

An isometric illustration depicting the mobile application development process. It features several computer monitors displaying various icons like a lightbulb, a play button, and code symbols. A robotic arm is shown interacting with a large red gear. Two people are seated at a desk with laptops, and another person is standing and pointing at a screen. The scene is set against a white background with a blue footer bar.

Conception d'applications mobiles
420-295-AH
Cours 11

Étape 3: Méthode Agile

Cours	Contenu
11	Méthode Agile
12	Assurance qualité et restructuration

Retour en 1968...

Avant cette année, des « experts » étaient engagé pour une somme hallucinante dans l'espoir d'avoir un logiciel fonctionnel en deux ans. Sauf que...

- Majorité des projets échouent ou déçoivent,
- Bogues, bogues, bogues, bogues ...
- Dépassements d'échéancier systématiques.

Solution de 1968: Produire de la **documentation** afin de suivre l'avancement et donc faciliter la gestion.

L'idée est que si un projet échoue, on peut au moins apprendre de la documentation produite.

Finalement?

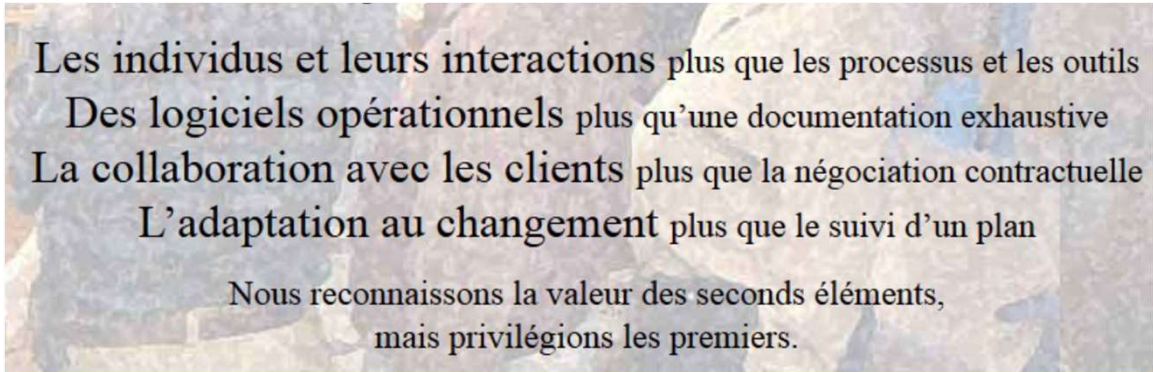
En 2001, même avec cette documentation:

- Majorité des projets échouent ou déçoivent
- Bogues, bogues, bogues, bogues ...
- Dépassements d'échéancier systématiques.

La documentation est si exhaustive qu'elle est impossible à gérer.

Ex. réel : 10k pages de documentation pour un projet de 2 ans ...

Révolte des ingénieurs: écriture du manifeste Agile. Cela ne veut pas dire d'abandonner la documentation, mais elle ne devrait pas être l'objectif du développement logiciel.



Les individus et leurs interactions plus que les processus et les outils
Des logiciels opérationnels plus qu'une documentation exhaustive
La collaboration avec les clients plus que la négociation contractuelle
L'adaptation au changement plus que le suivi d'un plan

Nous reconnaissons la valeur des seconds éléments,
mais privilégions les premiers.

