le projet est court (quelques mois maximum).

Qui utilise ce genre de méthode ?

- Institutionnel
- Banque / Assurance
- Aéronautique
- ….

Chaos Report

Quelles sont les chances de réussites d'un projet ?

Quelles sont les chances et les facteurs de réussites d'un projet ? Pour comprendre le taux de réussite des projets on va se baser sur le **Chaos Report de 2015**.

- Source
  - https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

Taux de succès

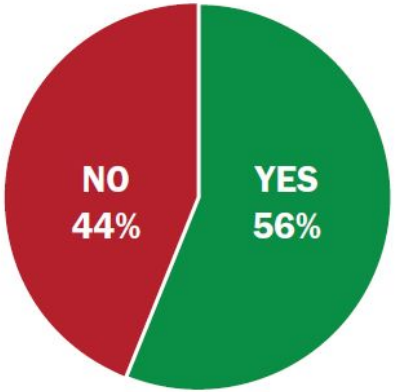
MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

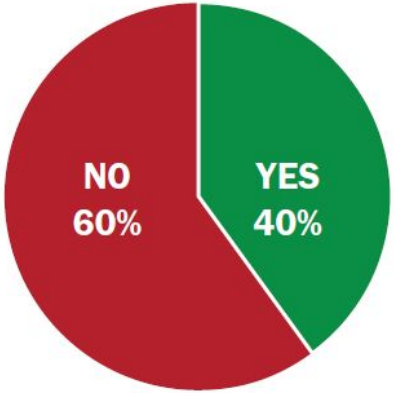
Respect des contraintes

ONBUDGET



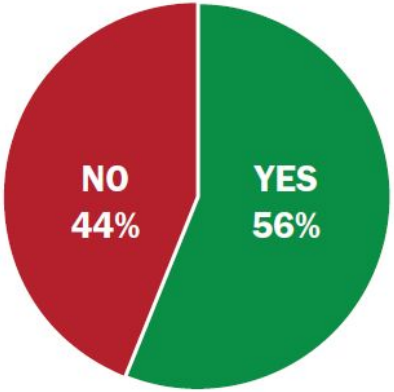
The percentage of projects that were OnBudget from FY2011–2015 within the new CHAOS database.

ONTIME



The percentage of projects that were OnTime from FY2011–2015 within the new CHAOS database.

ONTARGET



The percentage of projects that were OnTarget from FY2011–2015 within the new CHAOS database.

La taille du projet

PROJECT SIZE BY CHAOS RESOLUTION

	SUCCESSFUL	CHALLENGED	FAILED	TOTAL
Grand	6%	51%	43%	100%
Large	11%	59%	30%	100%
Medium	12%	62%	26%	100%
Moderate	24%	64%	12%	100%
Small	61%	32%	7%	100%

The size of software projects by the Modern Resolution definition from FY2011–2015 within the new CHAOS database.

CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	**100%**	**100%**	**100%**

The resolution of all software projects by size from FY2011–2015 within the new CHAOS database.

Par industrie

CHAOS RESOLUTION BY INDUSTRY

	SUCCESSFUL	CHALLENGED	FAILED
Banking	30%	55%	15%
Financial	29%	56%	15%
Government	21%	55%	24%
Healthcare	29%	53%	18%
Manufacturing	28%	53%	19%
Retail	35%	49%	16%
Services	29%	52%	19%
Telecom	24%	53%	23%
Other	29%	48%	23%

The resolution of all software projects by industry from FY2011–2015 within the new CHAOS database.

Par région du monde

CHAOS RESOLUTION BY AREA OF THE WORLD

	SUCCESSFUL	CHALLENGED	FAILED
North America	31%	51%	18%
Europe	25%	56%	19%
Asia	22%	58%	20%
Rest of World	24%	55%	21%

The resolution of all software projects from FY2011–2015 by the four major areas of the world.

Par complexité

CHAOS RESOLUTION BY COMPLEXITY

	SUCCESSFUL	CHALLENGED	FAILED
Very Complex	15%	57%	28%
Complex	18%	56%	26%
Average	28%	54%	18%
Easy	35%	49%	16%
Very Easy	38%	47%	15%

The resolution of all software projects by complexity from FY2011–2015 within the new CHAOS database.

Par méthodologie

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

Agile

Nouvelles méthodologies

Dans les années 90, plusieurs méthodologies ont été créées pour améliorer la productivité et la qualité du travail fournis par les équipes IT.

- XP programming.
- Feature-driven Development.
- Kanban.
- Lean.
- Scrum.
- Etc.

Réflexion

Comment pourrait-on modifier le processus waterfall afin de l'améliorer ?

Empirisme

Agile est fondé sur l'**Empirisme**.

- **La connaissance provient exclusivement de l'expérience.**
- On prend des décisions en fonction de ce que l'on sait pas sur ce que l'on croit savoir.
- Agile propose une approche itérative et incrémentale pour optimiser la prévisibilité et contrôler les risques.

Les 3 piliers de l'Empirisme

1. **Inspection**
    a. Remise en question constante sur le projet et le process.
2. **Adaption**
    a. Modifier les plans pour faire face à ce que l'on a appris de nouveau durant nos inspections.
3. **Transparence**
    a. Partager ouvertement les informations nécessaires.

Incrémentation et itération

Incrémentation

Le développement va être découpé en plusieurs parties. Chacun des développements vient enrichir l'existant. Un incrément est donc une avancée dans le processus de développement.

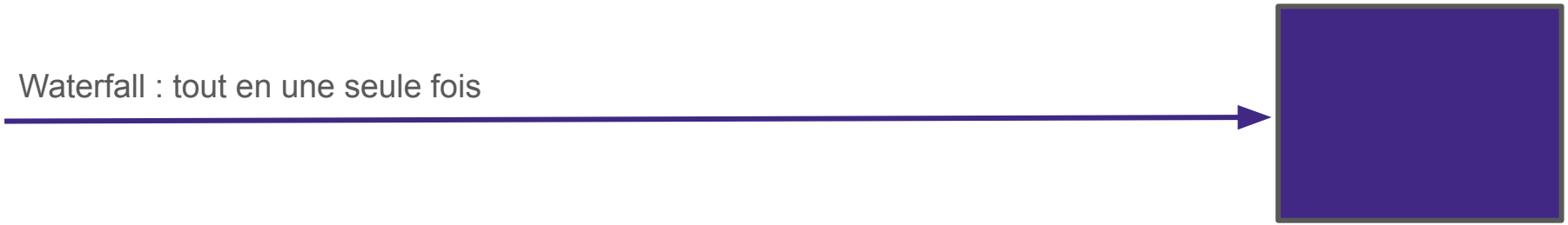
Itération

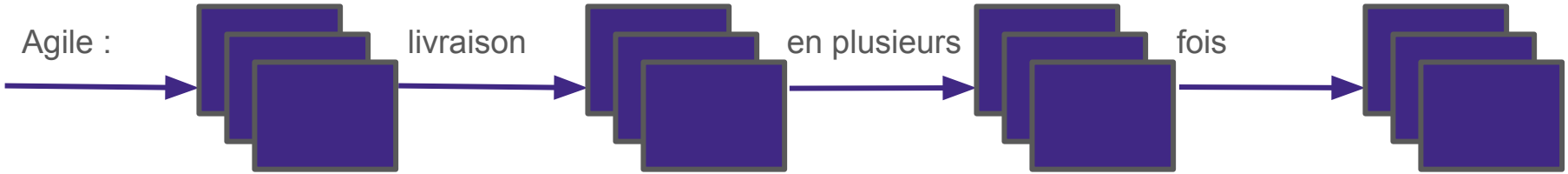
A chaque itération, les parties prenantes ont l'occasion de faire des commentaires sur l'augmentation la plus récente, ainsi que sur le produit dans son ensemble. Ce retour sera incorporé dans la version suivante.

Processus Incrémental

Processus Incrémental

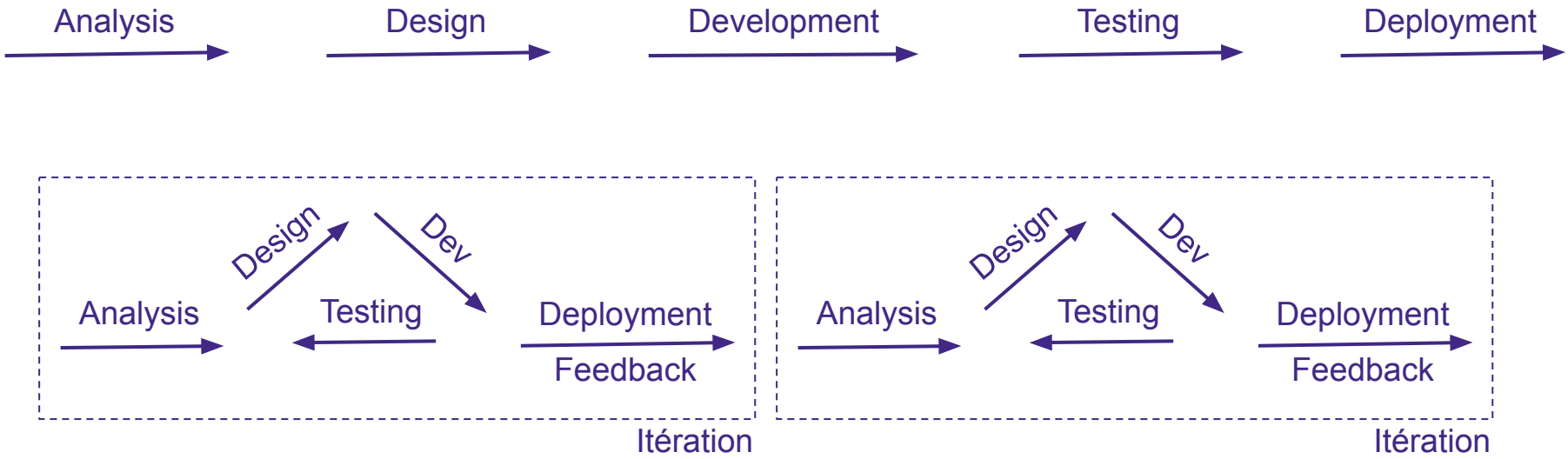
Waterfall : tout en une seule fois





Processus Itératif

Processus Itératif



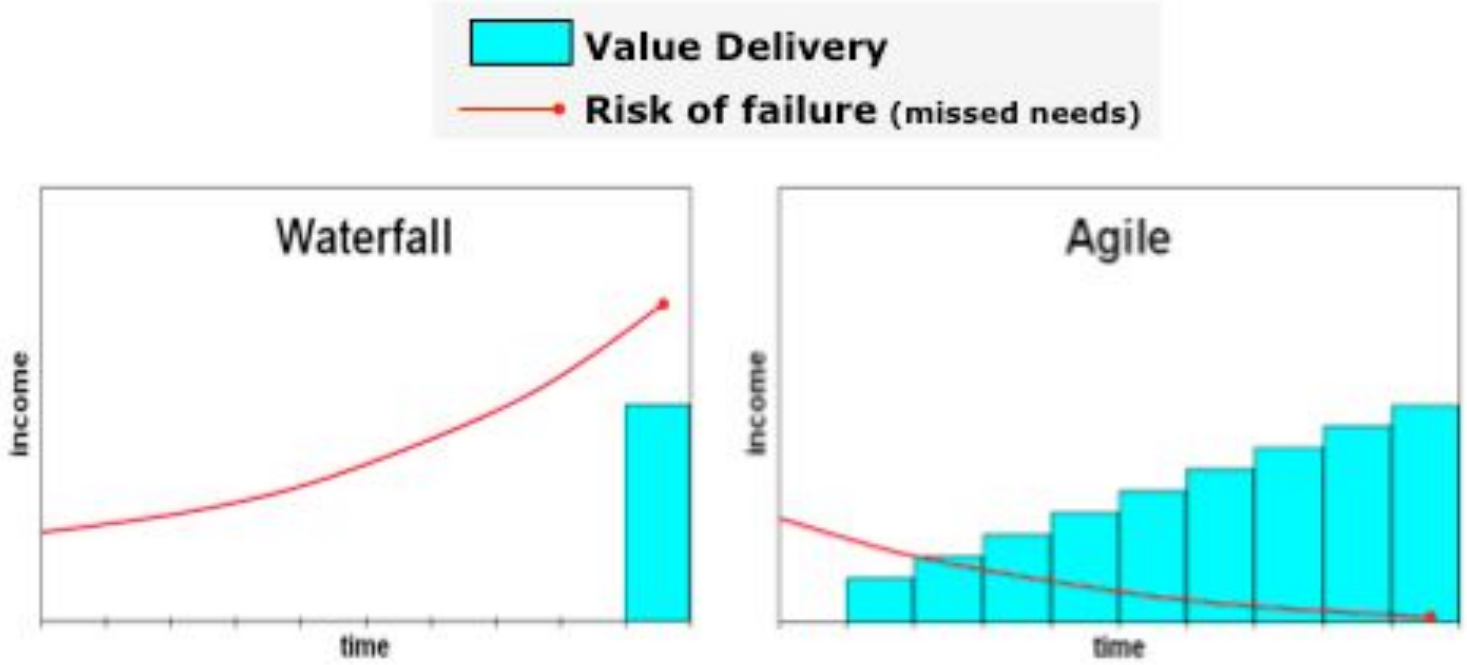
Avantages de releases fréquentes

1. **La communication est de meilleure qualité**
   - L'utilisateur a la possibilité de clarifier ses exigences au fur et à mesure.
   - Malentendus, incompréhensions, incohérences sont mis en évidence très tôt dans le projet. Il est donc encore possible de les corriger.
2. **Plus de transparence**
   - Permet de montrer l'avancement du projet au fur et mesure.
   - Le client reçoit des « preuves » tangibles de l'avancement du projet.
3. **Risque réduit**
   - Créer un produit prêt à être expédié à tout moment.
   - Les tests sont effectués à chaque itération, les anomalies détectées sont corrigées au fur et à mesure.

Avantages de releases fréquentes

4. **Meilleure valeur commerciale**
   - Produit des revenus dès le début.
   - Récupérer les coûts plus tôt.
   - Diminuer l'investissement global.
5. **Les coûts sont contrôlés**
   - Les coûts sont limités au périmètre de l'itération.
   - On ne perd que les efforts de cette itération et non la totalité du projet.
   - On peut arrêter à n'importe quelle moment, puisqu'on a une version finie à la fin de chaque itération.
6. **Possibilité d'exploiter méthodiquement les leçons tirées d'une itération**
7. **L'équipe prend confiance**

Avantages de releases fréquentes



Le Manifeste Agile

Le Manifeste Agile

En février 2001, un groupe de 17 développeurs s'est réuni à Snowbird (Utah) pour discuter de la possibilité de créer une meilleure méthode de développement.

Cette réunion a débouché sur un document intitulé **The Agile Manifesto** (Le Manifeste Agile), qui décrit 4 valeurs fondamentales et 12 principes pour améliorer la manière dont on construit des softwares.

- https://agilemanifesto.org/

Mythe sur l’origine

Une idée fausse commune est que le Manifeste Agile a été écrit, puis les méthodologies Agiles spécifiques en ont toutes été dérivées. En fait, c'était le contraire.

Au moment de la rédaction du manifeste, plusieurs méthodologies Agiles différentes avaient déjà commencé à prendre forme et à gagner du terrain auprès des équipes de développement de logiciels.

Le Manifeste était une tentative pour unir les partisans de ces méthodologies afin que les idées puissent être partagées et qu’ils puissent avancer ensemble.

4 Valeurs Fondamentales

1. Priorité aux personnes et aux interactions par rapport aux procédures et aux outils
    - Travail en groupe, communication avec les clients plutôt que de travailler uniquement à partir d'un outil de gestion de projet.
2. Priorité aux applications opérationnelles par rapport à une documentation pléthorique
    - Qui lit des centaines de pages de documentation ?
    - Souvent obsolète avant la fin du projet.
    - L'objectif est d'avoir un software qui fonctionne, pas une documentation.
    - Se limiter à la documentations succinctes à jour, documentation permanente du code.

4 Valeurs Fondamentales

3. Priorité à la collaboration avec le client par rapport à la négociation de contrat
   - Etablir une relation de confiance avec le client.
   - Feedback régulier du client, solution répondant réellement aux attentes.
4. Priorité de l'acceptation du changement par rapport à la planification
   - S'adapter au fur et à mesure de l'avancement du développement.
   - Planning flexible, modifications possibles après 1ère version du système.

12 Principes

1. Prioriser la satisfaction du client
   - La plus grande priorité est de satisfaire le client en lui livrant très tôt et régulièrement des versions fonctionnelles de l'application source de valeur.
   - Le client peut décider de la mise en production de l'application.
2. Accepter les changements
   - Accueillir les demandes de changement à bras ouverts, même tard dans le processus de développement. Les méthodologies agiles exploitent les changements pour apporter au client un avantage concurrentiel.
   - Produire des systèmes flexibles.

12 Principes

3. Livrer en permanence des versions opérationnelles de l'application
   - Livrer le plus souvent possible des versions opérationnelles de l'application, avec une fréquence comprise entre deux semaines et deux mois, avec une préférence pour l'échelle de temps la plus courte.
   - Objectif : livrer une application qui satisfasse aux besoins du client.
4. Coopérer quotidiennement
   - Clients et développeurs doivent coopérer quotidiennement tout au long du projet.
5. Construire des projets autour d'individus motivés
   - Leur donner l'environnement et le support dont ils ont besoin et leur faire confiance pour remplir leur mission.

12 Principes

6. Favoriser le dialogue direct
   - La méthode la plus efficace pour communiquer des informations à une équipe et à l'intérieur de celle-ci reste la conversation en face à face.
7. Mesurer l'avancement du projet en fonction de l'opérationnalité du produit
   - Le fonctionnement de l'application est le premier indicateur d'avancement du projet.
8. Adopter un rythme constant et soutenable
   - Adapter le rythme pour préserver la qualité du travail sur la durée du projet.
   - Développeurs et utilisateurs devraient pouvoir maintenir un rythme constant indéfiniment.

12 Principes

9. Contrôler continuellement l'excellence technique et à la conception
    - Maintenir le code source propre, clair et robuste.
10. Privilégier la simplicité en évitant le travail inutile
    - La simplicité, art de maximiser la quantité de travail à ne pas faire, est essentielle.
    - Répondre le + simplement aux besoins actuels pour que celui ci soit adaptable.
11. Auto-organiser et responsabiliser les équipes
    - Partage des responsabilités par volontariat.

12 Principes

12. Améliorer régulièrement l'efficacité de l'équipe en ajustant son comportement
    - A intervalles de temps réguliers, l'ensemble de l'équipe s'interroge sur la manière de devenir encore plus efficace, puis ajuste son comportement en conséquence.
    - Environnement en perpétuelle évolution.

Une philosophie

Observations

Agile est une philosophie de travail.

1. Agile n'est pas normatif
   - Décrit un résultat spécifique, mais pas comment l'équipe doit parvenir à ce résultat.
   - Pourquoi ? Chaque situation est différente.
2. Agile ne prétend pas être la seule solution
   - Priorité à… par rapport à…
   - Agile n’est pas la seulement manière de faire, mais souvent la meilleur.

Quelle méthodologie choisir ?

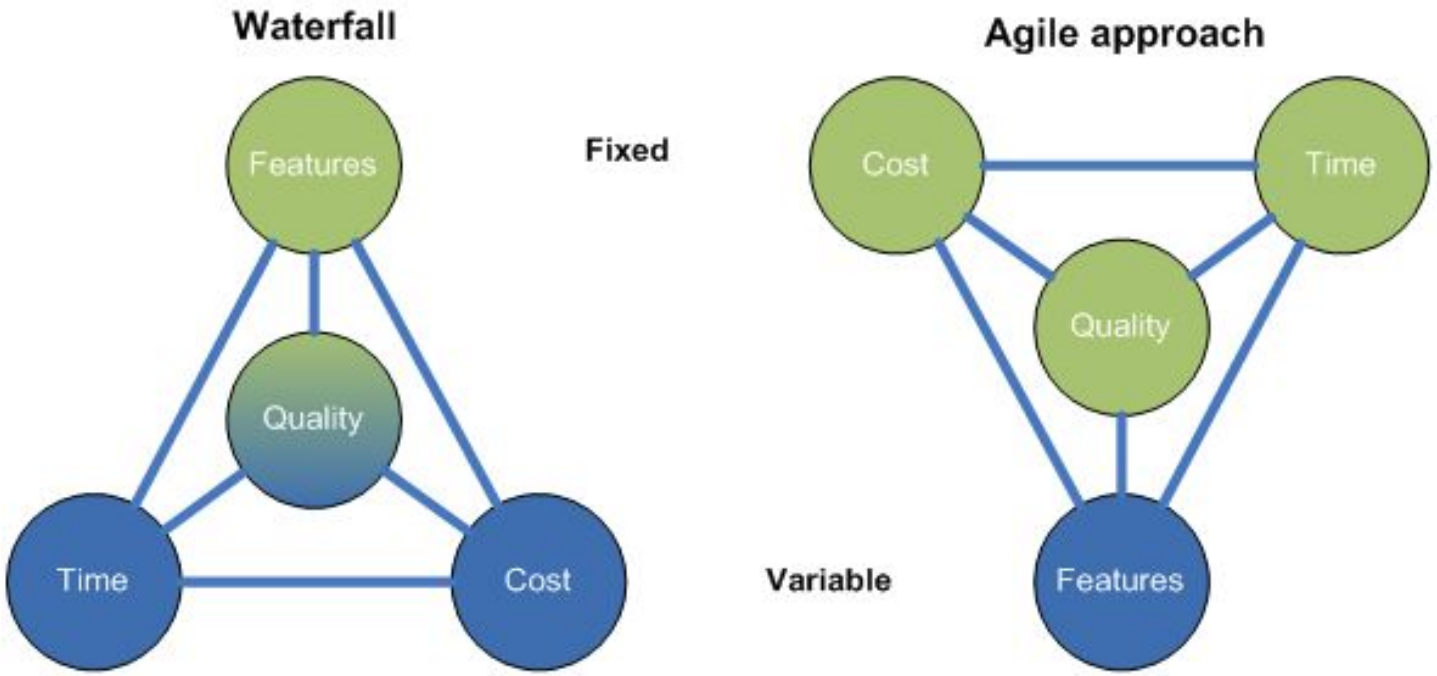
Comparaison

Cycle de vie	Waterfall	Agile
Planification	Prédictive	Adaptative
Documentation	Produite en quantité	Réduite au strict nécessaire
Équipe	Ressources spécialisées	Responsabilisation, initiative et communication
Qualité	Contrôle à la fin du cycle	Contrôle qualité précoce et permanent
Changement	Opposition au changement	Intégré dans le processus

Comparaison

Cycle de vie	Waterfall	Agile
Suivi de l'avancement	Mesure de la conformité aux plans initiaux	Travail restant à faire
Gestion des risques	Processus distinct	Intégré dans le processus
Mesure du succès	Respect des engagements initiaux	Satisfaction client

Comparaison



Quelle méthodologie choisir ?

Agile n'est pas la solution à tous les problèmes, ni la méthode la plus adaptés pour toutes les équipes. Convient pour les projets :

1. Complexes.
2. Avec des deadlines courtes.
3. Avec un risque élevé (inconnue / innovation).

Exercice

Exercice

Déterminez quelle serait la meilleure méthode entre waterfall et agile.

Justifiez votre réponse.

