Collège Lionel-Groulx

Département d'informatique

Cours : 420-KBE-LG, Projet dirigé

Hiver 2020

Projet dirigé

Scrum, pour la gestion de projet

Saliha Yacoub

CLG

Table des matières

Chapitre1, Introduction : le processus de développement logiciel 4
Définitions 4
Qu'est-ce qu'un processus de développement ? 4
Qu'est-ce qu'un artéfact ? 4
Qu'est-ce qu'un système logiciel ? 4
Projet versus produit 4
Activités ou étapes de développement de produit logiciel 4
Activité 1. Étude préliminaire ou étude des besoins 5
Activité 2. Analyse fonctionnelle ou (Le modèle d'analyse) 5
Activité 3. Conception : Le modèle de conception 5
Activité 4. Réalisation technique du nouveau système (implantation ou implémentation) .. 7
Activité 5. Mise en place, exploitation et évaluation 7
Méthodes de développement de produit logiciel 7
Les méthodes en cascade (Waterfalls) 7
Le processus Unifié (PU), l'ancêtre des approches agiles, ... 8
Méthodes agiles : (détaillée plus loin) 9
Chapitre 2, Recueillir efficacement les besoins 10
Partager une vision 11
Faire émerger les besoins : 11
Définition : Un besoin : 11
Émergence des besoins 12
Recueillir les besoins : 12
Le Brainstorming : 12
L'interview : 12
Le Questionnaire : 13
L'observation des utilisateurs en action ou en situation 13
L'analyse de l'existant : 13
Formaliser les besoins : 13
La norme IEEE830-1998 : 14
Les use-case ou les cas d'utilisation d'UML (Unified Modeling Langage) 15

Exemple. Caisse enregistreuse 18
Les user-stories : 20
Propriétés INVEST d'une User story 21
Les user-stories : Détails et tests d'acceptation 23
Conclusion : comparaison user-stories et use-case 25
Chapitre 3, Introduction aux méthodes agiles : 27
Manifeste agile 27
Valeurs agiles 27
Les principes : 27
Présentation de quelques méthodes agiles : 28
XP pour eXtreme Programming: 28
Kanban 29
Chapitre 4, La méthode agile SCRUM : Présentation 31
Présentation : 31
Définitions : 31
En bref, c'est quoi Scrum? 31
Cycle de développement SCRUM : 31
Aspect temporel : 31
Activités et cycles de développement 32
Le résultat d'un sprint 33
Le résultat d'une release. 33
Rôles et responsabilités : Le Product Owner et le Scrum Master 33
Le Product Owner : (PO) 33
Le SCRUM Master : 34
Démarrer un projet SCRUM 34
Le backlog du produit : 34
SCRUM est un développement léger et agile (phases de développement) 35
Chapitre 5, SCRUM en action 36
Construire une bonne vision 36
Identifier les éléments d'une vision 36
Identifier les utilisateurs (rôles) : 37

Les features , définition 38
Décomposer en user-stories 38
Comment passer de la vision aux stories ? 38
Construire le backlog du produit : 38
Prioriser les stories : La méthode Moscow 38
La méthode priority Poker (pour les feature) 38
Exemple de backlog initial, pour e jeu Médiéval 39
Estimer les stories … comment ? 39
Estimer les stories : La planning Poker 39
Estimation en effort ou en temps : 41
Conclusion pour le backlog 42
Exemple de backlog détaillé. 42
Planifier la release 43
Étape 1 : définir le critère de fin de la release : 44
Étape 2 : estimer les stories du backlog : La planning Pocker (voir page 37) 44
Étape 3 : définir la durée des sprints : 44
Étape 4 : estimer la capacité de l'équipe 45
Étape 5 : produire un plan de release 45
Le sprint zéro 46
Références bibliographiques: 47

Chapitre1, Introduction : le processus de développement logiciel

Définitions

Qu'est-ce qu'un processus de développement ?

Un processus défini qui fait quoi, à quel moment et de quelle façon pour atteindre un certain objectif. Dans le domaine du développement logiciel le but consiste à élaborer un produit fini ou à en améliorer un existant.

Un processus digne de ce nom doit fournir des directives garantissant le développement efficace de logiciels de qualités.