Caractéristiques de la méthode Agile

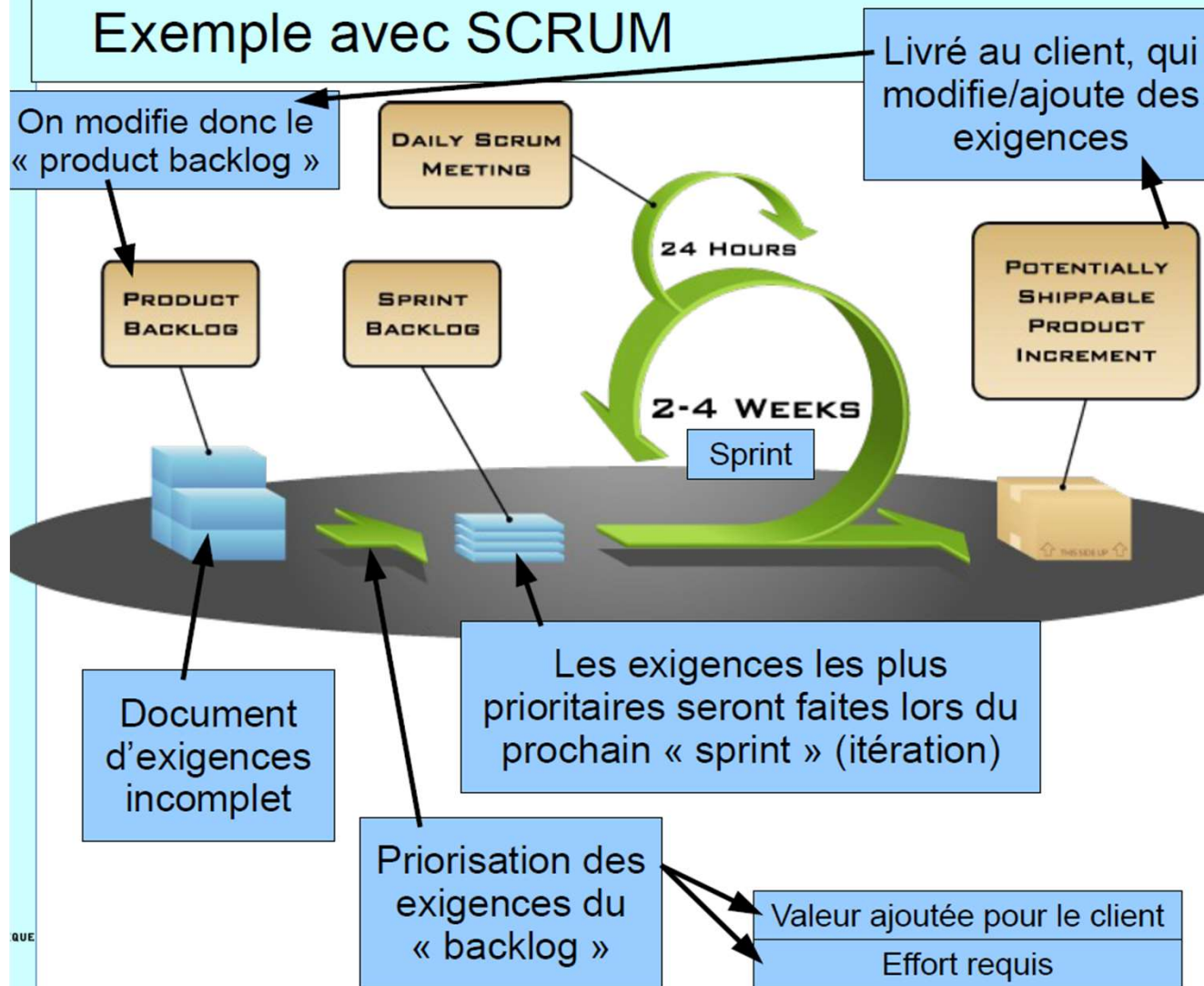
Itérations rapides (<1 jour à 6 semaines) :

- Chaque itération produit un logiciel fonctionnel.
- Chaque itération ajoute quelque chose au logiciel de l'itération précédente.

Ouverture aux changements : les exigences changent constamment durant le projet.

Remplacement de la documentation papier par des échanges face-à-face.