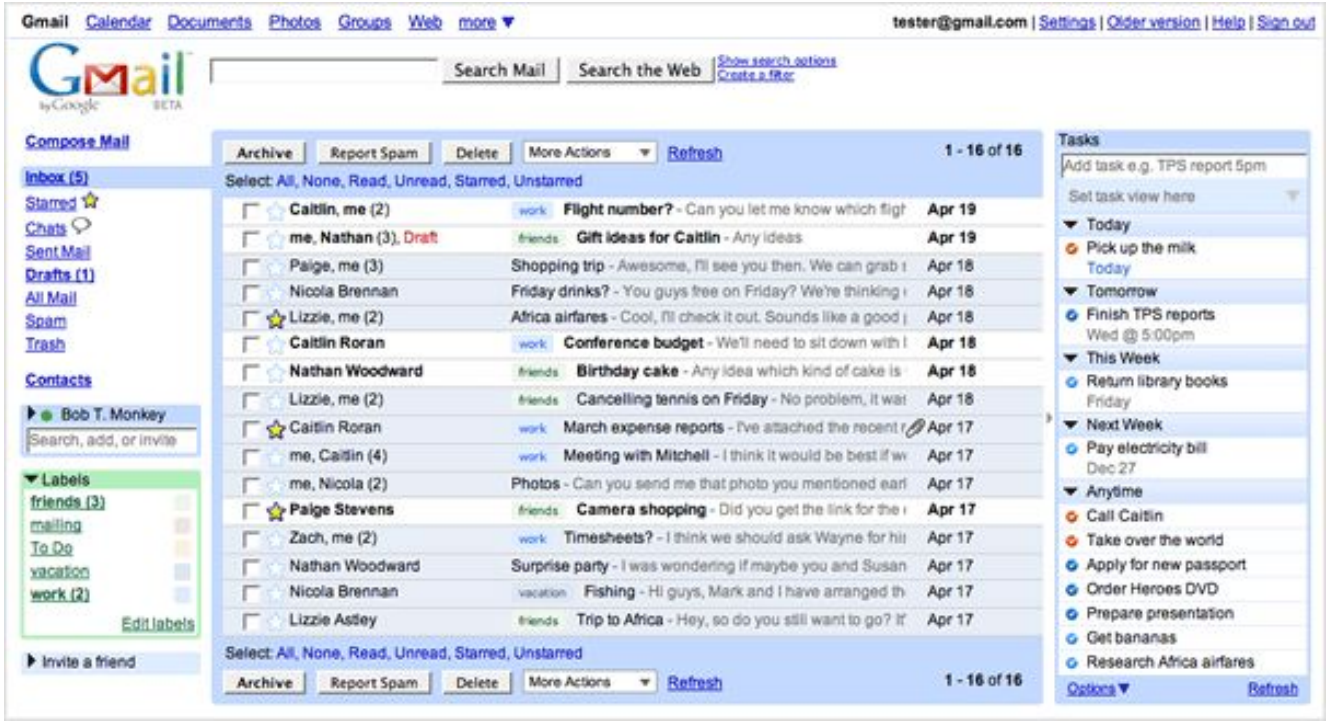
Système de navigation A380

Système de navigation A380

Waterfall

- Airbus dispose de la conception complète.
  - Ils savent ce qu'il veulent.
- Taille équipe.
- Difficulté de mettre en place un système itératif.

Gmail


Gmail Calendar Documents Photos Groups Web more ▼
tester@gmail.com | Settings | Older version | Help | Sign out
Gmail by Google BETA
Search Mail
Search the Web
Show search options
Create a filter
Compose Mail
Inbox (5)
Starred
Chats
Sent Mail
Drafts (1)
All Mail
Spam
Trash
Contacts
Bob T. Monkey
Search, add, or invite
Labels
friends (3)
mailing
To Do
vacation
work (2)
Edit labels
Invite a friend
Archive
Report Spam
Delete
More Actions
Refresh
1 - 16 of 16
Select: All, None, Read, Unread, Starred, Unstarred
Caitlin, me (2) work Flight number? - Can you let me know which flig Apr 19
me, Nathan (3), Draft friends Gift ideas for Caitlin - Any ideas Apr 19
Paige, me (3) Shopping trip - Awesome, I'll see you then. We can grab Apr 18
Nicola Brennan Friday drinks? - You guys free on Friday? We're thinking Apr 18
Lizzie, me (2) Africa airfares - Cool, I'll check it out. Sounds like a good Apr 18
Caitlin Roran work Conference budget - We'll need to sit down with Apr 18
Nathan Woodward friends Birthday cake - Any idea which kind of cake is Apr 18
Lizzie, me (2) friends Cancelling tennis on Friday - No problem, it wa Apr 18
Caitlin Roran work March expense reports - I've attached the recent Apr 17
me, Caitlin (4) work Meeting with Mitchell - I think it would be best if w Apr 17
me, Nicola (2) Photos - Can you send me that photo you mentioned earl Apr 17
Paige Stevens friends Camera shopping - Did you get the link for the Apr 17
Zach, me (2) work Timesheets? - I think we should ask Wayne for hi Apr 17
Nathan Woodward Surprise party - I was wondering if maybe you and Susan Apr 17
Nicola Brennan vacation Fishing - Hi guys, Mark and I have arranged th Apr 17
Lizzie Astley friends Trip to Africa - Hey, so do you still want to go? It Apr 17
Tasks
Add task e.g. TPS report 5pm
Set task view here
Today
Pick up the milk
Today
Tomorrow
Finish TPS reports
Wed @ 5:00pm
This Week
Return library books
Friday
Next Week
Pay electricity bill
Dec 27
Anytime
Call Caitlin
Take over the world
Apply for new passport
Order Heroes DVD
Prepare presentation
Get bananas
Research Africa airfares
Options
Refresh

Gmail

Agile - Google adopte les valeurs Agile

- Priorité au personnes
  - Googleplex, Free friday, …
- Priorité application fonctionnelle
  - Difficile à juger sans information supp.
- Priorité à la satisfaction du client / satisfaction cahier des charges
  - Récolte feedback utilisateurs, abandonne produit/fonctionnalité pas utilisé.
- Priorité changement
  - Fréquente release et roadmap de développement qui s'adapte par rapport à l'évolution du web.

Système de gestion intégrée du stock

Système de gestion intégrée du stock

Pas de réponse tranchée !

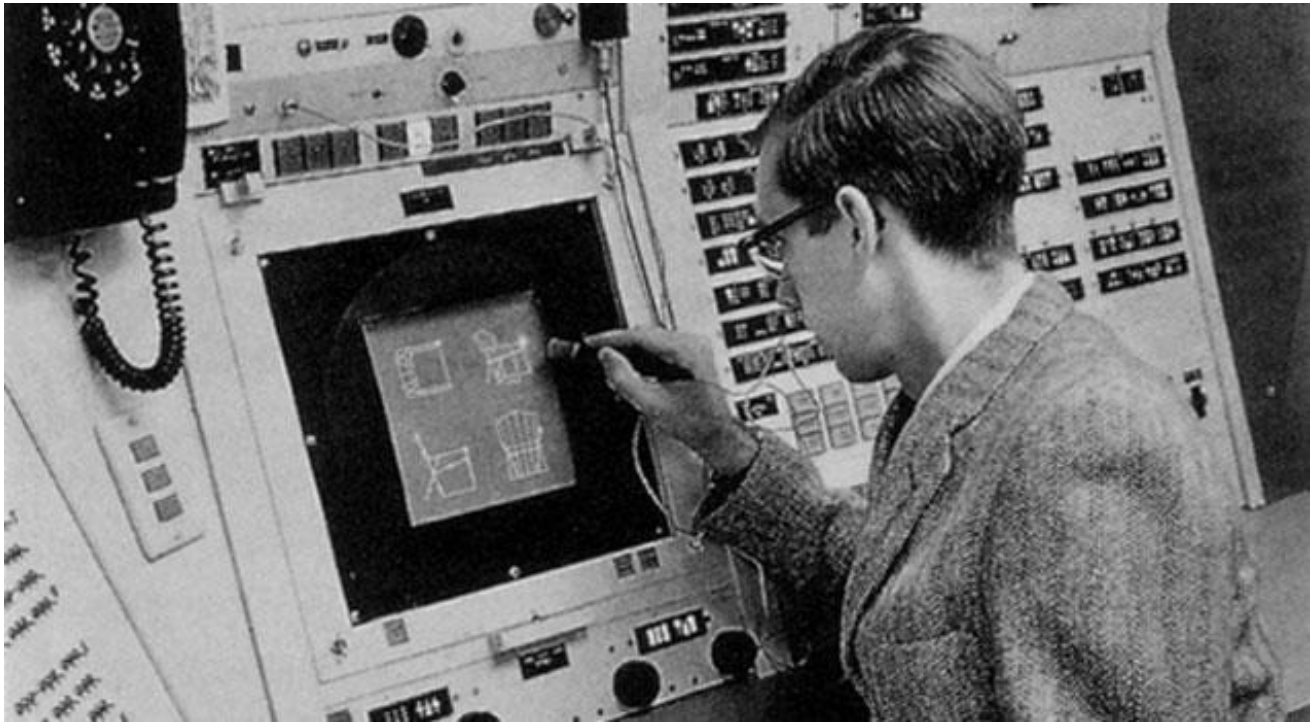
PME - Agile

- Doivent s'adapter constamment à leurs clients et fournisseurs.
- Petite équipe (agile max 10 personnes).

Carrefour - Waterfall

- Ils savent ce qu'ils veulent.
- Stable : opérationnelle depuis des dizaines d'années.
- Grande équipe.

ARPANET (ancêtre d'Internet)



ARPANET (ancêtre d'Internet)

Historiquement - Waterfall

A refaire - Agile

- Car c'est de la recherche, on ne sait pas où l'on va...

Facebook


www.facebook.com/zuck?sk=wall
facebook
Search
Mark Zuckerberg
Works at Facebook Studied Computer Science at Harvard University Lives in Palo Alto, California Knows English, Mandarin Chinese From Dobbs Ferry, New York Born on May 14, 1984
Wall
Info
Share Profile
Report/Block This Person
Wall
RECENT ACTIVITY
"I like dangerous thoughts." on Samuel W. Lessin's status.
Mark Zuckerberg
Steve, you've done so much good for the world already. I hope you get better soon.
January 17 at 11:43am via iPhone
150 people like this.

Facebook

Agile

- Constante évolution.

Scanner médicale

Scanner médicale

- **Waterfall**
- **Agile** (si expérimental)

Agile

Agile Frameworks

CONSULTING BSTORM

Agile Frameworks

Agile vs Scrum

Agile	Scrum
philosophie	**framework**

Agile vs Scrum

Les termes Agile et Scrum sont souvent utilisés de manière interchangeable, mais ils ne sont en réalité pas la même chose.

- Agile
  - “Philosophie” d’organisation de travail
  - Énoncer des principes
  - Ne donne aucune indication sur le “comment” faire
- Scrum
  - Framework d’Agile
  - Oriente vers de bonnes pratiques

Scrum is A Framework, Not A Process

Process

- Normatif : dicter étape à étape.
- Répétable de projet en projet.
- Idéal pour une utilisation dans un contexte bien défini.

Framework (“*cadre*”)

- Non normatif.
- Définition des tâches clés et d’une routine.
- Plus adaptable aux environnements volatiles.

SCRUM framework + notre environnement = notre process

CONSULTING BSTORM

Scrum is A Framework, Not A Process

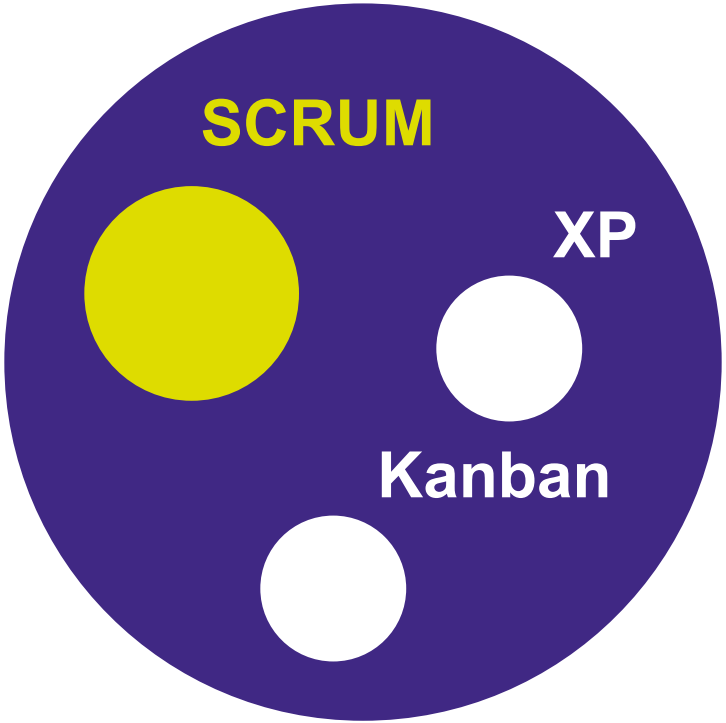
mon
SCRUM

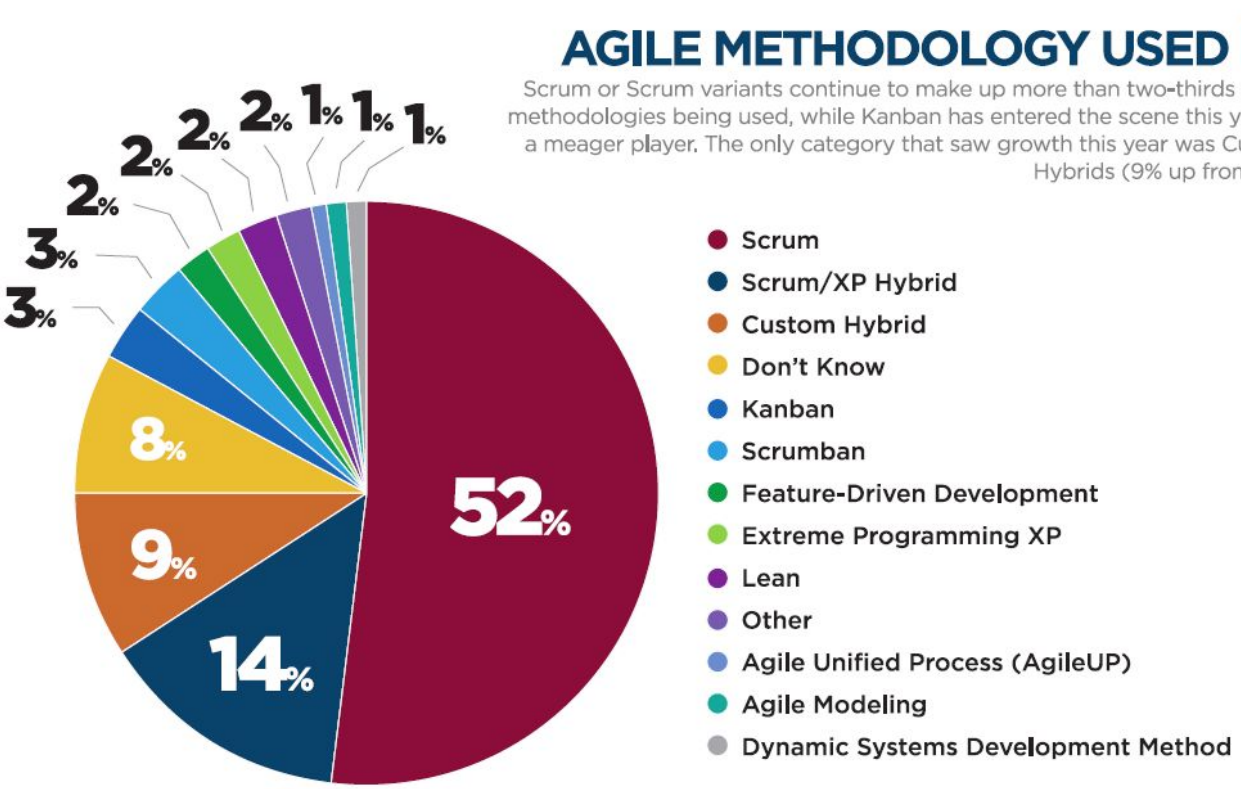
votre
SCRUM

BSTORM CONSULTING

Scrum is A Framework, Not A Process

Agile vs Scrum





BSTORM CONSULTING

75% des équipes Agiles ont adopté l'approche Scrum ou un hybride basé sur Scrum

Kanban

Kanban, c'est quoi ?

Kanban est une **méthode de gestion de flux de travail**. Il a été conçu pour aider à visualiser le travail qu'il reste à faire, limiter le travail en cours, maximiser l'efficacité et aide à devenir plus agile.

La mot **Kanban** vient du mot japonais (**看板**) qui veut dire **"tableau d'affichage"** ou "carte visuelle".

Le terme désigne aussi la méthode de gestion de **production en flux tendu** employée à la fin des années 1950 dans les usines Toyota et consistant à limiter la production d'un poste en amont d'une chaîne de travail aux besoins exacts du poste en aval.

Production "Just-In-Time"

Making only "what is needed, when it is needed, and in the amount needed"

source: https://global.toyota/en/company/vision-and-philosophy/production-system/

Taiichi Ohno et les supermarchés

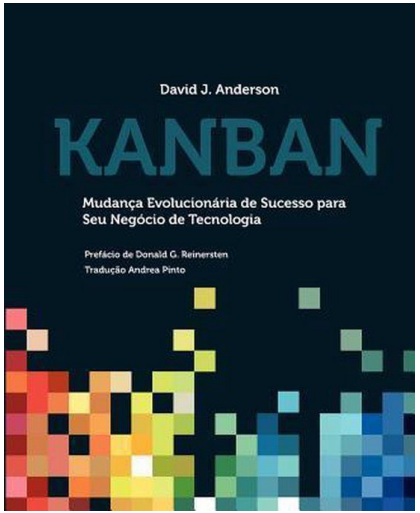
Taiichi Ohno est considéré comme le père du système de production en flux tendu de Toyota. Son inspiration lui vient du constat que les employés d'un supermarché remplissent uniquement les rayons qui sont vides. Lui vient alors l'idée de reproduire ce même concept dans les usines Toyota.

La production ne suit plus un calendrier établit à l'avance, mais suit l'évolution de la demande. Ce qui a pour effet de satisfaire plus rapidement la demande et de réduire les stocks.

David Anderson

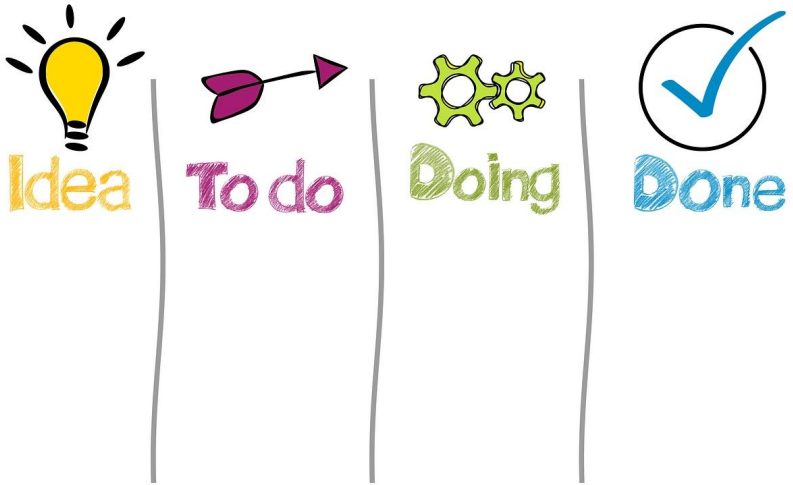
David Anderson a repris ces concepts née dans l'industrie pour l'adapter dans le secteur de l'informatique début des années 2000.

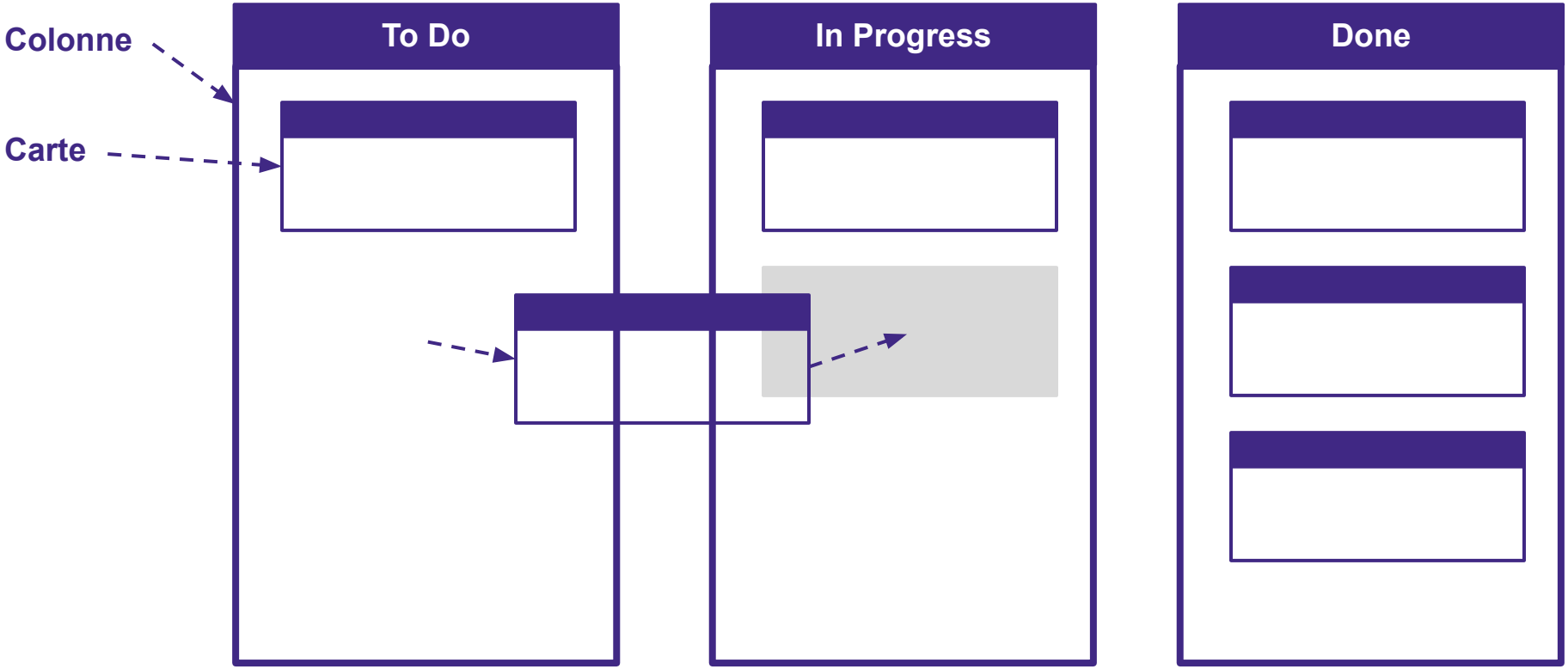
Maintenant Kanban est utilisé pour n'importe quel type de projet.



Kanban Board

Il est souvent difficile de pouvoir visualiser un workflow. L'objectif d'un tableau Kanban est visualiser le travail qui reste à faire, qui a été terminé et ce sur quoi on travaille actuellement.





BSTORM CONSULTING

Kanban Board

Kanban Board

Le board peut être en papier ou des post-it que l'on colle sur un mur ou des outils en ligne. Il suffit de rechercher sur Google "Kanban Board" pour voir la quantité d'outils disponibles. Les plus connus sont :

- Trello.
- Jira.
- Asana.
- etc.

Workflow

Un workflow correspond à toutes vos étapes de travail par lequel vous passez.

- Par défaut : To Do, In Progress, Done.

Il sera différent pour chaque équipe.

- Exemple : To Do, Analysis, Design, Development, Testing, Done.

To Do	Analysis	Design	Development	Testing	Done

Kanban Card

Une carte Kanban contient toutes les informations nécessaires pour la réalisation d'une tâche à faire. Elle contient généralement les informations suivantes

- Titre.
- Description.
- Personne(s) responsable(s).
- Date limite d'exécution.
- Statut.

On peut y ajouter beaucoup d'autres informations comme des documents ou des schéma.

La taille d'une carte Kanban

La taille des cartes ne doit pas être trop grande pour ne pas mettre des semaines pour les terminer, ni trop petite où chaque carte représente chaque action que vous devez faire sur la journée.

Avec la pratique, on finit par trouver la taille optimale pour nos cartes.

Exemple : ticket de vol

Les passagers possèdent une carte d'embarquement qui contient toutes les informations nécessaires pour chaque passager ou membre du personnel pour savoir quelle direction prendre pour rejoindre son embarcation.