Qu'est-ce qu'un artéfact ?

Un artefact désigne toute sorte d'information créé, modifiée ou produite ou utilisée par les intervenants dans le processus de développement logiciel. Nous nous intéressons uniquement aux artefacts d'ingénierie comme : les diagramme UML, les modèles de bases de données, les plannings de projet, le backlog du produit, les user stories etc….

Qu'est-ce qu'un système logiciel ?

Un système logiciel n'est pas uniquement le code machine (le programme exécutable).

Un système logiciel se compose de TOUS les artefacts nécessaires à sa représentation sous forme lisible aussi bien pour les machines (code source) que pour les concepteurs (modèles de classes, modèle des composants, modèle de base de données) , les architectes, les analystes, les chefs de projet(les user-stories, les cas d'utilisations, le planning du projet …), le client, le service marketing..

Projet versus produit

Un projet de développement débouche sur un produit logiciel ou sur une nouvelle version de produit.

Activités ou étapes de développement de produit logiciel

De ce qui suit, il n'est nullement question de **METHODES** de développement logiciel, il s'agit tout simplement de présenter les activités qui font partie du processus de développement de logiciel. La façon dont les activités sont exécutées détermine la méthode de développement du système logiciel. Par exemple si les activités sont exécutées les unes à la suite des autres, on parlera de la méthode en cascade. Ces activités sont alors des étapes.

Voici les activités de développement d'un produit logiciel

Activité 1. Étude préliminaire ou étude des besoins

L'étude des besoin ou l'étude préliminaire a pour objectifs :

- De clarifier la demande du client
- D'évaluer la faisabilité du projet
- De fournir à la direction de l'organisation ou au comité directeur les données pertinentes pour prendre une décision au sujet de l'opportunité, de la faisabilité et de la rentabilité d'un projet de développement de logiciel.

Cette activité doit être effectuée relativement rapidement et ne pas engager trop de frais. Elle comporte les tâches suivantes :

1. Planification de l'étude préliminaire
2. Clarification de la demande
3. Évaluation de la faisabilité
4. Préparation et présentation du rapport d'étude préliminaire

Activité 2. Analyse fonctionnelle ou (Le modèle d'analyse)

Cette étape vient après qu'une décision positive ait été prise à l'étape précédente. L'analyse répond à la question **QUOI ?** en d'autres mots, elle répond à la question « Qu'est-ce que le produit logiciel doit faire ?» on ne s'intéresse pas à la question *Comment* ?

À cette étape l'équipe de projet prend connaissance du système en place à savoir : Les personnes qui interviennent dans le système et leur rôle dans le système et les tâches accomplies par les utilisateurs ainsi que les problèmes rencontrés

Les principaux objectifs de ces activités sont :

- Évaluer la performance du processus actuel
- Comprendre les problèmes du système à l'étude afin de déterminer les véritables causes de ces problèmes
- Pointer les exigences et les contraintes imposées au système.
- Formaliser les besoins sous formes de cas d'utilisation ou d'user-stories.
- Prioriser et estimer les besoins.
- Proposer un plan de réalisation ou un échéancier

Activité 3. Conception : Le modèle de conception

Cette activité répond à la question **COMMENT ?** Elle a pour objectifs de proposer un nouveau produit logiciel qui saura atteindre les objectifs établis au cours de l'analyse.

Elle consiste à déterminer toutes les composantes d'un produit logiciel qui permettrait d'éliminer les problèmes du système actuel et d'atteindre les objectifs établis lors du diagnostic

La conception consiste à décrire le **COMMENT?** Sans aller jusqu'à la réalisation complète du système. Le modèle de conception décrit les objets, leurs relations, leurs interactions réciproques, l'allocation et la répartition sur les ressources physiques de traitement ainsi que les activités concurrentes, leur ordonnancement et leur synchronisation dans le système.

La conception va refléter plus la manière dont l'application (logiciel, système) va être réalisée, elle met en évidence la réflexion du concepteur (informaticien) par rapport à la réalisation. Dans la phase de conception, on ne peut pas faire abstraction des différents logiciels et matériels qui seront utilisés pour la réalisation. Durant cette activité, le concepteur fixe son choix sur la technologie employée et décide de la nature des ensembles entrants dans la composition du produit ou du système final

Exemple il faudra déterminer si le SGBD qui sera utilisé est Oracle, MySQL ou autre car le modèle de données va dépendre de votre SGBD. Il faudra aussi déterminer le langage de programmation, une implémentation en PHP n'a rien à voir avec une implémentation en JAVA ou en C#. Et surtout déterminer la plateforme de développement.