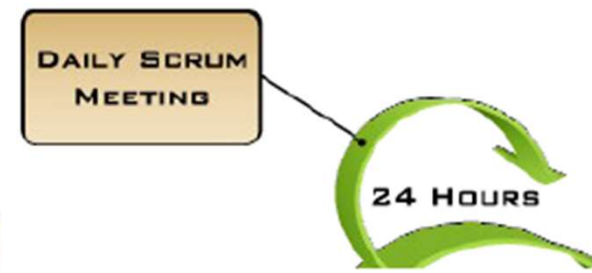
Exemple avec SCRUM



Exemple avec SCRUM

Qu'est-ce que le « Daily SCRUM Meeting » ?

- Provient du « Scrum » au Rugby, littéralement la mêlée quotidienne.
- Réunion quotidienne, la plus courte possible (on vise 5-15 minutes).
 - Souvent faite debout (« stand-up meeting ») afin d'éviter qu'elle s'éternise.
- Tour de l'équipe où chaque personne répond aux trois questions suivantes :
 - Qu'est-ce que j'ai fait hier.
 - Qu'est-ce que je prévois faire aujourd'hui.
 - Qu'est-ce qui bloque mon avancement.



Exemple avec SCRUM

Rôles dans un projet SCRUM :

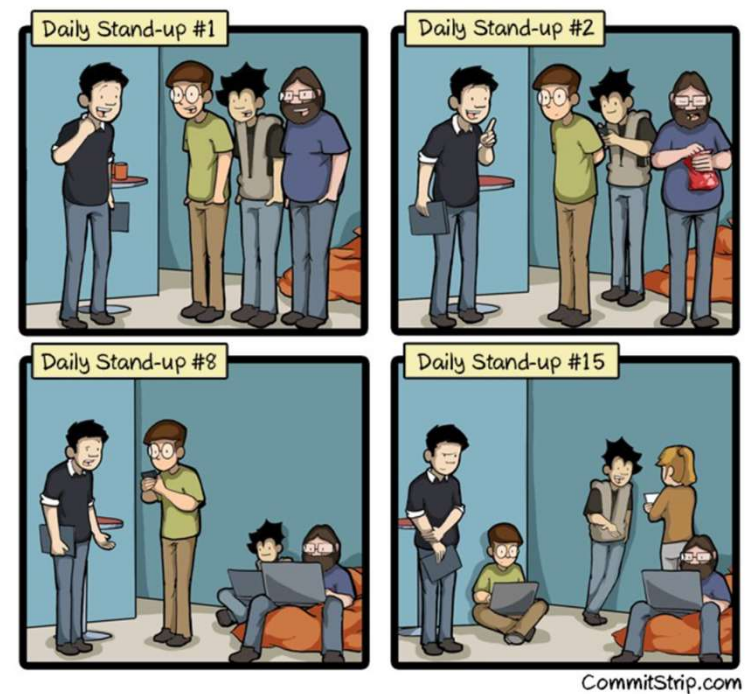
- **Product Owner** : représentant des clients.
 - Itérations rapides → il faut des rétroactions fréquentes afin de mettre à jour le « product backlog ».
 - Idéalement, on a les clients, mais sinon on a des personnes qui connaissent très bien les besoins des clients.
- **Scrum Master** : facilitateur/facilitatrice du processus SCRUM.
 - S'assure que les étapes sont respectées.
 - Pas de « boss » dans une équipe SCRUM → nécessite une équipe disciplinée.
- **Développeurs** :
 - Pas de gestionnaires officiels → les développeurs doivent être capable d'estimer correctement leurs tâches.

Dans la pratique, il y a souvent un gestionnaire qui fait le lien entre l'équipe et la direction, mais qui se base sur les estimations de l'équipe.