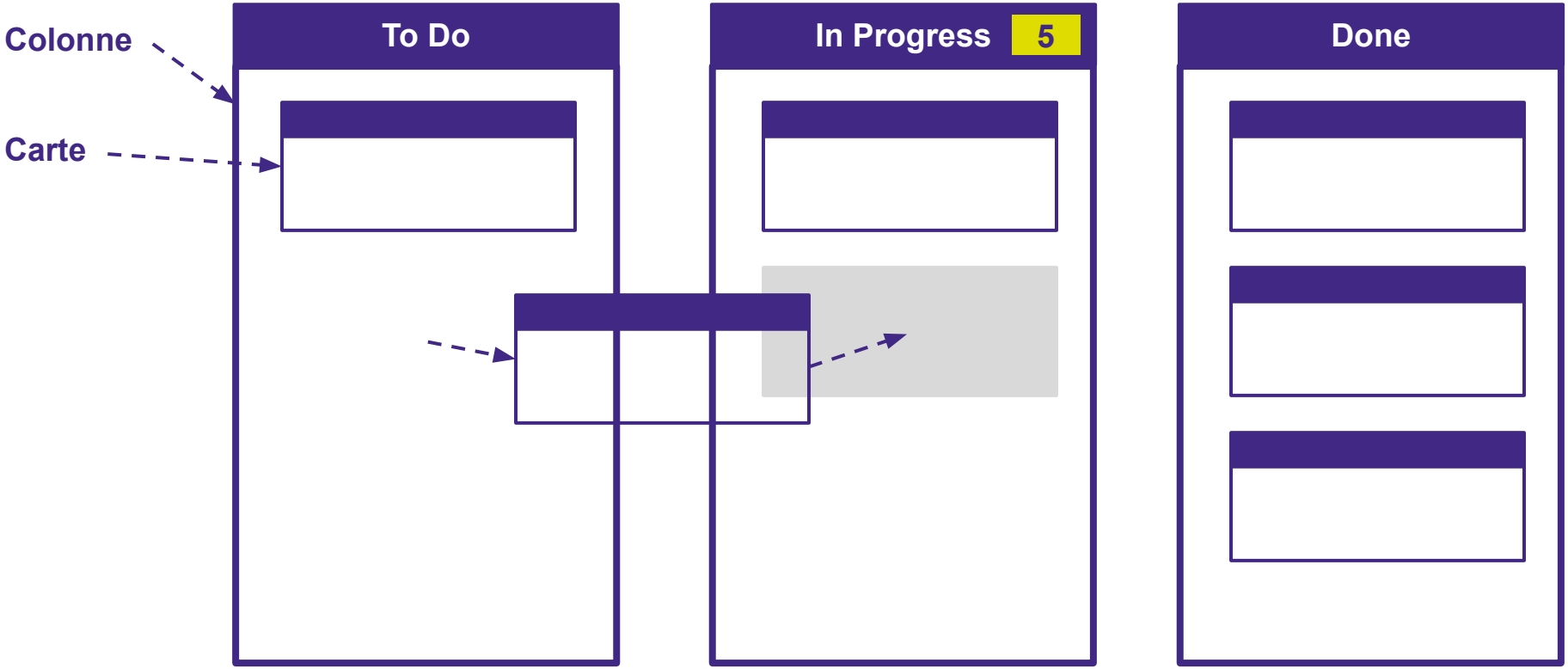
Sans utiliser cette méthode, on devrait centraliser ce processus. Des responsables de l'aéroport devrait coordonner tous les passagers individuellement.

Work In Progress Limit

Avec une méthode comme Kanban, il est facile de commencer à travailler sur trop de tâches en même temps. C'est pour cela que l'on ajoute un règle : **WIP Limit** (Work In Progress Limit).

On définit un nombre maximum de tâche sur lequel l'on peut travailler simultanément pour chaque colonne. Une fois ce nombre atteint, avant de pouvoir commencer à travailler sur une nouvelle tâche on doit en terminer une.

La plupart des outils en ligne donne la possibilité de définir cette limite.



BSTORM CONSULTING

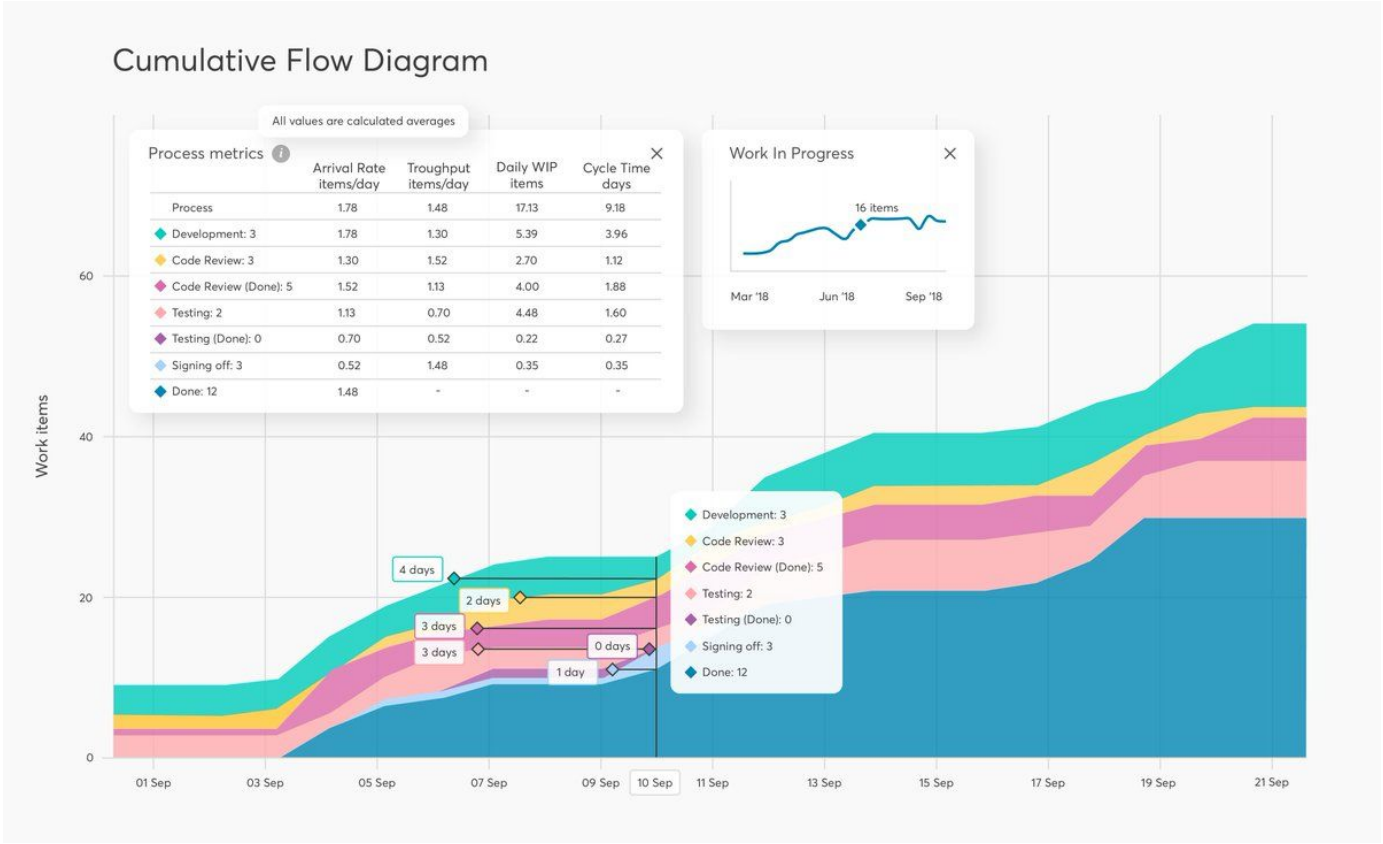
Kanban Board WIP Limit

Cumulative Flow Diagram

Lorsque l'on veut voir si l'évolution de notre workflow et identifier s'il n'y a pas un problème, on peut utiliser plusieurs techniques.

Par exemple, le "Cumulative Flow Diagram" représente l'évolution de chaque étape les unes par rapports aux autres.

- Lorsqu'une étape grandit plus vite que les autres cela peut indiquer un potentiel problème.
- Un changement dans la pente des courbes indique un changement de rythme de travail.



CONSULTING BSTORM

Cumulative Flow Diagram (par Nave pour Trello)

Lead Time

Le "Lead Time" correspond au temps pour une tâche de passer de la colonne To Do à la colonne Done. C'est donc le temps de travail.

Si on construit des tâches de taille identique (par exemple 1h ou 1j), il est alors possible de faire des plannings "précis" sur les futurs avancements.

Les 4 principes Kanban

Principe 1: Commencez par ce que vous faites maintenant.

Principe 2: Accepter de poursuivre un changement progressif et évolutif.

Principe 3: Respecter le processus actuel, les rôles et les responsabilités.

Principe 4: Encourager les actes de leadership à tous les niveaux.

Agile

Scrum

CONSULTING BSTORM

Concepts de SCRUM

Origine

SCRUM a été formalisé en 1995 par:

- Jeff Sutherland
  - http://jeffsutherland.com/
- Ken Schwaber
  - http://kenschwaber.wordpress.com/

Scrum

Définition

“A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value”

Scrum Guide

Scrum est :

- Léger
- Facile à comprendre
- Difficile à maîtriser

Big Picture

Scrum est un Framework d'Agile qui produit dans un laps de temps (**Sprint** de 1 à 4 semaines) une version de l'application qui fonctionne

- Les exigences et les priorités sont définies dans le **Product Backlog**.
- L'équipe Scrum s'auto-organise.
- A chaque fin de Sprint, la décision de livrer le produit dans l'état ou de continuer à l'améliorer est prise.

Agile vs Scrum

Agile	Scrum
philosophie	**framework**

Scrum is A Framework, Not A Process

Process

- Normatif : dicter étape à étape.
- Répétable de projet en projet.
- Idéal pour une utilisation dans un contexte bien défini.

Framework (“*cadre*”)

- Non normatif.
- Définition des tâches clés et d’une routine.
- Plus adaptable aux environnements volatiles.

SCRUM framework + notre environnement = notre process

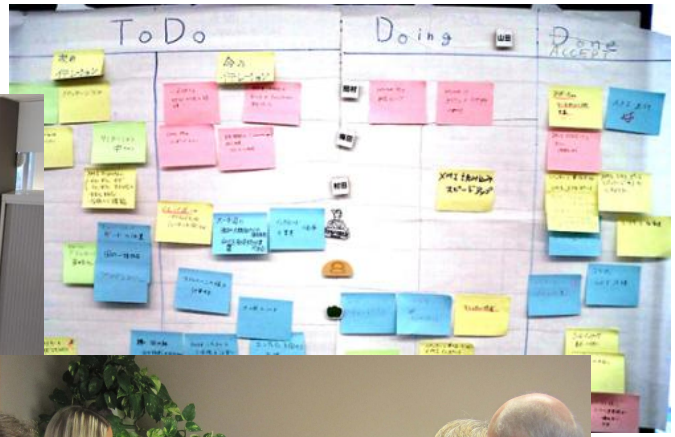
mon
SCRUM

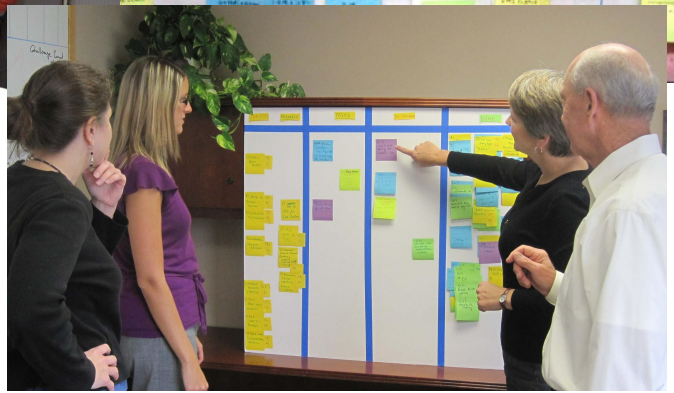
votre
SCRUM

CONSULTING BSTORM

Scrum is A Framework, Not A Process

SCRUM en images





SCRUM en images

Le cycle de vie d’un projet SCRUM

Le Processus

Un processus SCRUM consiste en 3 phases

1. **La phase Initiale**
   - Phase d'analyse.
   - Création d'un Product Backlog.
2. **La phase de Sprints**
   - Phase de développement de 1 à 4 semaines.
   - Création d'un Sprint Backlog.
3. **La phase de Clôture**
   - Préparation du produit pour une livraison.

Phase Initiale

La phase initiale aboutit à :

1. La conceptualisation et l'analyse du système.
2. La mise en place d'un Product Backlog.
    - Liste de tâches restant à effectuer.
3. La définition approximative de la date de livraison.
4. La formation et la constitution de l'équipe de développement.
5. L'analyse du risque et des coûts.

Phase Initiale

Phase courte mais qui requiert des compétences variées :

- connaissance du marché,
- des utilisateurs potentiels,
- des ressources techniques issues d'autres développement similaires,
- etc...

Phase de Sprint

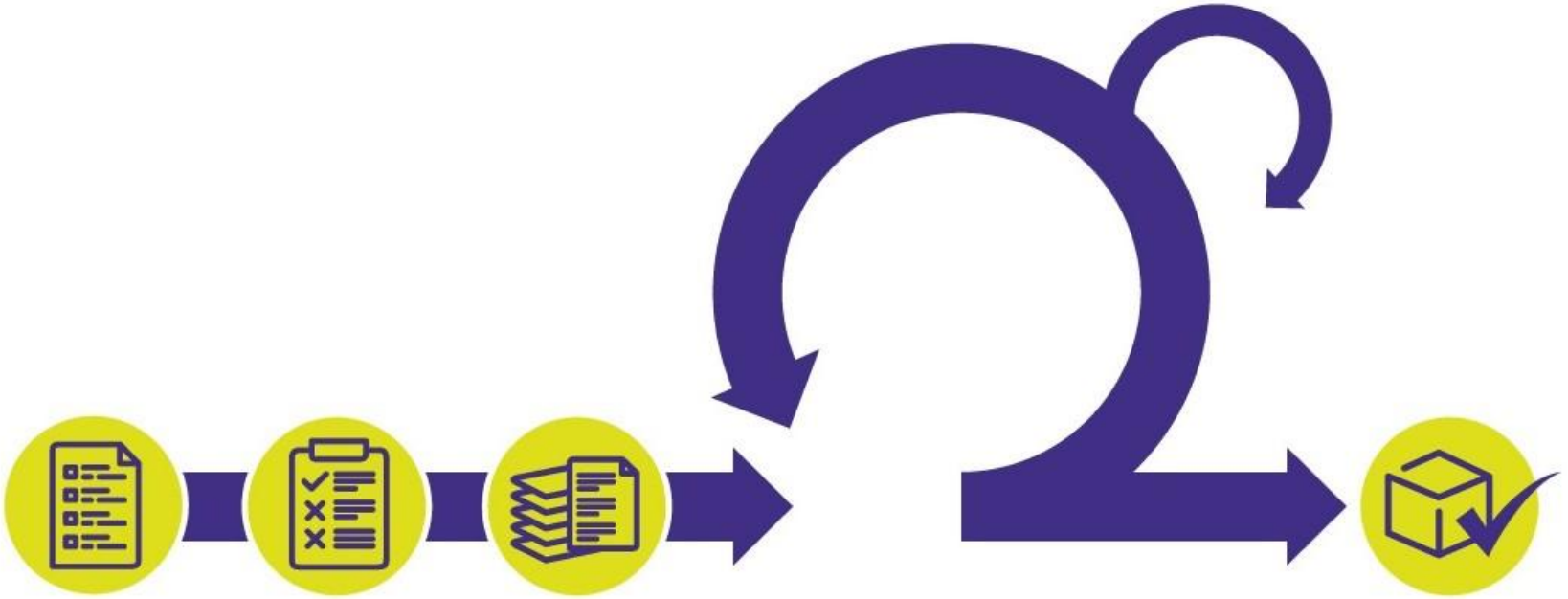
- Phase où le développement est, à proprement parlé, réalisé.
  - Analyse, conception, implémentation... de chaque élément.
- Les sprints sont guidés par une liste de tâches provenant du Product Backlog formant le **Sprint Backlog**.
- Durée de 1 à 4 semaines.
- Isolation de l'équipe de toute influence extérieure.

Phase de clôture

Préparation du produit pour une livraison :

- intégration,
- tests systèmes,
- documentation utilisateur,
- préparation de supports de formation,
- préparation de supports marketing,
- etc...

Le Processus



Les rôles SCRUM

Les rôles

1. **Product Owner**
   - Objectif : maximiser la valeur du produit.
   - Définit et partage la vision globale du produit.
2. **SCRUM Master**
   - Aide l'équipe à respecter les règles du Framework Scrum.
3. **SCRUM Team**
   - Responsable du déroulement de chaque Sprint (créer et livrer le produit).

Product Owner

Les rôles du Product Owner

- Est responsable de la gestion du Product Backlog.
- S'assure que le Product Backlog soit compris pour chacun.
- S'assure de la valeur du travail fourni par l'équipe.

Les caractéristiques

- Un seul PO par projet.
  - Une seule Vision du Produit (référant).
- Leader (communication) et Team Player (négociation).
- Visionnaire, Disponible et Qualifié.

3 facettes du Product Owner

1. **Product Management**
   - Définir et communiquer la Vision du Produit.
   - Définir la stratégie de livraison des différentes versions du produit sur le marché.
2. **Business Analysis**
   - Identifier les besoins et exigences des utilisateurs.
   - Mise à jour du Product Backlog pour refléter ces besoins et exigences.
3. **Project Management**
   - Gérer le budget du projet.
   - S’assurer du respect de chaque échéance.

Product Owner

Collaboration avec l'équipe SCRUM

- Lui même membre de la SCRUM Team
- Le Product Owner doit se trouver sur le même site que la SCRUM Team

Collaboration avec le SCRUM Master

- Rôles complémentaires
- Product Owner est responsable du "QUOI"
- SCRUM Master est responsable du "COMMENT"
- Une personne ne doit pas cumuler les 2 rôles

Un Product Owner pour deux équipes

Product Owner est une activité à plein-temps. Dans de rares cas, un PO peut s'occuper de 2 Scrum Team en même temps seulement si 3 conditions sont réunis :

1. Les deux équipes travaillant dans le même domaine.
2. Les deux équipes travaillant dans le même produit.
3. Les deux équipes ont de l'expérience avec Scrum.

Ex : une équipe de développement et l'autre sur la maintenance d'un même produit.

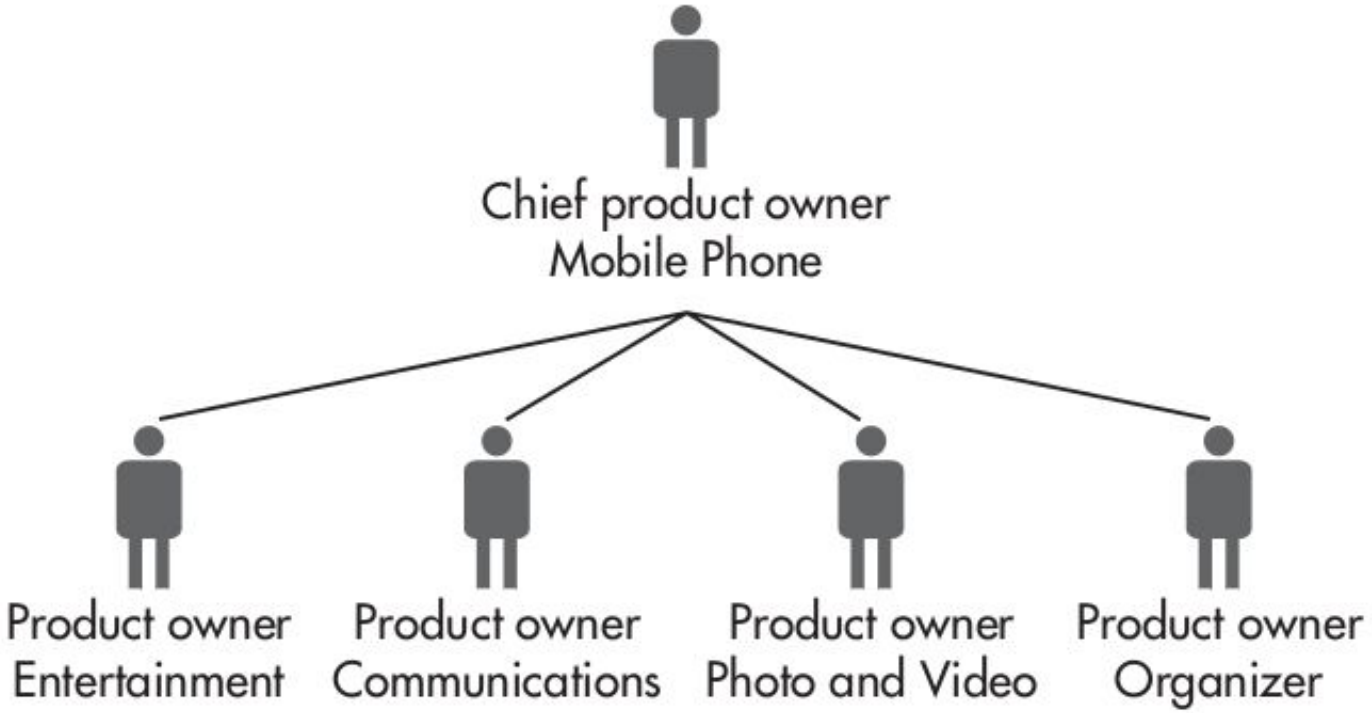
Chief Product Owner

Difficile pour un Product Owner de s'occuper de 2 SCRUM Team. Quid de projets comprenant plus de 2 SCRUM Team alors que le Product Owner est unique ?

Introduction du rôle du **Chief Product Owner**

- Responsable de coordonner différents Product Owner.
- Responsable de la vision du produit.

Chief Product Owner



Erreurs courantes

1. Ne pas donner assez de pouvoir au Product Owner
    - Il ne dispose pas assez d'autonomie
2. Surcharger le Product Owner
    - Il risque de se focaliser ce sur quoi il sera jugé, à savoir le Project Management (budget et temps), en mettant de côté l'écoute des clients et d'analyse.
3. Ne pas séparer (transversalement) les tâches du Product Owner entre différentes personnes

SCRUM Master - rôles

- Assure que l'équipe Scrum fonctionne aussi efficacement que possible en respectant les règles du Framework Scrum.
- Aide l'équipe à travailler de façon autonome et à s'améliorer constamment.
- Anime les réunions de Pré-Sprint, Scrum, Post-Sprint.
- Elimine les obstacles qui ralentissent l'équipe.
- Communique avec le management (ex. rapports d'avancement).

SCRUM Master - caractéristiques

- **Bien connaître SCRUM**
- **Bien organisé**
  - Garde l'équipe sur la bonne voie.
- **Talent de communication**
  - Pouvoir résoudre les conflits.
- **Etre respecté par l'équipe et le Product Owner**
  - Etre capable de guider sans imposer.
- **Etre respecté par l'organisation**
  - Protège l'équipe des influences extérieures.
- **Idéalement, est élu par la Scrum Team**
  - Pour les équipes expérimentées.

“Servant Leaders”

- Le SCRUM Master n'est pas un chef d’équipe ou de projet
  - Ce n’est pas la Scrum Team qui est aux services du Scrum Master mais l’inverse.
- Il ne dirige pas, il n'impose pas, il n’est pas craint.
  - Il fait partie de l'équipe.
  - Il aide l’équipe à mieux travailler.
  - Ceux qui font le travail sont les mieux placés pour déterminer *comment* faire le travail.

Backgrounds pour être Scrum Masters

1. **Développeur**
   - Respecté par ses collègues.
   - Bonne connaissance des processus de développement.
2. **Testeur**
   - Vision globale du projet.
   - Habitué au relation 1-to-1 avec chaque membre de l'équipe.
   - "Spécialiste" des bugs et erreurs de développement.
3. **Project Manager**
   - Bien organisé.
   - Bonne relation avec toute l'organisation.
   - Attention aux habitudes autoritaires !

À plein temps ?