Objectifs de l'activité de conception :

- Acquérir une compréhension approfondie des questions concernant les exigences non fonctionnelles et les contraintes liées aux langages de programmation, à la réutilisation des composants, aux systèmes d'exploitation, etc.
- Constitue un point d'entrée pour l'implémentation, au sens ou l'implémentation est un raffinement direct de la conception.
- Déterminer les principales interfaces des sous-systèmes.

  Il existe deux niveaux de conception :

  1. Générale ou d'architecture (dite également de haut niveau): Elle définit les mécanismes communs, et les choix importants du système. Le modèle de base de données et le diagramme de déploiement peuvent être utilisés, un diagramme de navigation dans un site Web.

  2. Détaillée, elle s'intéresse au niveau élémentaire ou une partie du logiciel à développer. Exemple, un formulaire de page Web, la réalisation d'un sprint…ou d'un sous-système ou d'un module….

Par analogie, la conception générale c'est comme déterminer les plans d'une maison avant sa construction. La conception détaillée est un peu comme définir où sera placée l'évier de cuisine.

Activité 4. Réalisation technique du nouveau système (implantation ou implémentation)

Le plus important produit de la réalisation technique est la portion informatisée du système d'information, c'est-à-dire le logiciel. Les responsables de cette activité devront aussi fournir de documents tels que des manuels d'utilisation et de la documentation sur le système.

Les principales tâches de la réalisation technique sont :

- Planification de la réalisation technique
- Conception physique
- Programmation
- Tests : unitaire, d'intégration
- Préparation et présentation de la documentation

Activité 5. Mise en place, exploitation et évaluation

Cette activité est celle qui assure le passage entre l'ancien et nouveau. Afin que ce passage s'effectue avec le minimum de heurts, il est important qu'il ait été planifié avec soin. Les principales tâches de la mise en place sont :

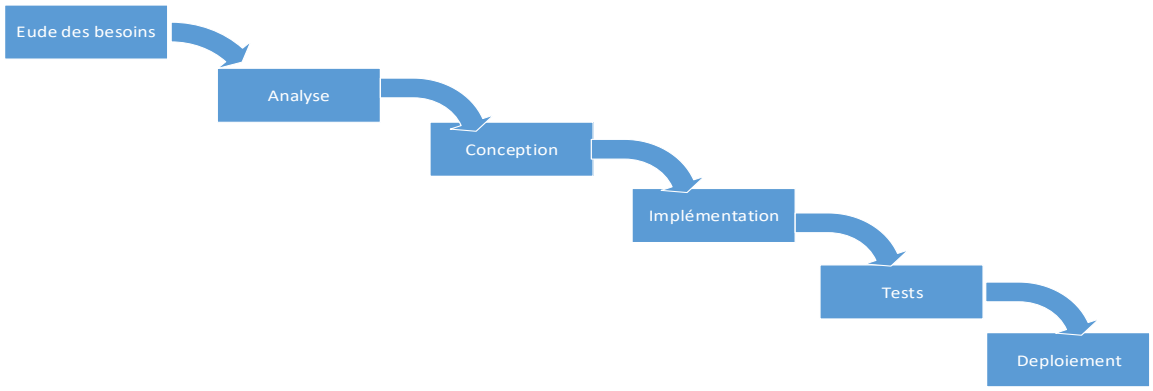
- Planification de la mise en place
- Conversion
- Exploitation et entretien
- Évaluation

Méthodes de développement de produit logiciel

Comme mentionné plus haut, la façon dont seront exécutées les différentes activités de développement va déterminer la méthode de développement.

Les méthodes en cascade (Waterfalls)

La plus populaire des méthodes traditionnelles. Ici, les activités sont exécutées les unes à la suite des autres.