Exemple avec SCRUM

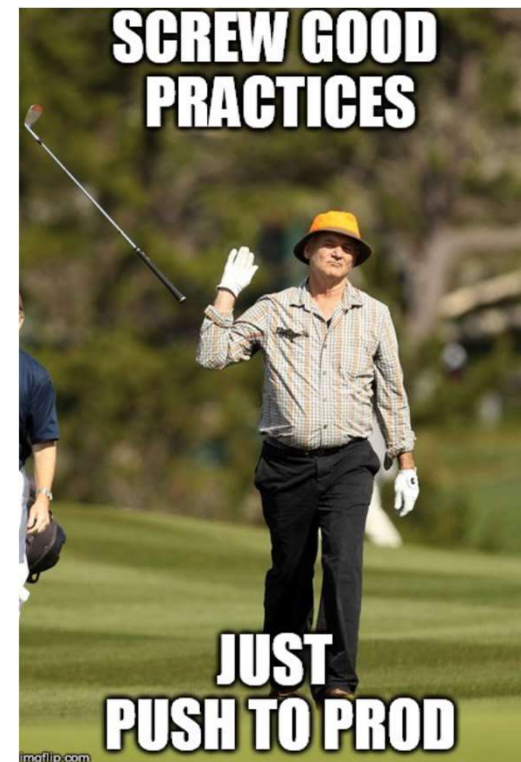
Le SCRUM ne donne aucun détail sur comment l'itération (le « sprint ») doit être exécuté.

- Chaque organisation implémente le SCRUM selon ses besoins.
 - Ex.: Ajouts d'activités d'inspection de code, utilisation du TDD (tests unitaires écrits avant le code), etc.
- Il n'y a pas d'interactions avec les clients durant l'itération, seulement à la fin d'une itération.
 - Par opposition : eXtreme Programming (XP) demande à ce qu'un client soit assis à côté des développeurs durant l'ensemble du projet.



Pratiques Agile populaires

- Afin d'éliminer le plus de documentation possible, des processus Agiles ont introduit certaines pratiques particulières.
- Certaines pratiques sont uniques à l'Agile, mais d'autres ont fini par être appliquées à la plupart des projets modernes.



Pratiques Agile populaires

- **Intégration continue** (« continuous integration ») :
 - Pratique reprise par la majorité des organisations.
 - À chaque « git commit » ou « git push », un outil fait l'intégration automatique du code et exécute certains scripts.
 - Ex.: exécution des tests unitaires, évaluation automatique du code selon le guide de programmation.
 - Permet de savoir tout de suite si les modifications faites fonctionnent et sont de qualité acceptable.
- Outils populaires : Jenkins, Buildbot, GitLab, Microsoft Team Foundation Server
- Beaucoup d'organisations montent leur propre système basé sur des scripts Make, des hooks Git, etc.

Pratiques Agile populaires

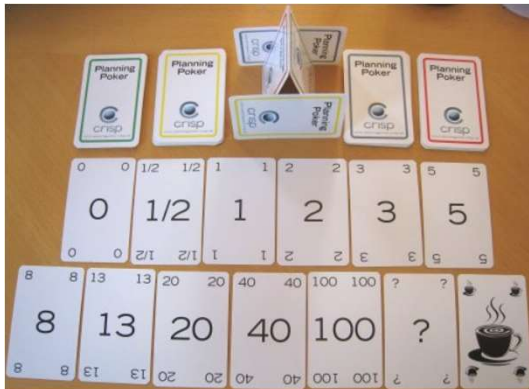
- **Récit (ou scénario) d'utilisateur** (« user stories ») :
 - Structure **d'exigence** orientée vers l'utilisateur/utilisatrice :
 - En tant que <rôle>, je veux faire <fonctionnalité>, afin de <objectif>.
 - Facile de créer des tests d'acceptation basé sur un récit d'utilisateur.
 - Utile en fin d'itération afin d'obtenir les rétroactions des clients.

Pratiques Agile populaires: planification

- Planification des tâches/itérations/projet :
 - Comment les développeurs peuvent estimer l'effort requis ?
 - Estimation basée sur des mesures de « vélocité », soit la quantité de travail faite dans une itération typique.
 - Approche 1 (popularisée par l'Agile) :
 - Attributions de « points » aux « user stories » selon leur niveau de difficulté → approche du Planning Poker.
 - Historique du nombre de « points » implémentés dans les itérations précédentes.
 - Approche 2 (traditionnelle) :
 - Estimation du temps requis pour faire une « user story » basée sur les fois précédentes où un travail similaire a été fait.
 - Historique du temps de travail réel fait lors des itérations précédentes.
- Témoignages d'équipes Agiles : il faut 3-6 mois pour que l'équipe commence à apprendre à faire des estimations acceptables.