1. **Full-time Dedicated Scrum Master**
   - Plein temps focalisé sur une seule équipe.
   - Permet d'être focus sur le projet.
   - Permet un suivi plus approfondi de l'équipe.
2. **Full-time Shared Scrum Master**
   - Plein temps partagé entre plusieurs (2 à 3) équipes.
   - Permet de partager le coût d'un Scrum Master entre plusieurs équipes.
3. **Part-time Scrum Master**
   - Temps partiel avec une seule équipe.
   - Plus proche des problématiques de l'équipe.
   - Idéal pour une équipe Scrum débutante.

L’équipe de développement

"Ceux qui sont responsable de l'exécution du travail."

- Responsable du déroulement de chaque Sprint.
  - Ce qui est mis en oeuvre pour atteindre les objectifs du Sprint.
- Impliqué dans :
  - L'estimation de la charge de travail.
  - La création du Sprint Backlog.
  - L'identification des freins à l'avancement du projet.

L’équipe de développement

Responsabilités	Caractéristiques
Créer le produit	Apprécie participer à la création du produit Dispose d’au moins une compétence nécessaire à la création du produit
S’auto-organise et s’auto-gère	Fait preuve d’initiative et d'indépendance
Est pluri-disciplinaire	A de la curiosité Désire contribuer au delà de sa zone de maîtrise Apprécie apprendre de nouvelles compétences Partage volontier sa connaissance
Est dédiée et rassemblée	

L’équipe de développement - pluridisciplinaire

Une Scrum Team doit être pluridisciplinaire pour être autonome, sans dépendance externe.

- Mettre de côté les titres qui inhibent les compétences.
  - Testeurs, développeur .NET, développeur Java…
  - Tous sont appelés développeur.
- Travailler à étendre ses compétences.
  - Apprendre quelque chose de nouveau chaque sprint.
- Se proposer pour aider un membre qui fait face à un obstacle.
- Être flexible.

L’équipe de développement - s’auto-organise

Afin de tirer avantage de la connaissance et l’expérience des membres de l’équipe

- S’engage vis-à-vis des objectifs du sprint
- Identifie ses tâches
- Estime l’effort nécessaire à chaque tâche
- Se focalise sur la communication
- Collabore
- Prend des décisions sur base de consensus
- Participe activement

L’équipe de développement - s’auto-gère

Contrôle la manière dont elle travaille afin d’assurer le succès du projet

- Autorise la fluctuation du leadership
- S’appuie sur les outils et processus agile
- Rapport régulièrement et de manière transparente l’avancement du projet
- Règle les problèmes au sein de l’équipe
- Créer une convention d’équipe
- Inspect and Adapt

First Among Equals

Le Product Owner et Scrum Master sont des rôles importants mais ils ne sont pas les chefs de la Team Scrum.

- La Team Scrum est autonome.
  - Ce qui permet au PO et SM de se concentrer sur le travail plutôt sur la gestion au quotidien de l'équipe.
- Le Product Owner est considéré comme "**First Among Equals**" (premier parmi ses pairs)
  - A le dernier mot, en cas de conflit ou indécision..
  - Puisque seul responsable de la vision produit.

Taille réduite (recommandée) de l'équipe

3 à 9

membres

Facilite la bonne communication

Maintien l'unicité de l'équipe

Évite la création de sous-équipes

Encourage la pluridisciplinarité, la communication en face-à-face…

Prise d'initiatives

Réussi ou échoue en tant qu'équipe.

Autres: Management et Client

Management

- Responsable de la décision finale
- Impliqué dans le choix du Product Owner
- Surveille l'avancement du projet
- Peut augmenter / réduire le Backlog en concertation avec le Product Owner

Client

- Participe à l'élaboration du Product Backlog

Les rôles

Les meetings

Meetings

4 types de meetings

1. Sprint Planning Meeting
   - Planification en début de chaque sprint
2. Daily Scrum
   - Réunion quotidienne informelle de toute l'équipe
3. Sprint Review
   - Présentation aux clients
4. Sprint Retrospective
   - Evaluation de l'efficacité du sprint

Sprint Planning Meeting

Réunion de pré-Sprint organisée par le SCRUM Master en 2-temps

- **1ère Phase**
  - Clients, utilisateurs, management, product owner et Scrum Team établissent le Sprint Backlog
  - Décider des objectifs et de la fonctionnalité du prochain Sprint
- **2ème Phase**
  - Le Scrum Master et la Scrum Team organisent le déroulement du Sprint
  - Décide de la manière dont l'incrément de produit est mis en œuvre pendant le sprint.

TIMEBOX: 2 heures par semaine de SPRINT

Daily Scrum

Quoi ?

- Réunion quotidienne informelle de toute l'équipe

Quand et où ?

- Toujours à la même heure et au même endroit

Qui ?

- N'interviennent que les personnes qui travaillent effectivement sur le développement
- Les extérieurs y sont invités pour suivre l'avancement du projet

Daily Scrum

Objectifs

1. Partager les connaissances acquises
2. Faire un point sur l'avancement
3. Donner au management une certaine visibilité sur la progression du projet

Contrôle continu et empirique via 3 questions quotidiennes

1. *Qu'est ce qui a été fait pendant la journée ?*
2. *Que reste-t-il à faire ?*
3. *Quels sont les obstacles qui gênent l'avancement du projet ?*

Daily Scrum

Le Scrum Master prend les décisions que l'équipe est incapable de prendre par elle même et s'engage à apporter une solution à tout ce qui entrave le développement du projet

TIMEBOX: 15 à 30 minutes max

Sprint Review

- Réunion informelle
- L'équipe présente au client ce qui a été développé pendant les 30 jours précédents via une démonstration
- Confronter les résultats du travail de l'équipe avec la complexité et le chaos de l'environnement dans lequel l'application sera utilisée
- Décision de release ou non

TIMEBOX: 1 heure par semaine de Sprint

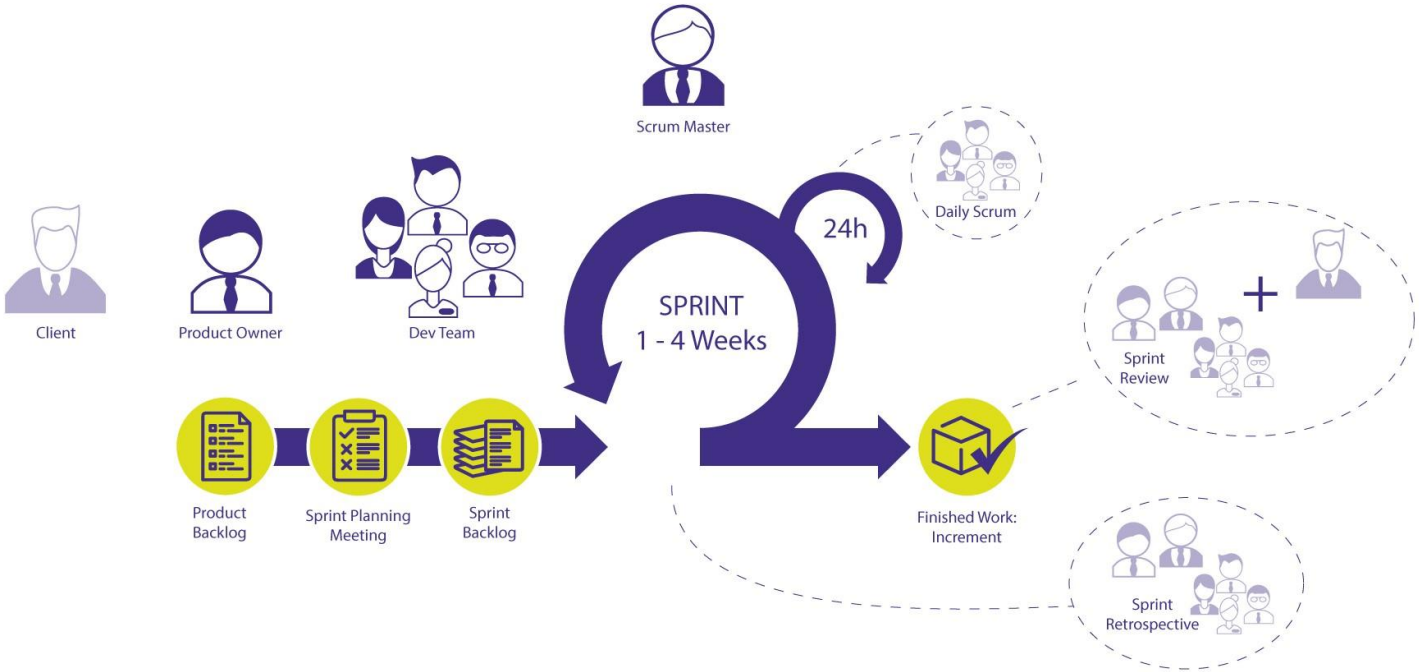
Sprint Retrospective

Réunion ayant lieu après le Sprint Review et avant le Sprint Planning Meeting.

- Le but du Sprint Rétrospective est de :
  - Inspecter la manière dont s'est déroulée le Sprint précédent quant aux personnes, relations, processus et outils utilisés.
  - Identifier et ordonner les éléments majeurs qui se sont déroulés ainsi que les améliorations potentielles.
  - Créer un plan pour implémenter des améliorations quant à la manière de travailler de l'équipe Scrum.

TIMEBOX: 45 minutes par semaine de Sprint.

Les meetings



User Story

Détaillé de manière appropriée

Les éléments du product backlog sont des user story

En tant que [rôle], **je veux** [action] **afin de** [but]

Exemple:

"En tant qu'utilisateur qui ferme l'application, je souhaite être invité à enregistrer tout ce qui a changé depuis la dernière sauvegarde afin de pouvoir conserver le travail utile et de supprimer le travail erroné."

Workshop d'écriture des User Stories

- Participants: le Product Owner, partenaires additionnels (ex. utilisateurs), (la SCRUM Team)
- Pas forcément à chaque Sprint
- Génération sur base de Brainstorming
  - On commence avec des "**epics**" et on découpe
- Ecrire un maximum de stories possible
  - Certaines directement implémentable
  - Certaines resteront temporairement des "epics"

Bonnes pratiques

1. Ecrire pour un seul utilisateur.
2. Ecrire à la forme active.
3. Ne pas numéroter les cartes.
4. Ecrire des user stories fermées.

Pourquoi utiliser des User Stories ?

1. Suscite la communication verbale.
2. Compréhensible pour chacun.
3. Taille appropriée pour la planification.
4. Fonctionne bien dans des processus itératifs.

INVEST

User Story doit être :

1. **I**ndependent.
2. **N**egotiable.
3. **V**aluable.
4. **E**stimable.
5. **S**mall.
6. **T**estable.

Independent

La user story doit être autonome, de manière à ce qu'il n'y ait pas de dépendance inhérente à une autre user story.

Négociable

Stimule la conversation

L'utilisateur peut payer avec une carte de crédit.	L'utilisateur peut payer avec une carte de crédit.
Remarque: Acceptez Visa et MasterCard. Considérez American Express. Sur achats de plus de 100 $ demander l'identifiant de la carte le dos de la carte. Le système peut dire quel type de carte il est des deux premiers chiffres du numéro de carte. Collecter le mois et année d'expiration.	Remarque: Acceptez Visa et MasterCard. Considérez American Express. Interface utilisateur: pas besoin d'un champ séparé pour le type de carte.
TROP DE DÉTAIL	**JUST ENOUGH**

Valuable

Aux utilisateurs et aux acheteurs, PAS aux développeurs.

Estimable

Unités utilisées

- Points Story (relatif)
- Jour-hommes (absolut)

Qu'est-ce qui rend une histoire difficile à estimer?

- Manque de connaissance du domaine
- Manque de connaissances techniques
- La taille est trop grande

SMALL

- Un utilisateur peut planifier des vacances.
  - Trop grand
- Un utilisateur peut poster son CV.
  - Trop grand

- Un utilisateur peut ajouter son CV.
- Un utilisateur peut mettre à jour son CV.
- Un utilisateur peut marquer les CV comme inactifs.

Stratégies de décomposition

1. Par étapes d'un Workflow
   - L'utilisateur accomplit une tâche selon un workflow bien établi. On découpe les stories par étapes qui seront développées de façon incrémental.
2. Par scénario
   - On obtient une User Story pour le scénario principal, le cas où tout se passe bien, on en a d'autres pour les cas d'erreurs ou les scénarios alternatifs : quand il se passe x, quand il se passe y.
3. Par séquence dans un scénario
   - Le cas est plus précis, on découpe cette fois-ci une séquence au sein d'un scénario…

Stratégies de décomposition

4. Par opérations
   - Souvent le mode de décomposition le plus évident … Le CRUD (create, Retrieve, Update, Delete) est un bon exemple. Il est souvent utile de découper ou d'en faire deux en même temps…créer un compte, le consulter, le modifier et le supprimer.
5. Par format ou type de données
   - On joue sur le type d'objet (ex : compte titres, espèces… messages en français, anglais, espagnol…)
6. Par type d'entrée, sortie ou configuration
   - Des variations d'un point de vue matériel ou non, selon les configurations mais aussi en termes de moyens de saisie. Cela peut se jouer aussi au niveau de l'interface…

Stratégies de décomposition

7. Par Persona ou rôle
    a. Cette fois-ci, on décompose les user stories en fonction du rôle et de celui qui va utiliser le produit, le fameux « en tant que… ». Pour cela, on peut s'appuyer le user story mapping, une activité menée au début du projet pour définir le backlog de produit et la roadmap produit.
8. Par niveau de connaissance
    a. Le niveau de connaissance acquis sur une fonctionnalité est un bon critère de décomposition…Une story pour ce qui est connu, une autre là où c'est moins maîtrisé. Cela peut déboucher sur un spike (une user story un peu particulière orientée exploration)

Stratégies de décomposition

9. Par niveau de complexité
   - Une user story va par exemple décrire une fonctionnalité dans son mode de réalisation le plus simple, d'autres suivront par un niveau de complexité plus grand
10. Par niveau de qualité attendu
   - Performance, Sécurité, Utilisabilité… ces exigences non fonctionnelles constituent le plus souvent des conditions de satisfaction pour des user stories spécifiques mais elles peuvent permettent également de distinguer des user stories entre elles (ex : afficher en moins de 60 sec, moins de 30 sec ; données en temps réels ou non…)

Testable

- Critères de test concrets.
- Habituellement écrit au verso de la carte.
- La plupart des tests devraient être automatisés.

- L'écran de démarrage disparaît peu de temps après l'exécution du programme.
  - Pas assez concret
- L'écran de démarrage disparaît dans les 4 secondes.
  - Concret & Testable

Testable

- Les tests sont utilisés pour :
  - La spécification
  - Déterminer qu'une user story est terminée
- Ecrire les tests avant de coder
  - Lors des discussions Product Owner / Développeurs
  - Au début du Sprint
- Les tests sont principalement spécifiés par le responsable du produit
- Les tests font partie du processus

Exemple

Une entreprise peut payer une offre d'emploi avec une carte de crédit

- Testez avec Visa, MasterCard, American Express
- Testez avec les bons, les mauvais et les numéros manquants
- Test avec des cartes expirées
- Testez avec différents montants d'achat (y compris une limite supérieure à la carte)

Définition du Done

Afin d’éviter des problèmes de compréhension, il est important de se mettre d’accord de ce qu’est une taché terminé.

La **définition de Done** est une liste d'exigences auxquelles une User Story doit adhérer pour que l'équipe puisse la qualifier de terminé.

Définition du Done

Les critères sur la définition du Done sont définis avant le premier Sprint par :

1. Le Product Owner
2. Le Scrum Master
3. L'équipe de développement

Un élément du product backlog est considéré comme achevé lorsque :

1. Il est implémenté
2. Il est testé
3. Il est intégré
4. Il est documenté

Définition du Done

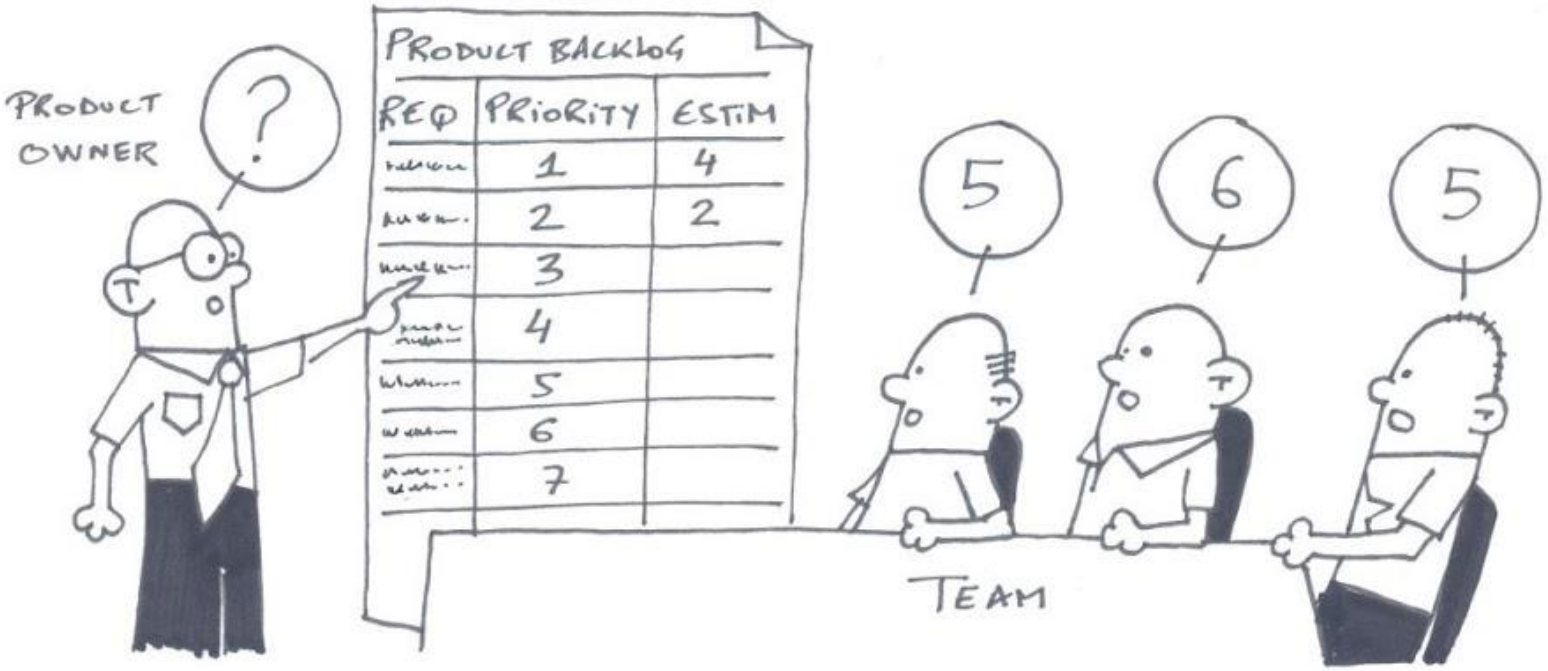
Une bonne définition doit :

1. **Être atteignable**
   - Représente de façon réaliste les capacités de l'équipe
   - Ex: s'assurer que l'équipe ait accès à toutes les ressources nécessaires
2. **Être fait de manière collaborative**
   - Tous les participants de la tâche doivent avoir la possibilité de donner leurs avis
3. **Être flexible**
   - Ouvert au changement à mesure que l'équipe trouve de meilleures façons de travailler

Product Backlog

Product Backlog - Big Picture

1. Une liste de tous les éléments nécessaires à la bonne livraison du produit.
2. Le Product Owner est responsable de la création et du suivi du bon déroulement de cette liste.
3. Tout membre de l'équipe Scrum peut ajouter des éléments à la liste en collaboration avec le Product Owner.



BSTORM CONSULTING

Product Backlog

Les qualités du Product Backlog

4 Qualités: **DEEP**

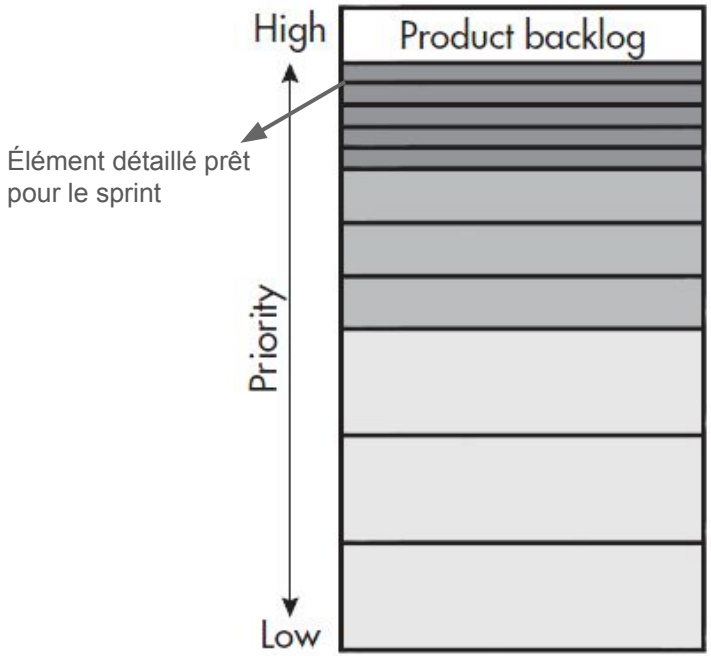
1. **D**etailed Appropriately
   - Les éléments hautement prioritaires sont décomposés et raffinés en vue de la réunion du planning de Sprint (Detailed Appropriately)
2. **E**stimated
   - Estimation des éléments via le planning poker
3. **E**mergent
   - Ajout, modification, suppression d'éléments
4. **P**rioritized
   - Prioritisation (plus important en haut)

Détaillé de manière appropriée

Les éléments les plus prioritaires sont les éléments les plus détaillés

Implique que les exigences soient décomposées et raffinées durant l'entièreté du projet

Les éléments du PB sont des **User Stories**



Emergent

Le contenu du product backlog change continuellement :

- De nouveaux éléments sont ajoutés
- Des éléments sont modifiés
- Des éléments sont supprimés

Priorisés

1. **Valeur apportée aux clients / utilisateurs**
   - *« Quel est le bénéfice financier à développer cette fonctionnalité ? »*
2. Coût de développement estimé
   - *« Quel est le coût de développement ? De maintenance ? De formation des utilisateurs ? »*
   - *« Quel est le coût d'un changement ? »*
3. Opportunité d'apprentissage pour l'équipe
   - *« L'implémentation de cette fonctionnalité nous permet-elle d'apprendre ou de développer de nouvelles compétences ? »*
   - *« Cette fonctionnalité va-t-elle améliorer la productivité de mon personnel ? »*

Triage des exigences

- **Problème**
  - Nombre important d'exigences élicitées
  - Ressources limitées
  - Impossibilité de faire tout, tout de suite
- **Solution**
  - Donner un ordre de priorité (prioritisation)
  - Différentes approches possibles
    - Approche Qualitative
      - **MoSCoW**
      - **Kano Model**
      - **Now-How-Wow Matrix**
    - Approche Quantitative
      - **Matrix Prioritization / VOLERE**

Modèle Kano

Le succès d'un produit ou service dépend de sa capacité à résoudre efficacement un ou plusieurs problèmes des clients.

Toutes ces exigences ne vont pas donner la même satisfaction aux clients.

Le modèle Kano est une représentation des besoins en 5 principales catégories en fonction de la satisfaction qu'ils procurent.

Kano Model

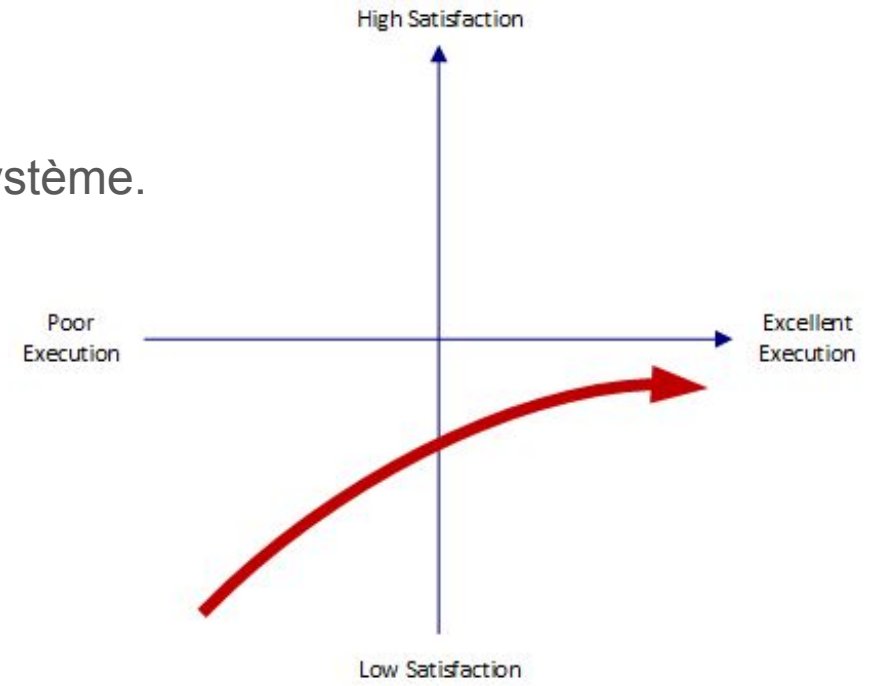
Kano a identifié cinq catégories différentes de produits basées sur la perception du client.

Author	Category 1	Category 2	Category 3	Category 4	Category 5
Kano (1984)	Must-Be	Attractive	One-Dimensional	Indifferent	Reverse
Cadotte & Turgeon (1988)	Dissatisfier	Satisfier	Critical	Neutral	
Brandt (1988)	Minimum Requirement	Value-Enhancing	Hybrid	Unimportant	
Venkitaraman and Jaworski (1993)	Flat	Value-Added	Key	Low	
Brandt and Scharioth (1998)	Basic	Attractive	One-Dimensional	Low-Impact	
Llosa (1997 and 1999)	Basic	Plus	Key	Secondary	
Pohl and Rupp (2011) [3]	Dissatisfier	Delighter	Satisfier		
KanoModel.com	Basic	Excitement	Performance	Indifferent	Reverse

Kano Model

Must-Be (Must)

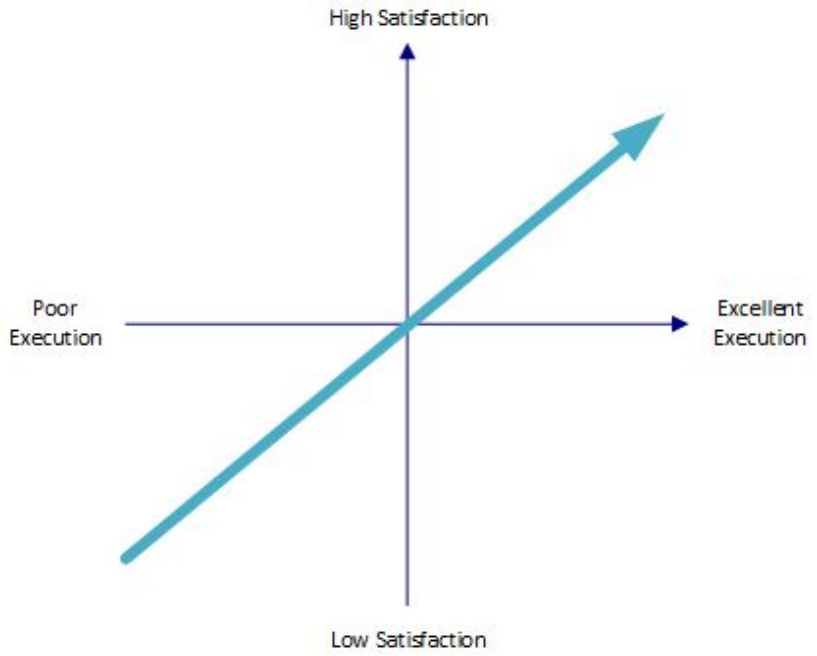
- Ce que le client prend pour acquis.
- Cost of entry pour pouvoir lancer le système.



Kano Model

One-Dimensional (Should Have)

- Ce que le client exige explicitement, et souhaite voir vérifié dans la solution finale.
- Engendre une augmentation proportionnelle de satisfaction.



Kano Model

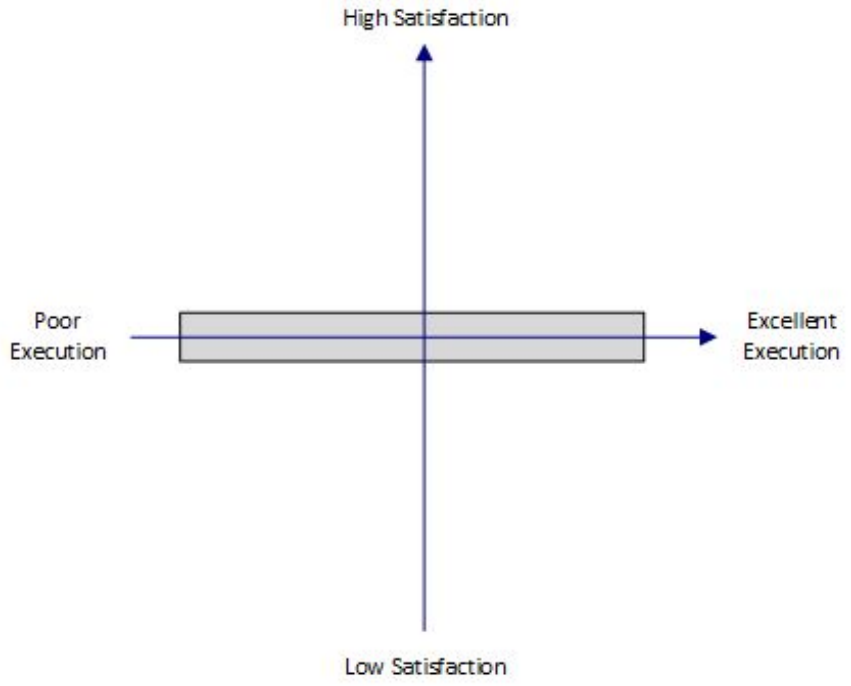
Attractive (Could Have)

- Ce que le client n'exige pas forcément, mais apprécie fortement.
- Engendre une augmentation non proportionnelle de satisfaction.

Kano Model

Indifferent

- Ce que le client n'exige pas forcément, et ne souhaite spécialement voir vérifié dans la solution finale.
- N'engendre pas d'augmentation de la satisfaction.



Kano Model

Reverse

- Ce que le client souhaite voir être évité dans la solution finale.
- Engendre une diminution proportionnelle de la satisfaction.

Modèle Kano

Fonctionnalité n

Si cette fonctionnalité était présente, que ressentiriez-vous ?	*« Ça me ferait plaisir »*	
	« Ce serait un minimum »	X
	« Je n'ai pas d'avis »	
	« Je l'accepterais »	
	« Ca me dérangerait »	
Si cette fonctionnalité n'était pas présente, que ressentiriez-vous ?	*« Ça me ferait plaisir »*	
	« Ce serait un minimum »	
	« Je n'ai pas d'avis »	
	« Je l'accepterais »	X
	« Ca me dérangerait »	

		Question dysfonctionnelle				
		Plaisir	Minimum	Neutre	Acceptation	Dérangement
Question fonctionnelle	Plaisir	?	L	L	L	E
	Minimum	R	I	I	I	O
	Neutre	R	I	I	I	O
	Acceptation	R	I	I	I	O
	Dérangement	R	R	R	R	?

O Obligatoire **R** "Reverse" (Inversée)

E Exprimée **?** Incertaine

L Latente **I** Indifférente

MoSCoW

Technique de priorisation sur base d'une échelle nominale (pas de chiffres)

- Extrêmement fréquente car simple et multi-contexte
- Elle se base sur 4 grandes catégories
  1. **Must**
  2. **Should**
  3. **Could**
  4. **Won't**

Note : Les o dans MoSCoW sont ajoutés uniquement pour rendre l'acronyme prononçable.

MoSCoW

- **Must Have**
  - Les choses que le système doit vérifier afin d'aller de l'avant.
  - Si vous demandez à un client « Quid si X est manquant? » et que la réponse est « Dans ce cas nous ne continuons pas », alors vous avez trouvé un "must ».
- Si
  - Livrer à la date cible sans X n'a aucun sens.
  - Livrer sans X n'est pas légal.
  - Livrer sans X n'est pas sur (risque).

MoSCoW

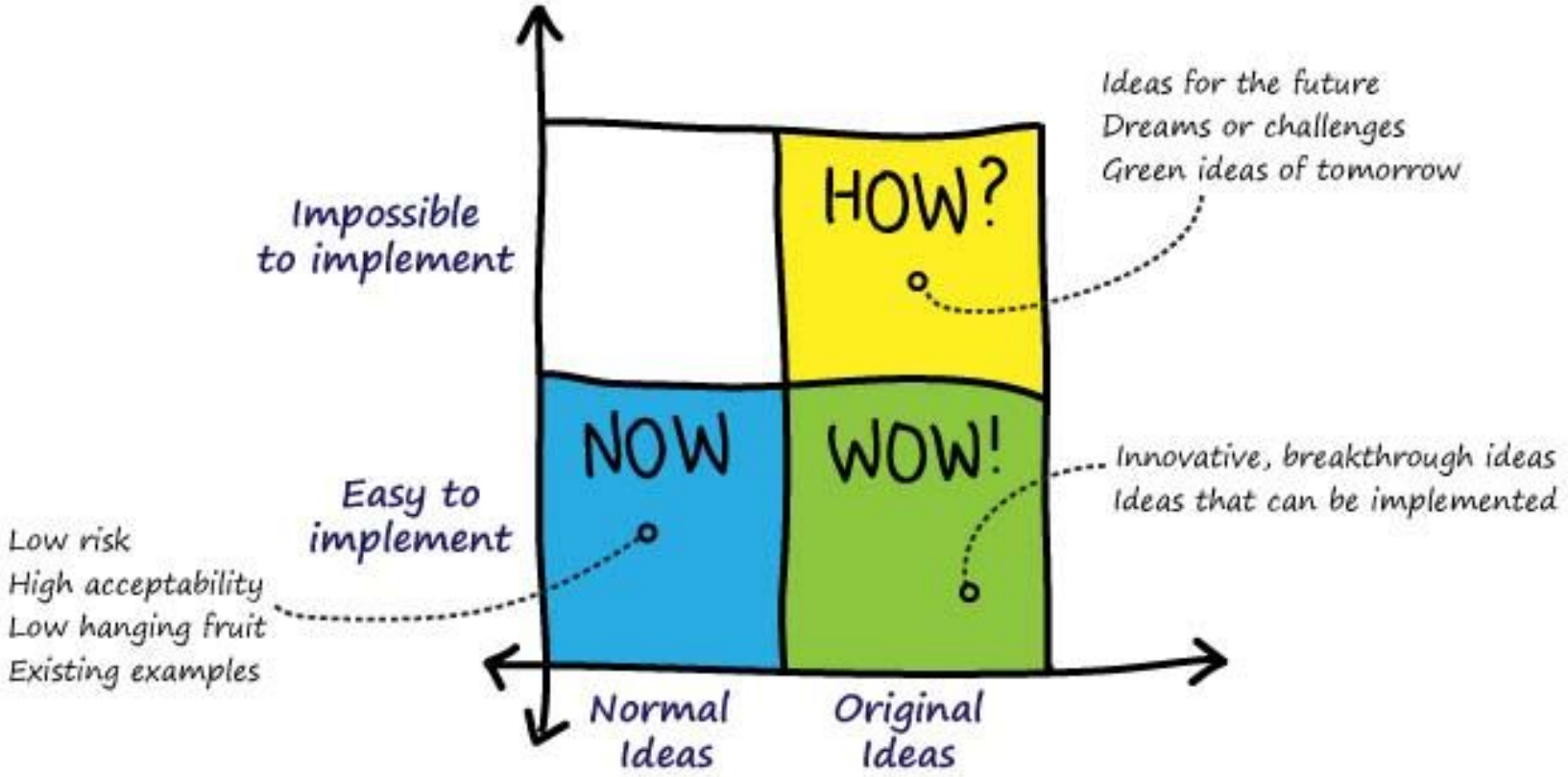
- **Should Have**
  - Diffère du Must Have dans le sens où il n'est pas absolument nécessaire pour aller de l'avant dans le projet, mais son absence entraînerait un niveau plus élevé d'insatisfaction de la part du client.
  - Important mais pas vital.
  - Peut être dommageable pour l'ensemble de laisser de côté, mais la solution est encore viable.
  - Peut nécessiter une sorte de solution de contournement, par exemple gestion des attentes, etc.

MoSCoW

- **Could Have**
  - Les éléments qui sont souhaitables, mais qui ont un impact plus faible si on les laisse sortir du scope du projet.
  - Recherché ou souhaitable, mais moins important.
  - Moins d'impact si laissé de côté (par rapport à Should).
- **Won't Have** (ou Would Have)
  - Les éléments identifiés comme étant souhaitables ou précieux, mais qui ne sont pas assez importants pour être implémentés dans le prochain release.
  - Ils ne sont pas invalides: ils sont tout simplement des éléments à ne pas prendre en compte "pour le moment ».

Now-How-Wow Matrix

- Préjugé
  - Les idées créatives sont souvent des idées qui sont difficiles à mettre en oeuvre
- Faux !
  - Certaines sont compliquées à implémenter.
  - D'autres offrent une valeur ajoutée très importante pour un coût relativement faible.
- Now-How-Wow aide à sélectionner et prioriser les idées selon leur faisabilité et leur degré d'innovation



BSTORM CONSULTING

Now-How-Wow Matrix

Volere Prioritisation Spreadsheet											
Copyright c The Atlantic Systems Guild 2002											
Requirement/Product Use Case/Feature	Number	Factor - score out of 10	%Weight applied	Factor - score out of 10	%Weight applied	Factor - score out of 10	%Weight applied	Factor - score out of 10	%Weight applied		Total Weight
		Value to Customer	40	Value to Business	20	Minimise Implementation Cost	10	Ease of Implementation	30	Priority Rating	100
Requirement 1	1	2	0.8	7	1.4	3	0.3	8	2.4	4.9	
Requirement 2	2	2	0.8	8	1.6	5	0.5	7	2.1	5	
Requirement 3	3	7	2.8	3	0.6	7	0.7	4	1.2	5.3	
Requirement 4	4	6	2.4	8	1.6	3	0.3	5	1.5	5.8	
Requirement 5	5	5	2	5	1	1	0.1	3	0.9	4	
Requirement 6	6	9	4	6	1.2	6	0.6	5	1.5	6.9	
Requirement 7	7	4	2	3	0.6	6	0.6	7	2.1	4.9	
Requirement											

Priorisés

1. Valeur apportée aux clients / utilisateurs
   - *« Quel est le bénéfice financier à développer cette fonctionnalité ? »*
2. **Coût de développement estimé**
   - *« Quel est le coût de développement ? De maintenance ? De formation des utilisateurs ? »*
   - *« Quel est le coût d'un changement ? »*
3. Opportunité d'apprentissage pour l'équipe
   - *« L'implémentation de cette fonctionnalité nous permet-elle d'apprendre ou de développer de nouvelles compétences ? »*
   - *« Cette fonctionnalité va-t-elle améliorer la productivité de mon personnel ? »*

Priorisés

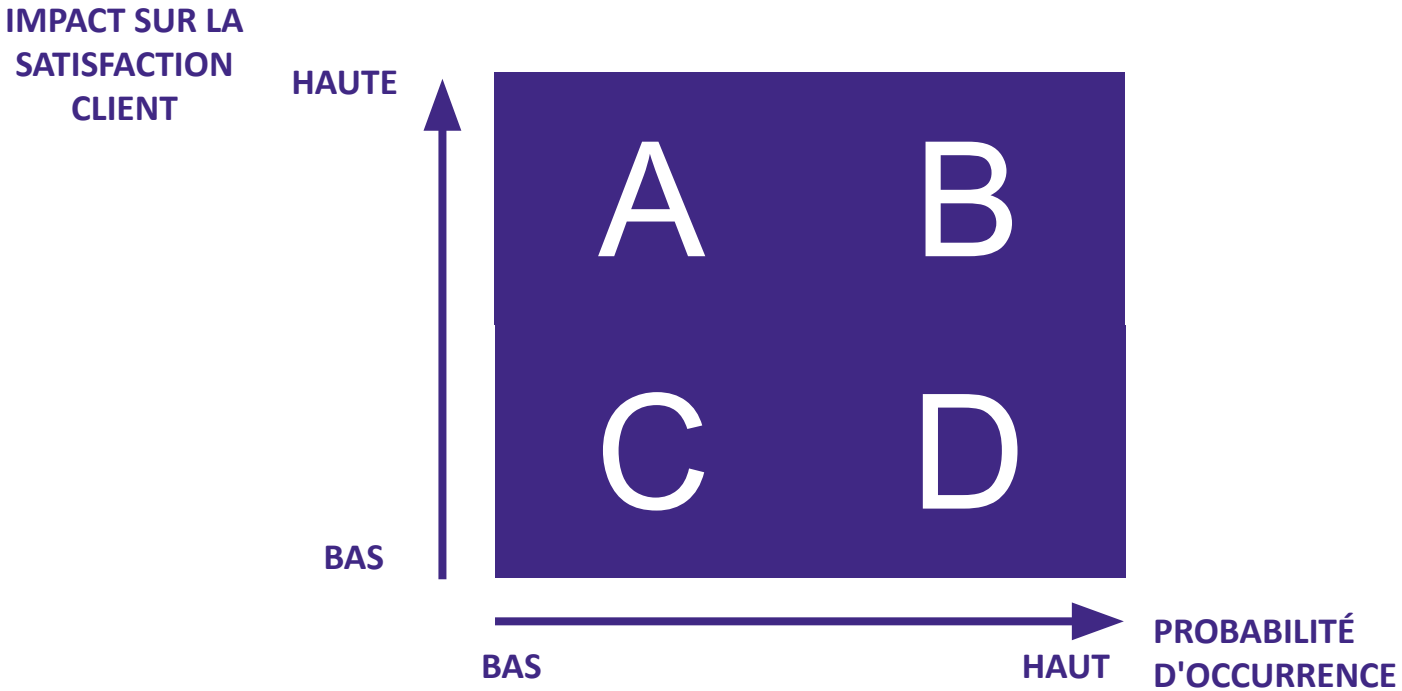
1. Valeur apportée aux clients / utilisateurs
   - *« Quel est le bénéfice financier à développer cette fonctionnalité ? »*
2. Coût de développement estimé
   - *« Quel est le coût de développement ? De maintenance ? De formation des utilisateurs ? »*
   - *« Quel est le coût d'un changement ? »*
3. **Opportunité d'apprentissage pour l'équipe**
   - *« L'implémentation de cette fonctionnalité nous permet-elle d'apprendre ou de développer de nouvelles compétences ? »*
   - *« Cette fonctionnalité va-t-elle améliorer la productivité de mon personnel ? »*

Risque de développement

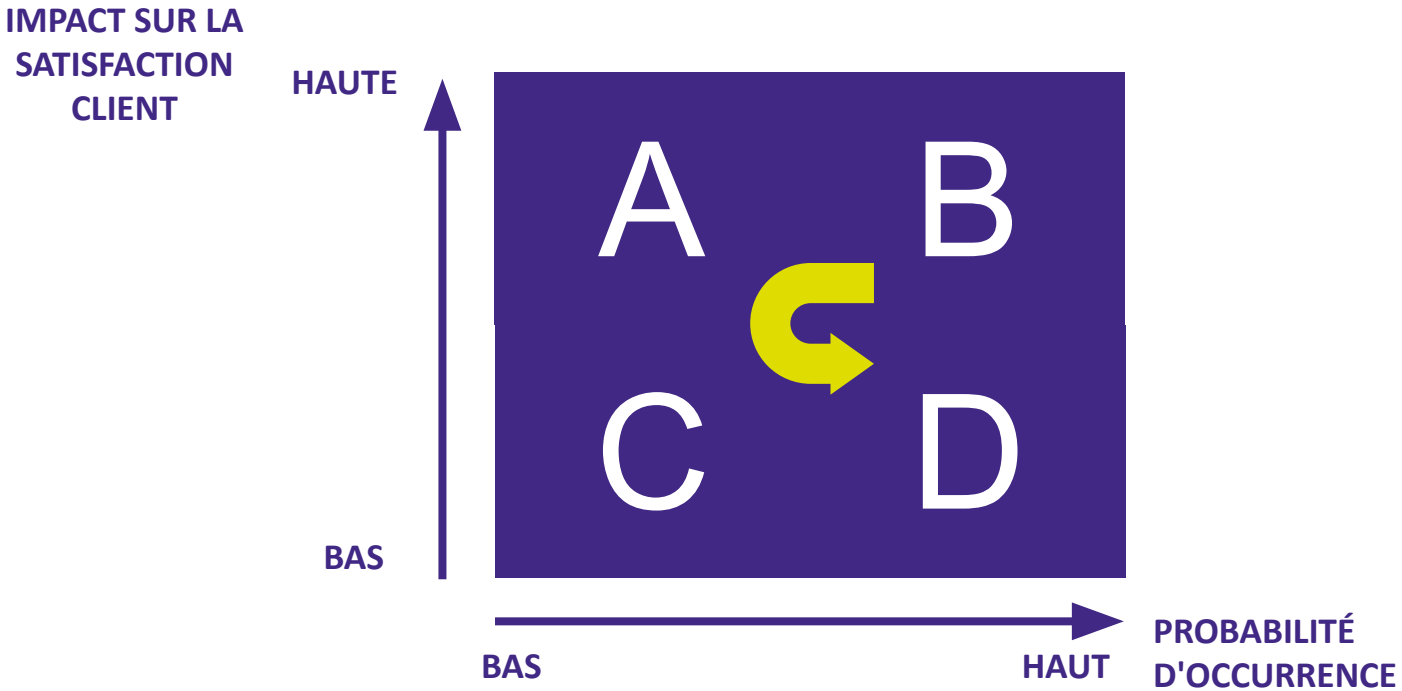
Le risque de développement

- *« Cette fonctionnalité nous expose-t-elle à davantage de risques ? Ou, au contraire, nous permet-elle de nous confronter au risque ? »*
- *« Cette fonctionnalité va-t-elle impacter les processus métier ? »*
- *« Cette fonctionnalité va-t-elle susciter de la frustration ou de la résistance ? »*
- *« Quel est le préjudice si cette fonctionnalité n'est pas implémentée ? »*

Analyse du risque



Analyse du risque



Exigences non-fonctionnelles

Ex.: performance, robustesse, utilisability...

Influence:

- Interface utilisateur
- Architecture
- Choix technologiques
- ...

Exigences non-fonctionnelles

Description sous forme de contraintes

- Ex.: Le système doit pouvoir répondre à n'importe quelle requête en moins d'une seconde

Distinction entre les exigences globales et locales

- Globales: doivent être définie avec la vision du produit
- Locales: peut être définie et attachée avec la user story affectée

Etendue du Product Backlog dans de larges projets

Utilisé un seul Product Backlog

- Permet d'éviter les problèmes de synchronisation

Préparer les éléments pour les 2 à 3 Sprints suivants

- Les nombres d'éléments détaillés est donc plus important

Proposer différentes vues pour les différentes teams SCRUM

Estimer l’effort et planifier

Eléments Estimés

Connaître la charge de travail aide à donner des priorités aux éléments.

Estimation grossière exprimée en nombre **Story Points**.

- Ne correspond pas à une unité de temps précise mais à une unité relative.
  - On compare le “temps” d’une tâche par rapport à une autre.
  - Cette tâche correspond à 2 ou 3 points ? On compare avec les autres tâches dont on a attribué 2 ou 3 points.
- Les estimations sont réalisées par les membres de la Scrum Team.
- Le Product Owner et le Scrum Master ne doivent ni participer, ni influencer l'estimation.

CURSE

L’un des principaux défis pour les équipes qui adoptent Scrum ou une autre méthodologie Agile est le concept de Story Point.