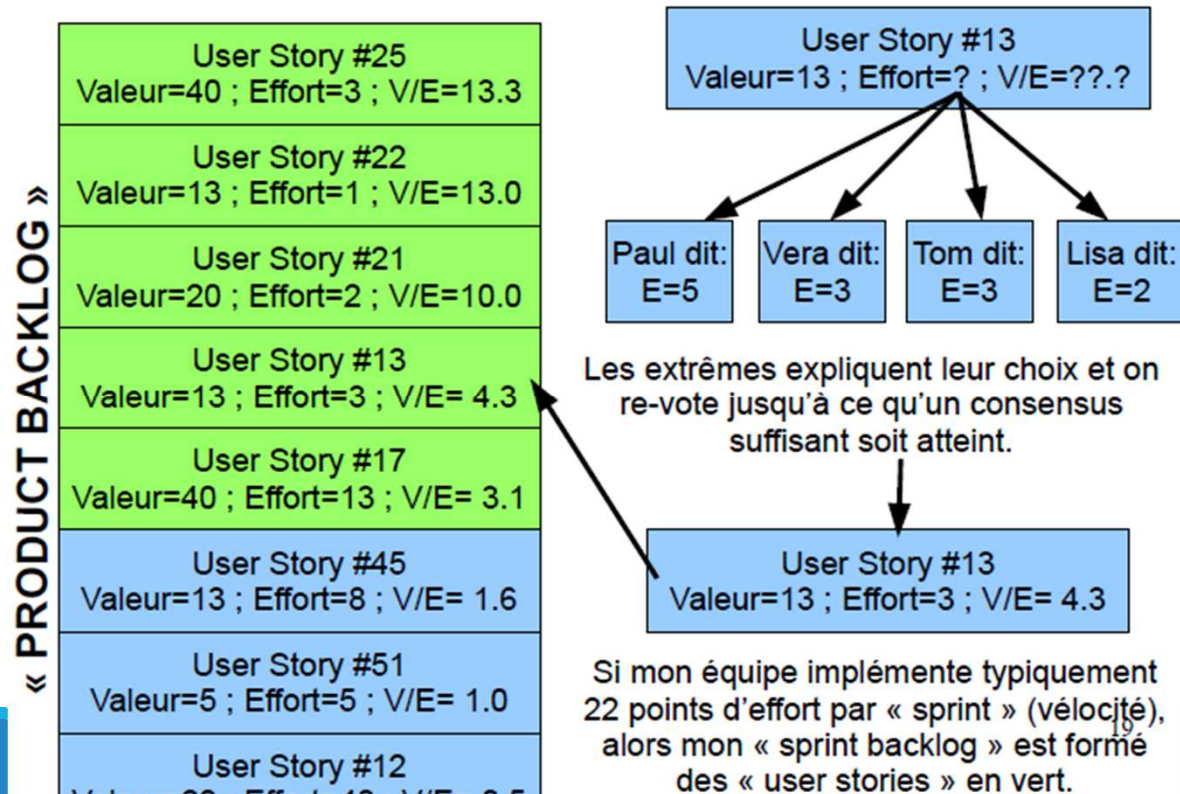
Pratiques Agile populaires: planification

- Approche du Planning Poker
 - Série de cartes correspondant +/- à la suite de Fibonacci.
 - Tout le monde tourne ses cartes en même temps.
 - Les gens aux extrêmes (haut et bas) justifient leur estimation.
 - On recommence jusqu'à ce qu'un consensus suffisant soit atteint.
 - Utilisé pour estimer la valeur pour les clients d'une user story et de l'effort pour l'implémenter.