Généralement les équipes commencent à utiliser assimiler ces Points à une journée, donc 7 Points une semaine, etc.

On se rend très vite compte que c’est incomplet et trompeur.

CURSE

CURSE est un acronyme pour

- **C** pour la complexité.
  - C’est la complexité, la complexité, l’implication, l’intégration, etc. du travail d’une histoire.
- **U** pour l'incertitude (“*Uncertainty*”) du travail à effectuer.
  - L'équipe peut ne pas bien connaître la technologie ou ne pas savoir comment un élément particulier de l'histoire peut avoir un impact sur l'architecture ou la conception.
- **R** est pour le risque.
  - Cela englobe le degré de dangerosité d'un changement ou d'une amélioration demandé pour l'ensemble du système, de la conception ou de l'architecture.

CURSE

CURSE est un acronyme pour

- **S** pour l'ampleur ("Scope").
  - La portée est la quantité de travail ou l'ampleur des changements particuliers.
- **E** pour l'effort.
  - C'est ce que nous associons généralement aux Story Points. Cela représente le travail réel qui va être fait, la mise en œuvre de la user story.

Suite de Fibonacci

Les Story Points sont des nombres de la Suite de Fibonacci

- Suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent.
- 0, 1, 1, 2, 3, 5, 8, 13, 21, etc.
- Pourquoi ?
  - Parce que au plus une tâche va prendre du temps, au moins on va être précis dans son estimation de temps.

Planning Poker

Planning Poker

- Objectif
  - Réussir à générer des estimations acceptables le plus rapidement possible

Planning Poker

1. Le Product Owner présente la User Story
   - Pas technique - uniquement fonctionnelle
2. La Scrum Team peut demander des éclaircissements
3. La Scrum Team estime et vote pour un nombre de Story Point
4. En cas de désaccord, on demande aux plus basses valeurs et plus hautes valeurs de justifier leur choix.
   - Objectif : lancer la discussion
   - Chaque discussion est dans une time-box. Toutes les x minutes, on revote jusqu'à ce qu'on ait un accord.
5. A la fin les éléments de même taille sont groupés ensemble pour valider leur valeur relative (triangulation)

Planning Poker

Planning Poker

Quelques bonnes pratiques:

1. Time-box : spécifier un intervalle de temps par tour de table (ex.: 2min max) et le chronométrer
2. Éviter d'influencer l'estimation avec des "Je pense que ce sera vite fait" ou "Ca risque de prendre des semaines… si pas des mois"
3. Ne jamais prononcer son estimation tant que tous les autres membres n'ont pas estimé
4. Toujours obtenir un consensus plutôt qu'une moyenne
5. Ceux qui votent sont ceux qui feront le boulot (le Product Owner ne vote pas)

Les Releases

La problématique de la planification

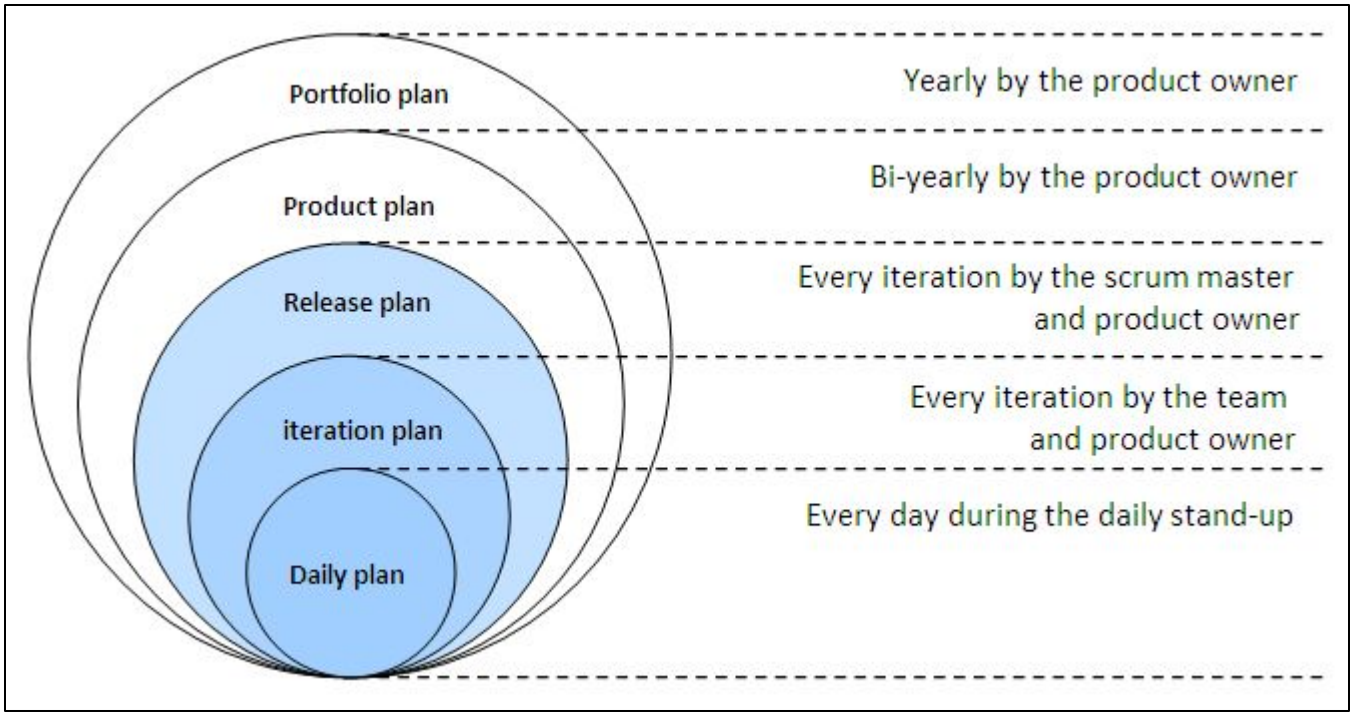
Pourquoi planifier :

1. Outil d'aide à la décision.
2. Outil de pilotage.
3. Support de communication.

5 niveaux de planification

1. **Etablissement de la vision**
   - Livraison finale
2. **Etablissement d'une roadmap**
   - Les différentes releases
3. **Planification d'une release**
   - Les différents Sprint aboutissant à la release
4. **Planification d'un SPRINT**
   - Les différentes user stories à réaliser durant cette itération
5. **SCRUM**
   - Réajustement du planning

Planning Onion



Release Plan

Major Feature	Sprint 1	Sprint 2	Sprint 3	Sprint 4 to 6	Quarter 2	Quarter 3	Year 2
Authen-tication	Login Screen		Security Questions		SSO Integration with Partner Sites		
	SSL Encryption						
Master	Product Master						RFID Implemen-tation
	Rate Master						
Order Entry		Product Selection				Recom-mended Products	Sponsored Look-ups
		Product Preview	Product Comparison				
				Product Review			
Checkout		Checkout		Coupons			Reward Points
				Order Track			

Durée des sprints

Entre 1 semaine et 4 semaines.

- Sprint court
  - Plus facile.
  - Réduction des risques...
  - … mais plus de réunion.
- Sprint long
  - Plus difficile.
  - Plus de risques...
  - … mais moins de réunion.

Estimer les durées

Principe de **vélocité** (v)

- La somme des story points (quantité de travail) qu'une équipe est capable de développer durant une itération
- Valeur hypothétique au début du projet et s'affine au fur et à mesure des itérations

Estimation durée du projet

- Après une première itération, on peut estimer une durée réaliste du projet
- Durée du projet = Somme Story Point / Vélocité

Les releases

Trois variables de décision à considérer pour planifier une release

1. **Temps**
   - *La date de lancement est-elle fixe ?*
2. **Coûts**
   - *Le budget de développement est-il fixe ?*
3. **Fonctionnalités**
   - *Un certain niveau de fonctionnalité doit-il être assuré pour une release ?*

Fixer les trois variables est impossible.

Fixer une date de lancement

- Déterminer une fenêtre d'opportunité.
- Déterminer une "*timebox*" identique pour toutes les releases.
  - Améliore l'innovation.
  - Améliore les estimations (coûts, planning...).
  - La détermination du budget est direct si la composition de l'équipe reste stable.
    - (Hypothèse: travail est coût principal).

Fixer un prix (enveloppe budgétaire)

Arrêt du développement lorsque :

1. Toutes les fonctionnalités sont implémentées.
2. Le budget est consommé.

Fixer les fonctionnalités

Ne prend pas en compte l'émergence des exigences.

Limite la capacité d'adaptation de l'équipe aux besoins du client.

Fixer un prix et un ensemble de fonctionnalités

- A éviter.
- Séparer le contrat en deux projets consécutifs.
  a. Créer la vision du produit et partiellement l'implémenter en 2 ou 3 sprints.
  b. Créer un planning de releases sur la base du Product Backlog avec un budget réaliste.

Burndown Chart

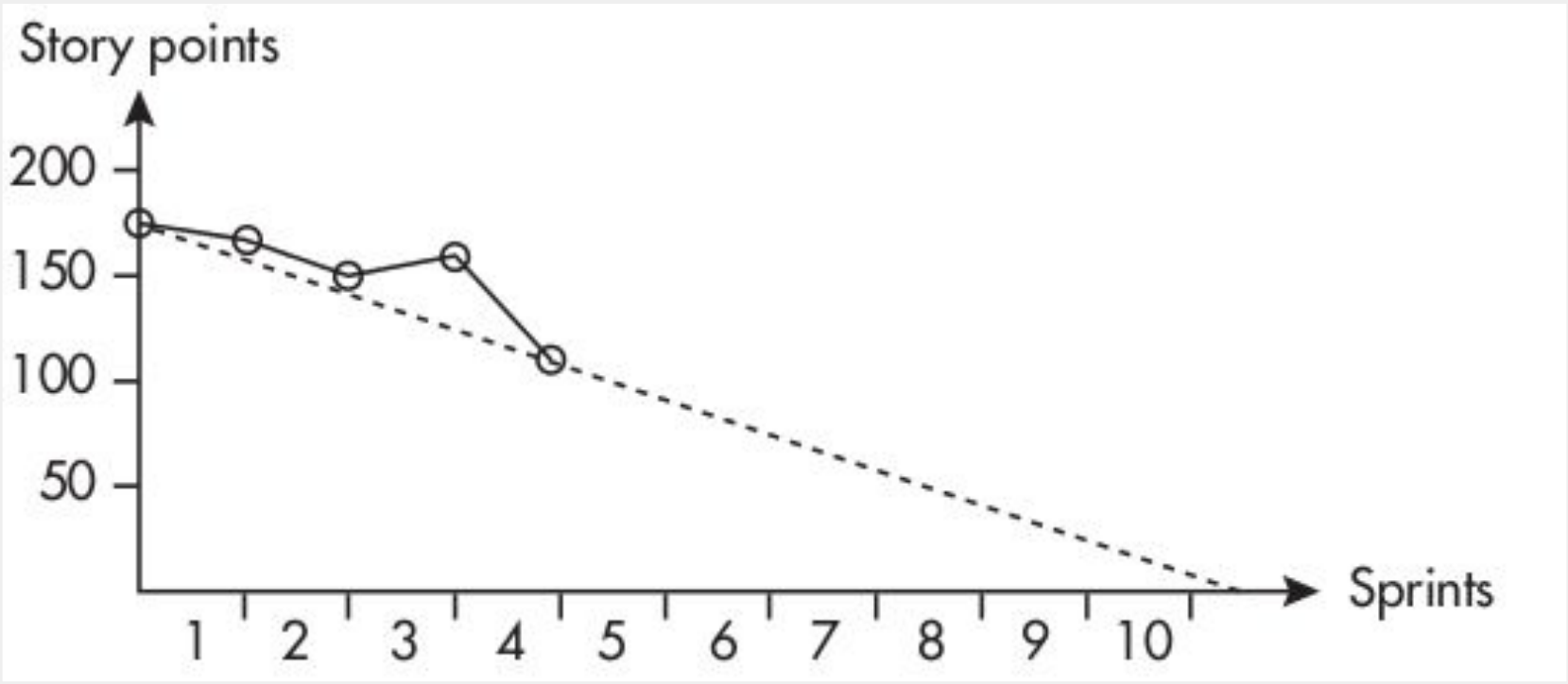
Représentation graphique de l'évolution de quantité de travail restante par rapport au temps sur une période de temps donnée.

- **Axe verticale** : quantité de travail restant (en Story Point)
- **Axe horizontale** : temps restant (en nombre de Sprint)

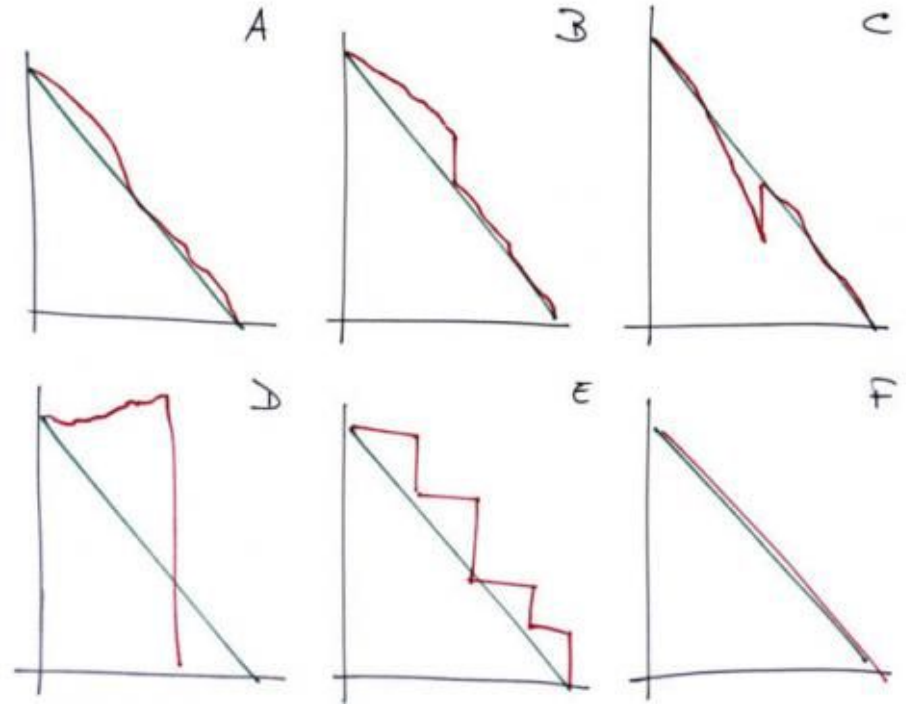
On peut faire un Burndown Chart par :

- Projet.
- Sprint.

Burndown Chart



Burndown Chart



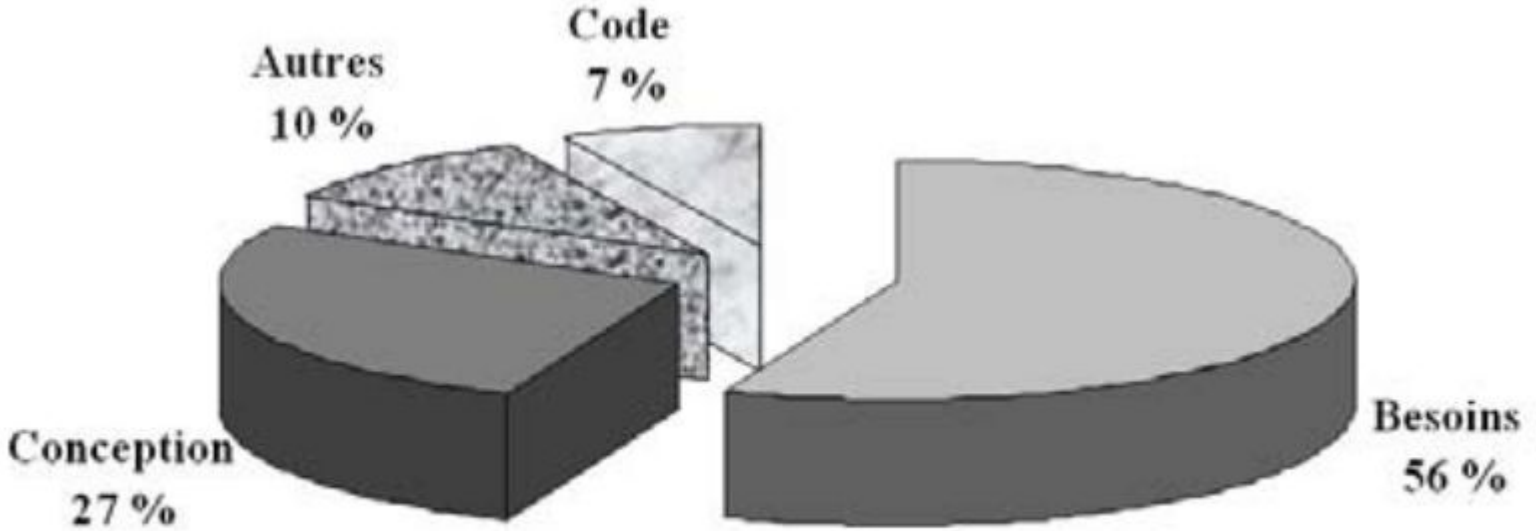
Agile

Product Vision

CONSULTING BSTORM

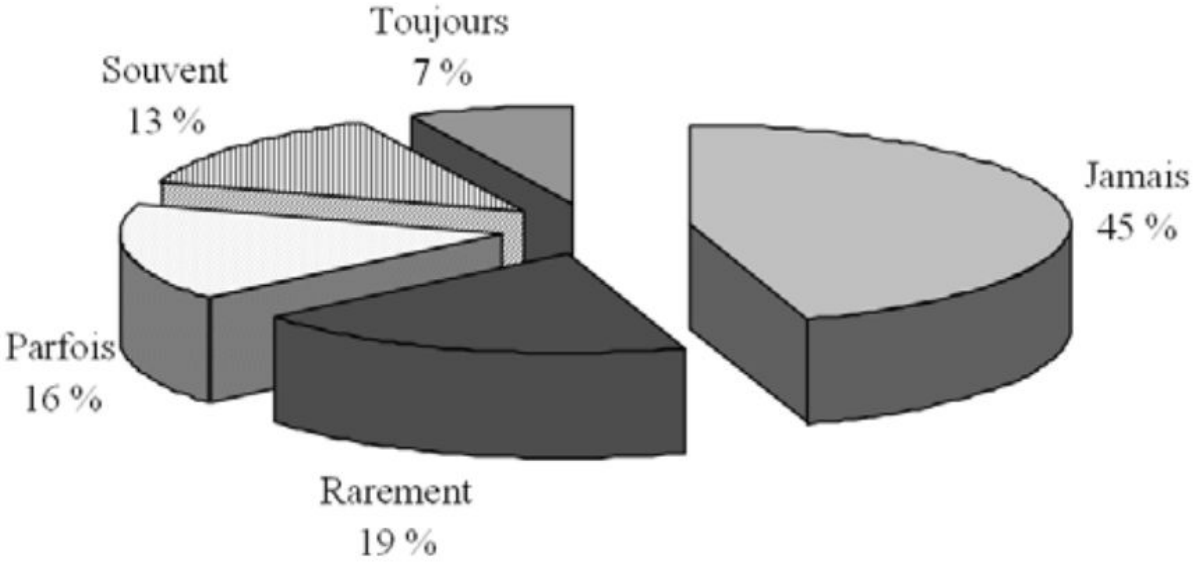
Le Produit

Origines des défauts logiciels



Source : J Johnson, Keynote speech, XP2002 (Sardaigne, Italie)

% de fonctionnalités réellement utilisées



Source : J Johnson, Keynote speech, XP2002 (Sardaigne, Italie)

Les besoins - difficultés du recueil

Recueillir les besoins présentent 3 problèmes majeurs :

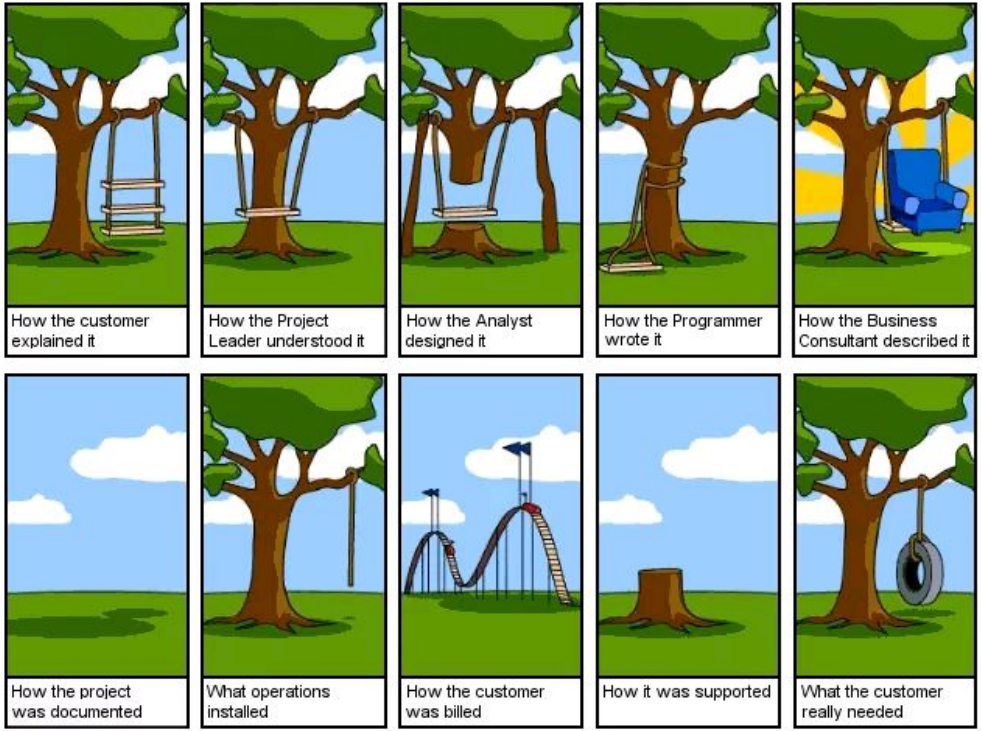
1. Une mauvaise communication
   - Communication entre équipe (Business Analyst)
   - Équipe de développement emploie un langage trop technique
2. L'illusoire exhaustivité
   - “Plus” ne veut pas forcément dire “meilleur”
   - Avoir les dernières technologies du moments (buzzword)
3. La défaillance du client
   - Les clients ne savent pas toujours ce qu’ils veulent
   - Des clients peuvent avoir des besoins différents

Les besoins - mauvaise communication

Divergences possibles entre :

1. La représentation qu'a le client de son futur produit
2. La difficulté éventuelle qu'il rencontre à le décrire
3. L'interprétation possible de cette description par l'équipe de réalisation
4. L'outil qui lui est livré au final

Les besoins - mauvaise communication



Les besoins - illusoire exhaustivité

- Le marché est instable et l'organisation doit être réactive face à ses fluctuations
- Priorités différentes entre les besoins
- Après analyse une idée peut s'avérer inutile, trop coûteuse ou secondaire
- L'introduction d'une nouvelle demande peut impliquer la renonciation d'un autre besoin

La Vision Produit

Comment se prémunir de ces problèmes ?

Vision Produit, un outil de communication

Pour qu'une Scrum Team puisse travailler de manière autonome, elle doit comprendre ce que l'on essaye de faire. Le Product Owner définit et communique une vision de produit qui explique :

1. **Quoi ?** (quelle solution pour quel problème)
    a. Ce que le produit doit faire pour répondre au mieux à une problématique.
2. **Pour qui ?**
    a. Quelles sont les différents utilisateurs que l'on essaye de cibler.
3. **Pourquoi ?**
    a. Raisons (internes) pour lesquelles on a décidé de lancer le projet.

The Product Vision Board

Avant d'avoir une bonne vision de produit, il faut d'abord se poser les bonnes questions. Pour plus de facilité, on peut utiliser des outils comme le **Product Vision Board** qui permet définir notre vision.

- Favoriser la collaboration en le faisant en équipe avec un tableau blanc.

5 .VISION

Quelle est la motivation pour créer le produit ?
Quel changement positif devrait-il apporter ?