Pratiques Agile populaires: planification

Exemple avec un « product backlog » déjà presque tout priorisé, utilisant les points et le Planning Poker.



Pratiques Agile populaires: Kanban

Du japonais pour « enseigne » ou « panneau d'affichage ».

- La planification complète du travail est visible pour toute l'équipe.
- Le développeur prend une tâche disponible sur le tableau et la remet une fois complétée.



Généralement en parallèle d'une gestion plus élaborée faite par le ou la gestionnaire.

Supporté par une gamme d'outils logiciels (ex: Trello, Slack, Jira...)

Pratiques Agile populaires: planification

- Que se passe-t-il si on se trompe dans nos estimations ?
 - Pas le temps de finir, on finit avant les temps, bogues/problèmes restants non-corrigés ...
- On révisé la planification à chaque itération :
 - Les bogues/problèmes restants sont enregistrés comme exigences dans le « product backlog ».
 - Le travail qui n'a pas été terminé → dans le « product backlog ».
 - Tout ça sera donc corrigé dans un « sprint » futur.
- On peut toujours ajouter un « sprint » de plus en fin de projet ...
 - Faiblesse de l'Agile : difficile de déterminer quand le travail est terminé.

Pratiques Agile populaires

- **TDD** : « Test-Driven Development », développement dirigé par les tests.
 - Écriture des tests unitaires avant d'écrire le code.
- Importance d'un code « simple » et « élégant » :
 - Écriture rapide de code fonctionnel suivi d'activités de restructuration (« refactoring »)
 - En pratique, le temps manque parfois pour nettoyer le code ...
- **Espace de travail ouvert** (« open floor plan ») plutôt que les cubicules traditionnels.
 - Objectif de favoriser les échanges face-à-face informels.
 - Controversé → bruyant, peut être difficile de travailler.
- **Pair Programming** (programmation en paire, ou en binôme) :
 - Deux personnes travaillant sur un ordinateur, avec une souris et un clavier.

Pratiques Agile populaires: programmation en paire

- Utile pour :
 - Travailler sur une partie de code difficile, complexe ou critique.
 - Travailler sur une partie de code qui bénéficierait de plusieurs expertises.
 - Ex.: Développeur + Testeur.
 - Ex.: Développeur + Expert du domaine.
 - Passer l'expérience entre quelqu'un qui connaît bien le code vers quelqu'un qui le voit pour la première fois.
 - Résoudre des problèmes de « code ownership ».
- Lorsque bien fait :
 - Peut produire du code de meilleure qualité → évite de coûteuses corrections plus tard.

Pratiques Agile populaires: programmation en paire

- « Code ownership » : À qui appartient le code ?
 - Ex.: Coéquipier qui est frustré à chaque fois que quelqu'un critique ou modifie « son » code.
 - La dépendance à une personne pour un module de code est un risque pour l'organisation → Que se passe-t-il si cette personne part ?
- « Collective code ownership » : Idée que le code n'appartient pas à la personne qui l'a écrit, mais à toute l'équipe.
 - Implémenté en Agile en forçant les développeurs à toucher au code des autres.
 - Ex.: « J'ai besoin de modifier le code que tu as écrit. On se fait une séance en paire et tu me montres ce que tu as fait. »
 - Avoir plusieurs yeux capables de lire le code est un avantage :
 - Plus de personnes capable de corriger les bogues.
 - Plus de critiques sur la manière dont le code est écrit.

Pratiques Agile populaires

- Super ! On est Agiles ! Mais est-ce que ça donne vraiment de meilleurs résultats ?
 - Pas spécialement.
 - C'est une approche de développement différente qui est utile dans beaucoup de contextes.
 - Ex.: Où il y a beaucoup d'interfaces, nécessité de beaucoup de rétroactions avec clients.
 - Moins dans d'autres.
 - Ex.: Projet très critique, projet difficile à fractionner en itérations ...