1. Target Group	2. Needs	3. Product	4. Business Goals
À quel marché ou segment de marché le produit s'adresse-t-il ? *Qui sont les clients et les utilisateurs cibles ?*	*Quels problèmes le produit résout-il ?* *Quels avantages offre-t-il ?*	*De quel produit s'agit-il ?* *Qu'est-ce qui le distingue de la concurrence ?* *Est-il possible de développer le produit ?*	*Comment le produit va-t-il profiter à l'entreprise ?* *Quels sont les objectifs de l'entreprise ?*

5 .VISION

Créer un outil de facturation simple pour aider les petites et moyennes entreprises.

1. Target Group

PME

- *Moins de 500 K € de revenus*
- *Moins de 30 employées*

2. Needs

1. *Comprendre leurs sources de profit*
2. *Anticipation et gestion des taxes*
3. *Dépenser plus efficacement*

3. Product

1. *Gratuit*
2. *Importation financière automatisée*
3. *Catégorisation automatique des dépenses*

4. Business Goals

1. *Augmenter la rentabilité*
2. *Meilleur visualisation des factures*

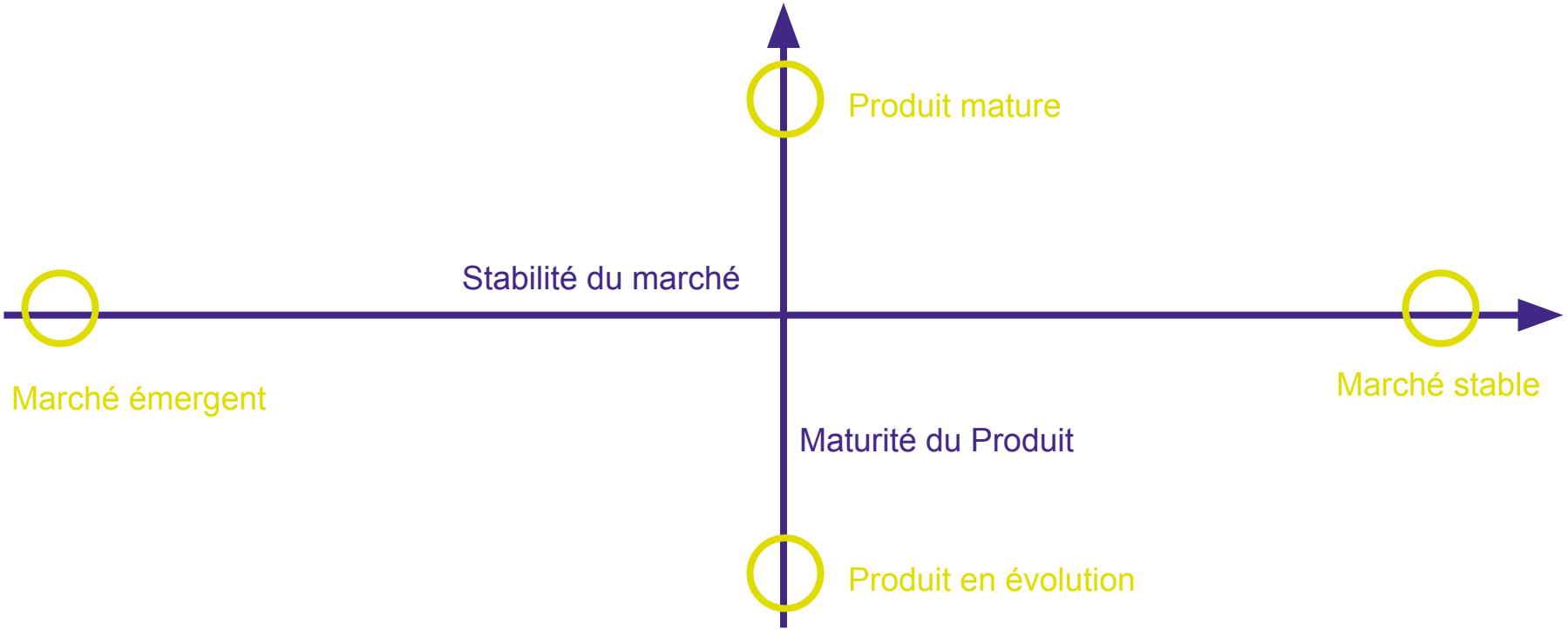
BSTORM CONSULTING

Product Vision Board : exemple

Comprendre le marché

Le succès d'un produit ne dépend pas seulement de la qualité du produit mais aussi du marché dans lequel on se trouve.

Comprendre le marché



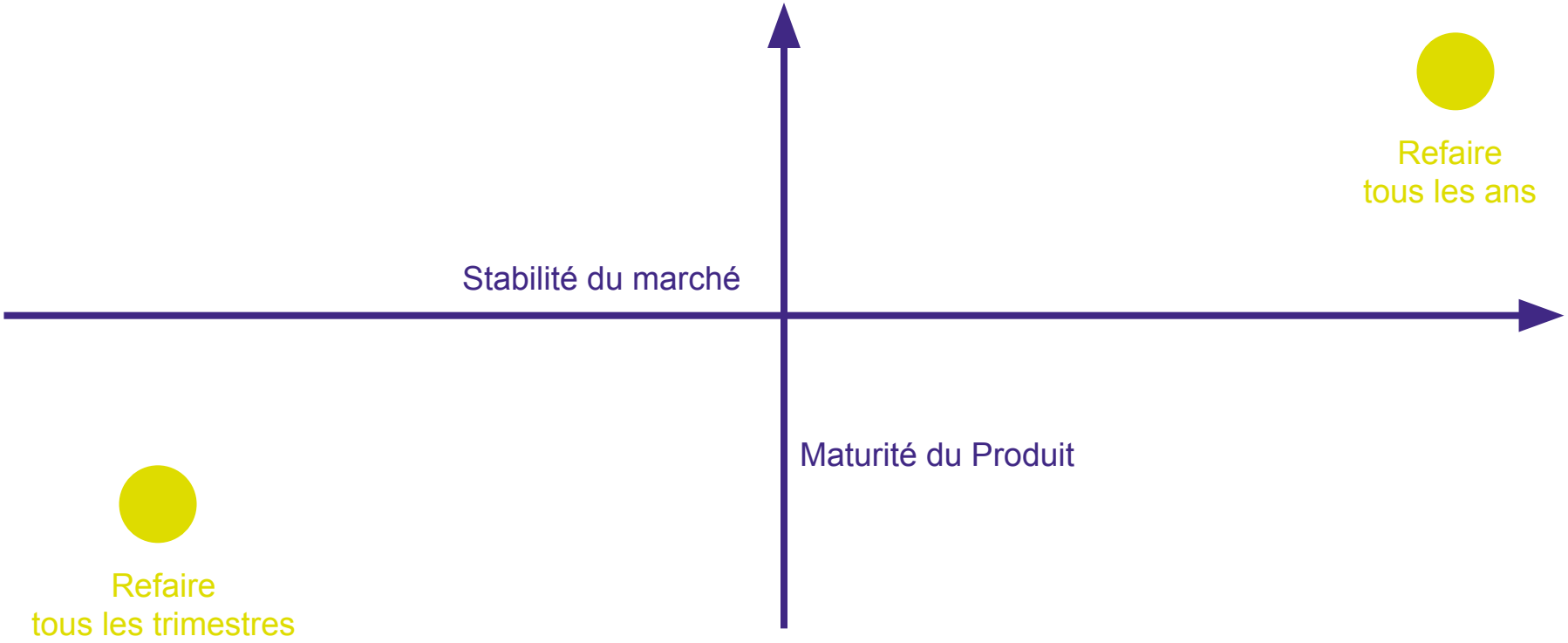
Love the problem, not your solution

La première vision correspond à la compréhension que l'on a initialement du problème. Au fur et à mesure des itérations et apprentissages, on comprend mieux le problème que l'on essaye de résoudre, notre solution doit donc s'adapter.

En fonction du marché et de la maturité du produit, il faut refaire la vision du produit afin de s'assurer que l'on développe toujours quelques choses qui correspond à un besoin.

- **1x par an** : marché stable et produit mature.
- **1x par trimestre** : marché instable et produit en évolution.

Love the problem, not your solution



Les qualités d'une bonne Vision

1. Partagée et unifiée
    - La vision doit être partagée et comprise par la Team
    - Permet d'aligner les efforts fournis par la Team
2. Large et engageante
    - Guide le développement mais laisse la place à la créativité
3. Brève et concise
    - Facile à comprendre (**30 sec elevator pitch**)
    - Ne contient que l'information critique au succès du produit

Minimum Viable Product

- Créer une vision = Projection du futur
- Capacité de prédiction limitée
- Nécessité de prévoir une version minimale du produit

- Exemple:
  - Newton vs iPhone

SIMPLICITÉ !

Minimum Viable Product

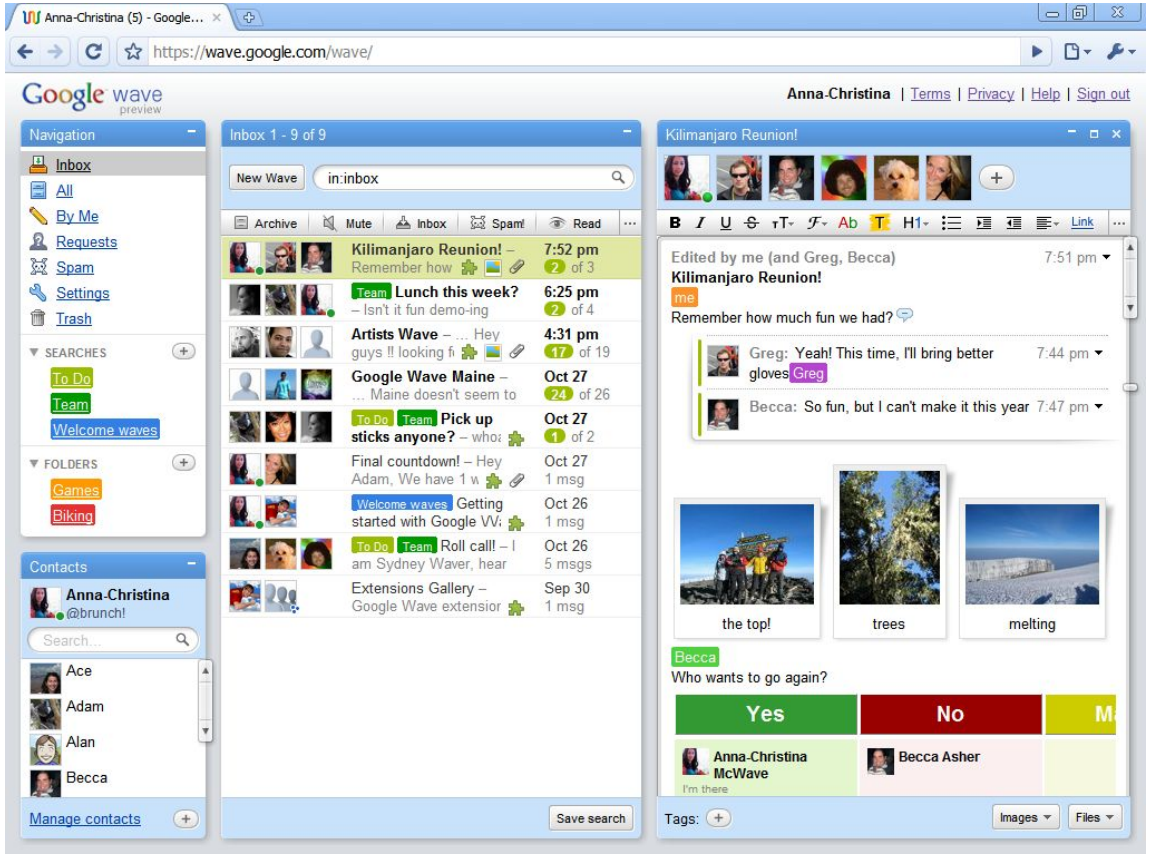
- Concentration sur les besoins cruciaux du consommateur
- N'a pas tenté d'offrir toutes les fonctionnalités offertes par les concurrents
  - Pas de copier-coller
  - Pas de possibilité d'envoyer un message à plusieurs destinataires
  - Pas de kit de développement

google.com

Google Chrome

CONSULTING BSTORM

Le produit Simplicité



" Google Wave vous permet de communiquer et de collaborer en temps réel "

CONSULTING BSTORM

Le produit Simplicité - Contre-exemple

Google Wave

Pourquoi Google Wave n'a pas fonctionné ?

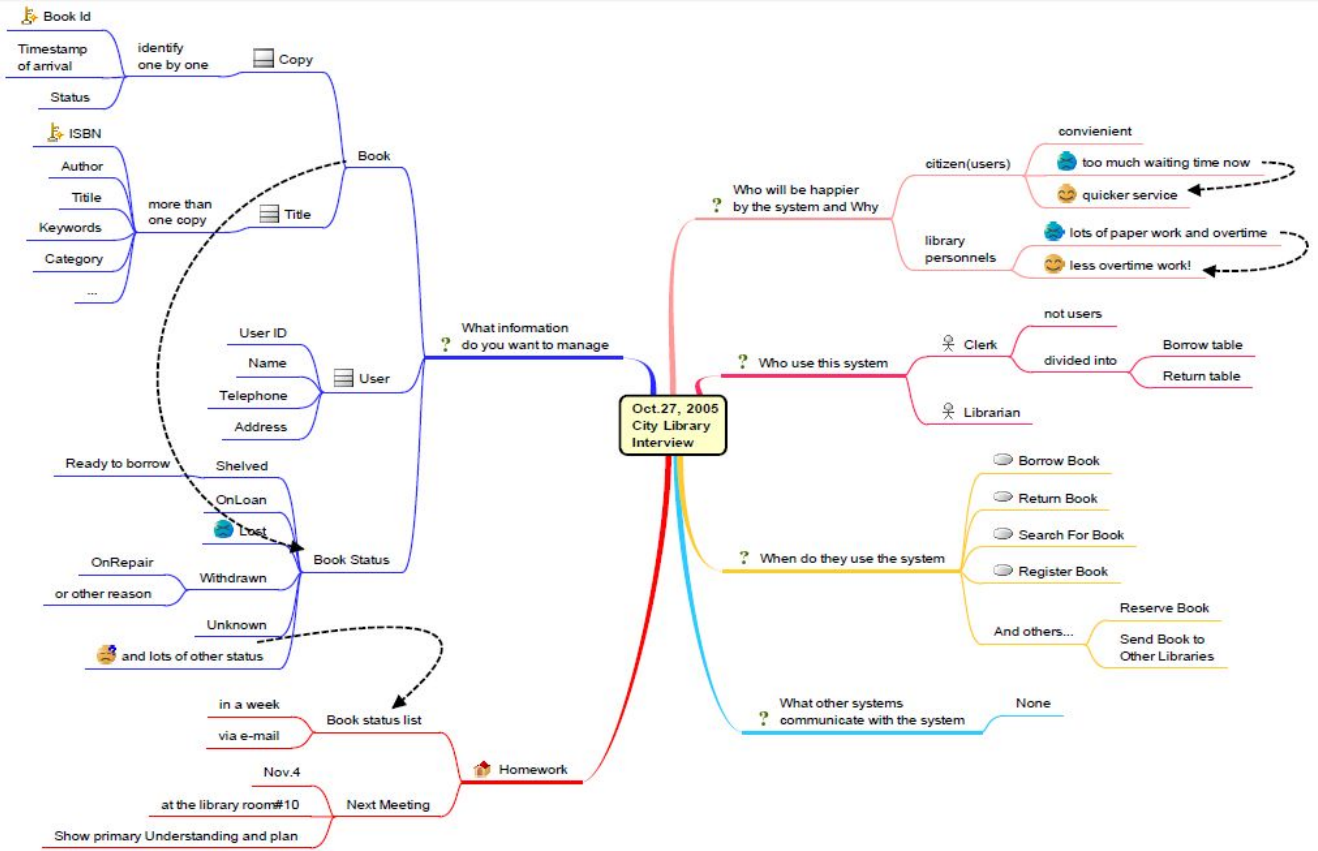
1. Schizophrénie du produit
   - Manque de cohérence dans la définition du produit
     - Communiquer ? Collaborer ? Pas la même chose
2. Complexité de l'UI
   - Incompréhensible pour les non-geeks
   - N'ont pas récolté le feedback des "*vrais*" utilisateurs
3. Complexité de l'UX
   - Trop de possibilité pour une petite équipe
   - Beaucoup de fonctionnalité très techniques (API, …)

Besoins et Attributs

- Besoins
  - Le produit est un moyen de satisfaire ces besoins, pas une fin en soi.
- Attributs du produit
  - Propriétés critiques du produit permettant de satisfaire ces besoins.
  - Explorer les différents attributs possibles et trouver la meilleure alternative.

Comment recueillir les besoins

1. **En amont du projet**
   a. Le brainstorming (à l'aide d'une mind map)
   b. L'analyse de la concurrence (benchmarking)
2. Affiner les besoins
   a. Interviews (Questionnaires, croisement des résultats)
   b. Workshops
3. La description détaillée
   a. L'analyse de l'existant
      i. Forces/faiblesses de l'application
      ii. Quelles fonctionnalités améliorer/remplacer
   b. L'observation du comportement de l'utilisateur « en situation »



Brainstorming à l'aide d'une Mind Map

Comment recueillir les besoins

1. En amont du projet
   a. Le brainstorming (à l'aide d'une mind map)
   b. L'analyse de la concurrence (benchmarking)
2. **Affiner les besoins**
   a. Interviews (Questionnaires, croisement des résultats)
   b. Workshops
3. La description détaillée
   a. L'analyse de l'existant
      i. Forces/faiblesses de l'application
      ii. Quelles fonctionnalités améliorer/remplacer
   b. L'observation du comportement de l'utilisateur « en situation »

	Quel est le problème?	Qui est impacté?	Visualiser la solution
Questions ouvertes "Dites moi..." "Racontez moi..." "Et puis..."	**1**	**4**	**7**
Contrôle Combien? Quand? Où?	**2**	**5**	**8**
Validation "Si je comprends bien..." Si non, revenir au questions ouvertes Si oui, passer à la question suivante	**3**	**6**	**9**

Les neuf cases de Solution Selling

L'objectif est d'arriver à la case 9 en passant par les différentes étapes.

1. **Question Ouverte**
   - Le client commence par raconter des « *histoires* » sur son métier, ses difficultés pour explorer et comprendre.
2. **Questions de contrôle**
   - Permettent de clarifier, d'apporter des précisions.
3. **Validation**
   - L'interviewer doit ensuite valider sa bonne compréhension et synthétiser l'essentiel par la reformulation. Cette étape permet de lever toute ambiguïté.

Les neuf cases de Solution Selling

4. Si la synchronisation s'opère (le client est compris par l'interviewer), ce dernier peut passer à l'étape suivante et tenter de comprendre qui est impacté par le problème et comment il est impacté (4).
5. Le processus de contrôle et de validation est ensuite le même (5 et 6). Si la compréhension n'est pas totale, il doit revenir aux étapes 1 et 2.

Les neuf cases de Solution Selling

7. Lorsque le problème est bien identifié et les impacts mesurés, l'objectif suivant est d'amener le client à se représenter la solution idéale. Sans qu'il soit nécessaire, pour lui, d'élaborer la solution, il peut cependant visualiser un environnement où le problème n'existe plus : que voit-il ? qu'entend-il ?... Le client vit le futur au présent !
    - NB : Si votre client tente de vous amener directement à la case 9 parce qu'il croit connaître la solution, veillez à reprendre obligatoirement les étapes une à une depuis la case 1, ne serait-ce que pour vous approprier son problème, le clarifier et valider ainsi l'adéquation de la solution envisagée.

Comment recueillir les besoins

1. En amont du projet
   a. Le brainstorming (à l'aide d'une mind map)
   b. L'analyse de la concurrence (benchmarking)
2. Affiner les besoins
   a. Interviews (Questionnaires, croisement des résultats)
   b. Workshops
3. **La description détaillée**
   a. L'analyse de l'existant
      i. Forces/faiblesses de l'application
      ii. Quelles fonctionnalités améliorer/remplacer
   b. L'observation du comportement de l'utilisateur « en situation »

Formaliser et exprimer les besoins

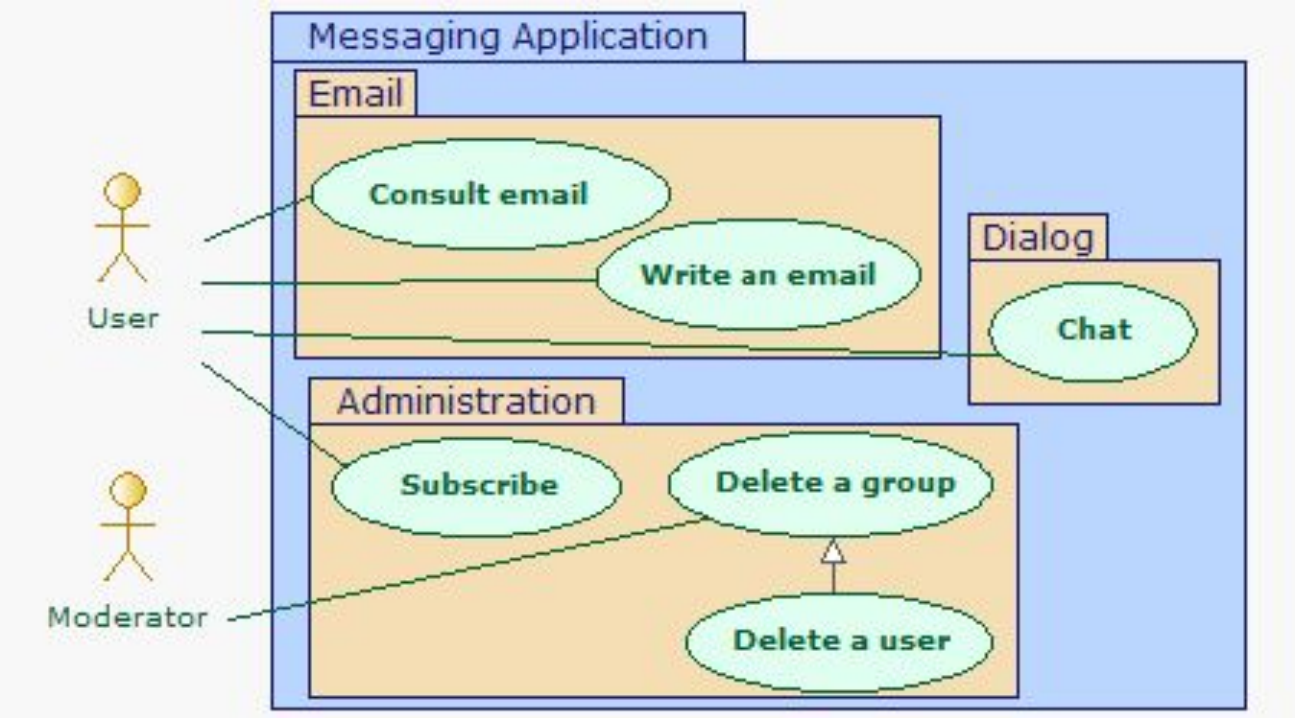
Use Cases (Cas d'utilisations)

- Modélisation UML
- Décrit les échanges entre le système et les acteurs externes (utilisateurs ou autres systèmes) pour un contexte particulier
- Décrit les différentes étapes pour les séquences:
  - Nominales
  - D'exception

User Stories

- En tant que ..., je veux ... afin de ...

Diagramme de cas d'utilisation (“*Use Cases*”)



Cas d'utilisation textuel

USE CASE # 1	*Forme verbale (infinitif)*		*(ex : Passer une commande)*
Résumé			
Acteur principal	X		
Intervenants & Intérêts	*Intervenants*	*Intérêt*	
	X Y Z		
Préconditions			
Garanties de succès	*(Conditions d'arrivée)*		
Déclencheur	*(Evènement)*		
Scénario nominal	Etape	Action	*(Le cas où tout se passe bien)*
	1	L'utilisateur	
	2	X lance	
	3	Le système	
	4	X lance le processus (Cas d'utilisation 1.1)	
	5	Le système.	
Extensions	Etape	Action	*(Variations au scénario et cas d'erreurs)*
	2a	X reçoit ...	
		2a.1 X fait ceci	
		2a.2 Y fait cela	

Use Case vs. User Story

User Story	Use Case
Brève description vue par l'utilisateur	Séquence d'action du système et ses interactions avec les autres acteurs
Format court, stimule la discussion orale	Format riche en information, peu de place à l'oral
Peut être une partie d'un Use Case	Somme d'un scénario
Utilisé pour la spécification et/ou la planification	Uniquement utilisé pour la spécification
Emergence rapide au travers d'ateliers collaboratifs	Long travail d'analyse et de formalisation
Grande visibilité	Visibilité réduite (due au volume)
Difficile à lier les unes aux autres	Liaison et vision globale

Les erreurs courantes

1. Pas de vision
   - Produit limité à un ensemble de features.
2. Vision prophétique
   - "*On sait ce qui est bon pour le client*"
   - Capacité limité de projection.
3. Analyse sans fin des besoins
   - Peur de l'échec.
4. "*Big is beautiful*"

Exemple : King (Candy Crush)

Le succès de Candy Crush s'explique par une bonne analyse des attentes de leurs joueurs.

- Game Design at King
  - https://www.youtube.com/watch?v=uQvkZruWyZk
- Game Designer at King
  - https://www.youtube.com/watch?v=Hl_HRmK66cY

Exercice : la vision du produit

Les questions à se poser :

- Qui va acheter le produit? Qui va utiliser le produit?
- Quels besoins vont être satisfaits par le produit? Qu'est ce que le produit ajoute de plus?
- Quels sont les attributs critiques pour répondre à ces besoins?
- Comment se comporte le produit par rapport à la concurrence?
- Quel est le business model?
- Le produit est-il réalisable?

Exercice : la vision du produit

Senseo vs Nespresso

	Senseo	Nespresso
Acheteur / Utilisateur	1. Grand public (classe moyenne) 2. Jeune ménage (revenus plus restreints) 3. Célibataires 4. Entreprise (employés) 5. Jeunes besoin rapide	1. Amateur Expresso 2. Classe moyenne plus aisée - BCBG (citadin) 3. Entreprise (clients) 4. Personne sensible à l'image de marque

Senseo vs Nespresso

	Senseo	Nespresso
Besoins	Café individuel (Pas de gaspillage; Fraîcheur) Simplicité Rapidité Diversité **Accessibilité** (budget, distribution) Stockage réduit () Syndrome IKEA	Café individuel (Pas de gaspillage; Fraîcheur) Simplicité Rapidité Café de qualité, diversité **Exclusivité** (Luxe (confort), Prestige...) Appartenance à une communauté

Senseo vs Nespresso

	Senseo	Nespresso
Attributs	Utilisation de dosettes Matériaux à faible coûts (plastique, filtre...) Distribution très large (grande surface, multiples fournisseurs de dosettes...) Un seul bouton	Utilisation de dosettes Matériaux de plus haute qualité, design élaboré (chrome)... Utilisation d'une pression élevée (10bar) pour réaliser l'expresso (obtenir le collet de mousse Distribution très régulée (brevets (70 - fin 2012), boutiques, internet...) Communication (George Clooney...) Le club Nespresso (carte)

Senseo vs Nespresso

	Senseo	Nespresso
Business Model	Prix réduit → Grand public → Volume	Ciblage plus étroit → Marge plus élevée Ventes de produits associés (tasses, etc...)

Agile

Scaling Agile

CONSULTING BSTORM

Être Agile dans une grande Organisation

La philosophie Agile dans des grandes organisations

La philosophie Agile et son framework Scrum sont initialement créé pour que des équipes de maximum 10 personnes travaille efficacement ensemble.

Mais comment fait-on lorsque l'on est une grande organisation et que l'on veut adopter une philosophie de travail Agile ?

Structure Top Down

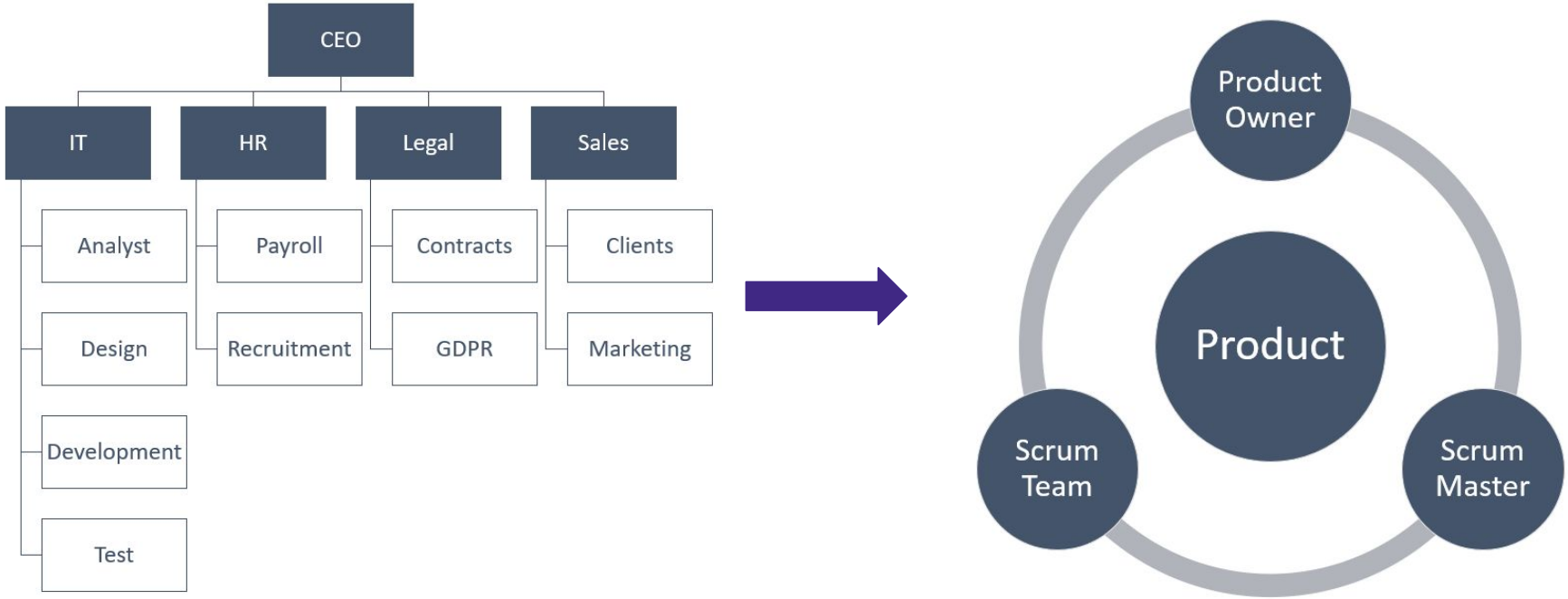
Une structure **Top Down** demande à chaque équipe de se spécialiser sur un sujet spécifique. Les projets sont organisés de manières transversales.

- Associée à une méthodologie de type **Waterfall**.
- Idéal pour la **production** ou pour la création de **produits standardisés**.

Inconvénient

- Difficultés à organiser plusieurs projets complexes en même temps.
- Difficultés de réorganiser un projet en cas de changement.

De Top Down à Scaling Agile



Difficultés de Scrum dans une grande organisation

1. **Garder toujours la même équipe**
   - Dans Scrum il est recommandé qu'une équipe travaille uniquement sur un seul projet et que l'équipe reste la même de projet en projet.
   - Hors dans une grande organisation, une équipe est constitué de plusieurs personnes de départements différents. Souvent on travaille sur plusieurs projets en même temps. Et une fois le projet terminé on dissout l'équipe.

2. **Garder l'autonomie des équipes**
   - Une équipe Agile doit avoir un contrôle sur les choix que l'on fait sur un produit.
   - Dans une grande organisation, le choix stratégique d'un produit sont souvent détenue hors de l'équipe dans les équipes de Management.
   - De plus, on associe souvent un chef de projet qui est responsable des résultats du projet. Il aura alors tendance à vouloir contrôler l'équipe.

Difficultés de Scrum dans une grande organisation

3. Éviter les dépendances

- Dans une grande organisation il est difficile d'avoir des équipes autonomes qui ne dépend pas d'autres équipes ou département.
- Il faut éviter les dépendances et garder le contrôle de chaque partie du projet au sein de l'équipe.

4. Avoir qu'une seule vision du produit

- Un produit dans une grande organisation dépend le plus souvent de plus de 10 personnes. On divise donc la conception d'un produit à plusieurs équipes. Ce qui rend plus difficile d'avoir et de maintenir une vision cohérente du produit.

Avantages d'Agile pour une grande Organisation

Quelles sont les avantages pour une grande organisation de travailler en Agile ?

1. **Time to Market**
   - Pouvoir livrer plus vite un produit ou service.
   - Prendre plus rapidement des parts de marché.
2. **Productivité**
   - Les équipes travaillant en petite équipe, cela permet à une grande organisation de limiter la "lourdeur administrative" d'une structure organisationnelle top-down.
3. **Qualité**
   - L'utilisation des feedbacks réguliers pour améliorer la qualité du produit.

Scaling Agile

Scaling Agile Framework

Indépendamment du choix de la méthodologie, la structure d'une organisation varie en fonction de sa taille.

Une startup ou une PME qui grandit finit par ne plus avoir la même efficacité avec Scrum dû à la taille de l'organisation.

Il existe plusieurs méthodes pour pallier à cela, notamment un autre framework appelé **Scaling Agile** qui permet de garder une philosophie de travail Agile malgré la grande taille.

Scaling Agile

L'objectif de Scaling Agile est de coordonner plusieurs équipes agiles.

- Les équipes agiles restent le coeur de la méthodologie.
  - Pour garantir les avantages des méthodes agiles, on veut laisser un maximum d'autonomie et de responsabilité au niveau des équipes.
- Scaling Agile ajoute différentes couches de coordinations par dessus.
  - Le nombre de couche dépend de la taille du projet et des équipes.
  - Ces "couches de management" doivent adopter une philosophie agile du management, c'est-à-dire agir en tant que "Servant Leader". Aider les équipes à travailler en leur laissant un maximum de responsabilité et minimiser les contraintes et dépendances.
  - Ces couches s'occupent également de la vision du produit.

The Core Values

Les valeurs fondamentales de Scaling Agile sont :

1. L'alignement
    - Garantir que toutes les équipes avancent ensemble dans la même direction. L'alignement est réalisé lorsque chaque membre de chaque équipe, comprend la stratégie et le rôle qu'il joue dans sa réalisation.
2.

L'équipe Agile

L'équipe Agile

Les équipes Agile restent au centre d'une grande organisation Agile.

- **Taille** : environ 8 personnes.
- **Méthodologie** : Scrum ou une autres frameworks d'Agile (ex: Kanban).
- **Backlog** : chaque équipe à son propre backlog.
- **Itération** : 1 à 4 semaines (selon la méthodologie).
- **Rôles** : Product Owner, Scrum Master, Scrum Team.

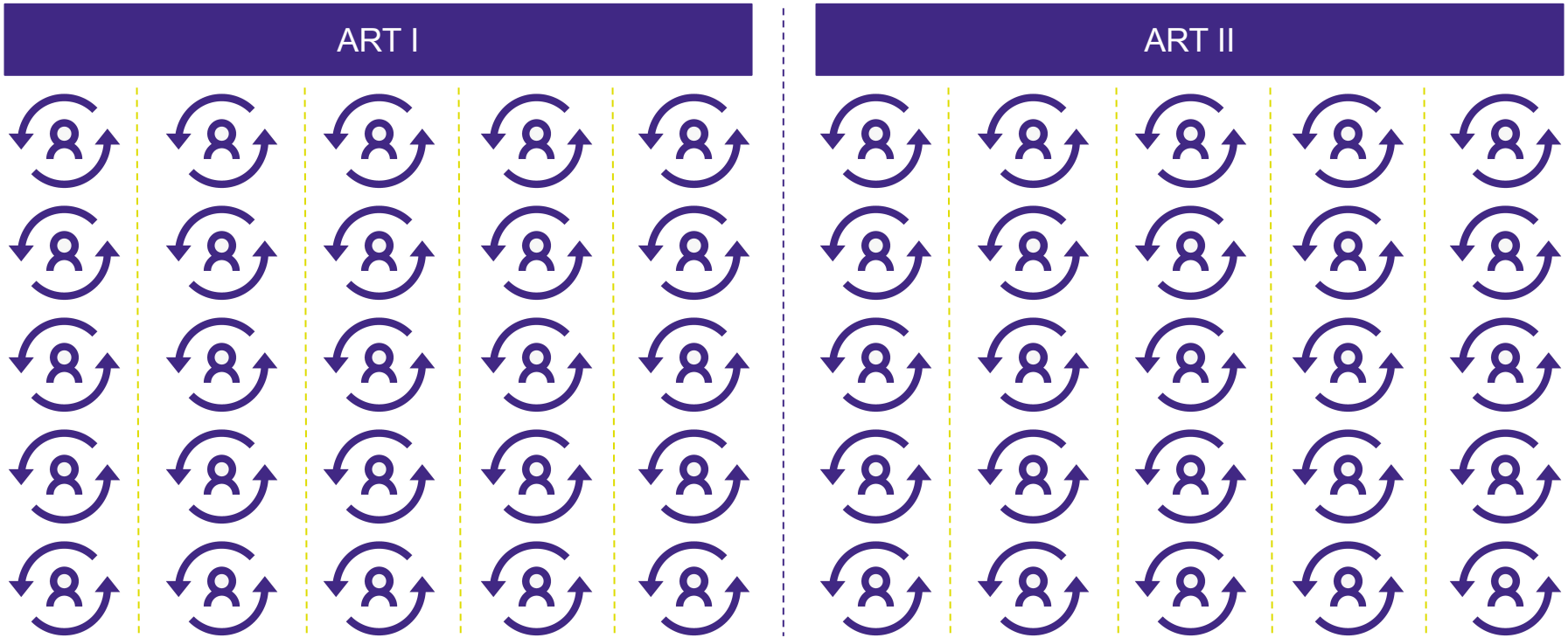
Program Level

Program Level

On ajoute une niveau au-dessus de toutes les équipes du projet pour assurer la coordination entre les celles-ci.

- **Taille** : environ 8 équipes.
- **Méthodologie** : Kanban.
- **Backlog** : reprend les **features** qui sera divisé en différents User Stories dans les backlogs des différentes équipe.
- **Itération** : **Agile Release Train** (ART) de 8 à 12 semaines. Généralement la dernière semaine est consacrés à l'intégration, les tests et optimisations.
- **Rôles** : Product Management, System Architect, Realease Train Engineer et Business Owners.

Agile Release Train (ART)



Attention !

Les niveaux d'abstraction supérieur servent à organiser le travail et coordonner les équipes. Mais…

1. Le travail sur le produit ou service n'est réalisé uniquement à niveau de l'équipe Agile.
2. Les différentes responsabilités sont maintenus dans les niveaux les plus bas, afin de garantir aux maximum d'autonomie aux équipes.

Solution Level

Solution Level

Généralement lorsque le projet nécessite plus de 8 équipes, on va ajouter une niveau d'abstraction supplémentaire, appelé **Solution**.

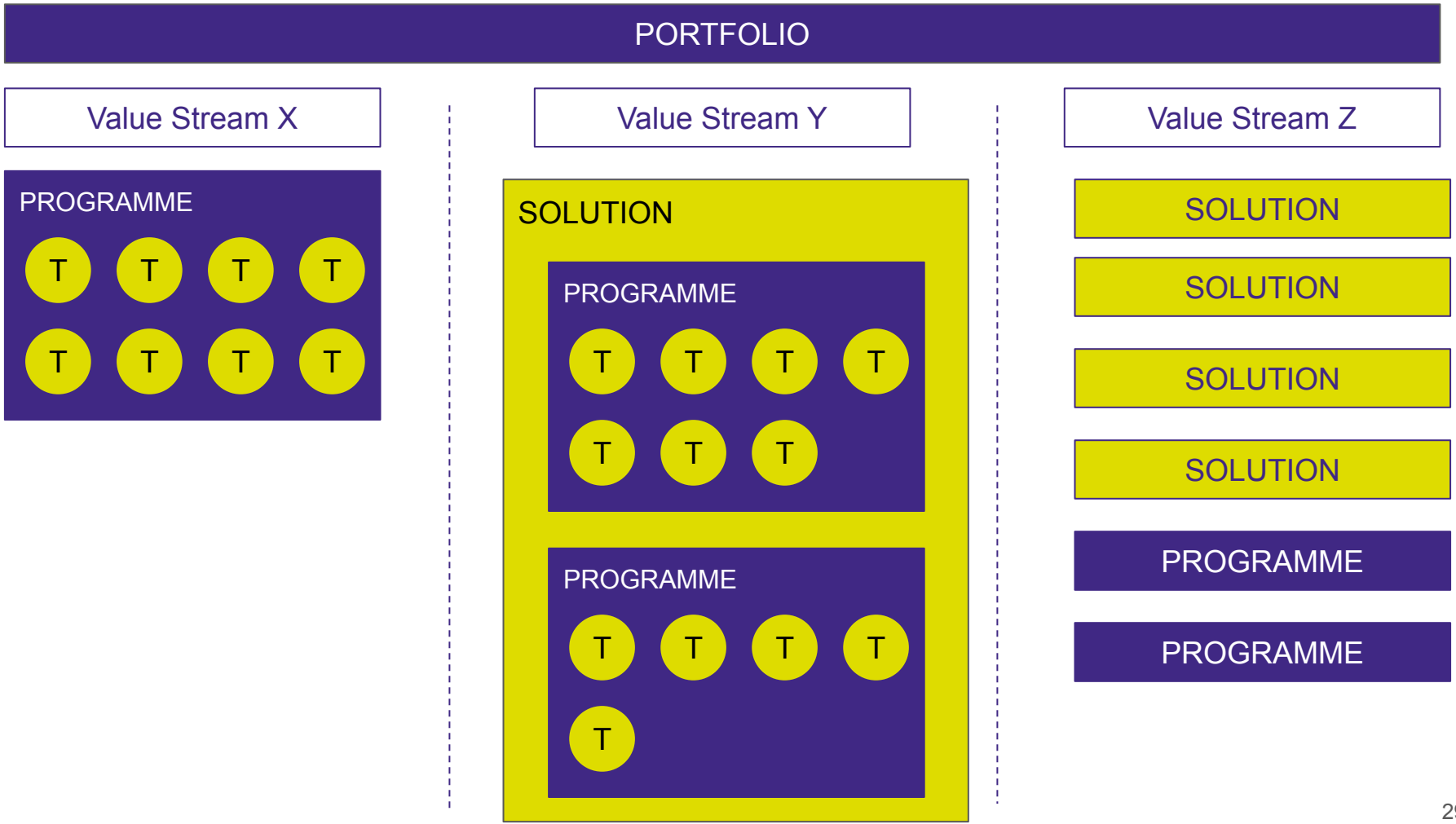
- **Méthodologie** : Kanban.
- **Backlog** : reprend les **capacités** qui sera divisé en différents features dans les backlogs des différentes programmes.
- **Rôles** : Solution Management, Solution Architect, Solution Train Engineer et clients.

Portfolio

Portfolio

Lorsqu'une grande organisation travaille exclusivement en Agile, on ajoute un dernier niveau appelé **Portfolio**. Où l'on regroupe tous les différents projets de l'entreprise appelé **Value Stream** qui sont divisés selon les besoins en solution, programme où équipe.

- **Backlog** : reprend les **Epics** qui sera divisé en différents capacité dans les backlogs des différentes solutions.
- **Rôles** : Lean Portfolio Management, Enterprise Architect, Epics Owners.


PORTFOLIO
Value Stream X
Value Stream Y
Value Stream Z
PROGRAMME
T T T T
T T T T
SOLUTION
PROGRAMME
T T T T
T T T
PROGRAMME
T T T T
T
SOLUTION
SOLUTION
SOLUTION
SOLUTION
PROGRAMME
PROGRAMME

Success Stories

Lego

En 2015, Lego a initié son changement de ses 20 équipes produits. Ce changement s'est fait en plusieurs phases.

1. **Convertir les différentes équipes en agile**
    - Changement progressif : d'abord 5 équipes pour tester la méthodologie, ensuite les 15 autres ont suivi.
2. **Ajouter le niveau Programme**
    - Même si les équipes avaient correctement intégrés individuellement la méthodologie Agile, elle n'arrivait pas à coopérer ensemble.
    - Meeting tous les 8 semaines pendant 1 jour et demi pour coordonner le travail des différentes équipes.

Lego

3. **Ajouter le niveau Portfolio**
    - Où les grandes orientations long termes des différents projets sont prises par le Top Management.

Lego

Résultats

1. Moins de documentations et de réunions.
2. Meilleures estimations du travail à réaliser.
3. Augmentation de la motivation et productivité des employés.

Cisco

Cisco auparavant suivait le méthodologie Waterfall. Il y avait des équipes distinctes chargées de la conception, développement, test et déploiement.

Problèmes

1. Bugs trop nombreux.
2. Retard trop important.
3. Trop d'heures supplémentaires.

Cisco

Solution

- En 2015, création de plusieurs Agile Release Trains qui suivait chacune des méthodologies Agile différentes.

Résultats

1. Réduction de 40% du nombres de bugs.
2. Livraison toujours dans les temps.
3. Budget sous contrôle.

British Telecom

Jusqu'en 2004, British Telecom connaissait de nombreux problèmes avec leur méthodologie (Waterfall).

1. Les exigences (besoins) étaient exprimées par trop de personnes.
2. Aucune priorisation des tâches (presque toutes avaient la priorité élevée).
3. Trop de personnes impliqués dans un projets (long processus d'approbation).
4. Les délais étaient impossibles à respecter.
5. Beaucoup de pression sur les développeurs.
6. Pas assez de temps pour le contrôle qualité.
7. Beaucoup de projets ont été des échecs.

British Telecom

Solution

- Transformation de deux ans vers un modèle Agile.

Résultats

1. Les objectifs sont définis au niveau du Portfolio pendant une réunion de 3 jours par le Top Management et les actionnaires.
2. Priorisation en fonction du besoin du clients.
3. Le cycle de livraison est passé de 12 mois à 90 jours.
4. Augmentation du moral et de la motivation des équipes.

Conclusion

Adaptés pour tous les projets ?

Scrum n'est pas adapté pour toutes les équipes, ni pour tous les projets. Convient pour les projets :

1. Complexes.
2. Avec des deadlines courtes.
3. Avec un risque élevé (inconnue / innovation).

Ne convient pas pour les projets

1. Où les priorités changent quotidiennement (moins d'une semaine).
2. Quand les équipes changent trop régulièrement.
3. Où la transparence et l'équité n'est pas souhaité.

5 valeurs d'une équipe Scrum (FORCE)

La méthode Scrum repose sur les 5 valeurs suivantes :

1. **Focus**
   - Se focaliser sur très peu de choses à la fois.
2. **Ouverture** ("*Openness*")
   - Ouverture et transparence envers le reste de l'équipe, de l'entreprise et des clients.
3. **Respect**
   - Respect et confiance envers le travail des autres (ex: PO envers la Scrum Team).
4. **Courage**
   - Créer un environnement où l'on a pas peur d'essayer. La peur de l'échec est diminué puisqu'une erreur peut être corrigée à la prochaine itération.
5. **Engagement** ("*Commitment*")
   - Respectez le contenu d'une mission tel qu'il a été convenu.

"If your organization doesn't like truth and honesty…
it probably won't like agile"

Henrik Kniberg
Agile Coach

CONSULTING BSTORM

Merci pour votre attention

Références

- Agile Product Management with Scrum, Creating Products that Customers Love, Roman Pichler, Addison-Wesley, 2010.
- Gestion de projet, Vers les méthodes Agiles, Véronique Messager Rota, Eyrolles, 2007.
- The Scrum Guide, The definitive Guide to Scrum: The Rules of the Game, Ken Schwaber, Jeff Sutherland, Scrum.org, 2011.
- Illustrations - Emmanuel Chenu:
  - http://emmanuelchenu.blogspot.com/