Problèmes du développement en cascade :

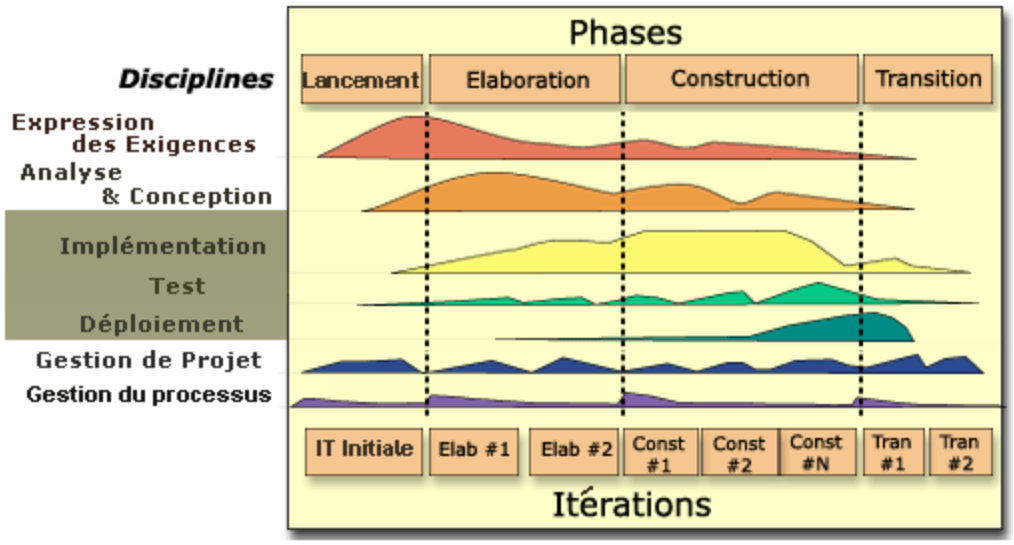
1. Les besoins du client pourraient changer, le processus de développement n'en tient pas compte.
2. Les erreurs, sont connues beaucoup plus tard ce qui rend leurs corrections difficiles.
3. Une erreur non détectée va se propager.
4. Le client ne voit son produit qu'à la fin du processus de développement.

Le processus Unifié (PU), l'ancêtre des approches agiles,

Le PU est itératif et incrémental. Pour le PU, il est important de découper le travail en plusieurs parties qui sont autant de mini-projets. Chacun d'entre eux représente une itération qui donne lieu à un incrément.

Les itérations désignent des étapes de l'enchainement des activités tant dis que l'incrément correspond à un stade de développement (activité) de produits. Pour mener à bien le projet, il est impératif de bien contrôler les itérations.

Contrairement aux méthodes d'analyse traditionnelles, le processus unifié, suggère une approche incrémentale du développement par une succession d'affinement ou d'itération autour de grandes disciplines du développement incluant l'expression des exigences, l'analyse et la conception, l'implémentation, le test et le déploiement. (Voir schéma)



Source : https://www.ibm.com/developerworks/rational/library/may05/brown/brown.html

Le RUP, Rational Unified Process est le processus de développement de la compagnie IBM. Le RUP utilise UML (diagrammes des cas d'utilisation, diagrammes de classes, diagrammes de déploiement, etc…) Pour la modélisation de systèmes. Le RUP est Itératif et Incrémental en plus d'être un processus de développement orienté objet.

Méthodes agiles : (détaillées plus loin)

Utilisées pour développer des produits en particulier à base de logiciels, les méthodes agiles visent à apporter plus de valeurs aux clients et aux utilisateurs ainsi qu'une plus grande satisfaction dans leur travail aux membres de l'équipe.

Le développement s'effectue par itérations successives, il est possible à la fin de chaque itération de changer les priorités pour faire en sorte que les éléments apportant le plus de valeurs soient réalisées en premier. Cela permet de maximiser la valeur ajoutée.

Les méthodes agiles sont basées sur un développement itératif et incrémental

Source de l'image : https://en.wikipedia.org/wiki/Iterative_and_incremental_development

Chapitre 2, Recueillir efficacement les besoins

Tout projet vise à atteindre les objectifs fixés par le client et à obtenir sa satisfaction. Or environ 45% des fonctionnalités développées ne sont jamais utilisées. Seulement 7 % des fonctionnalité développées sont toujours utilisées. (Voir figure1).

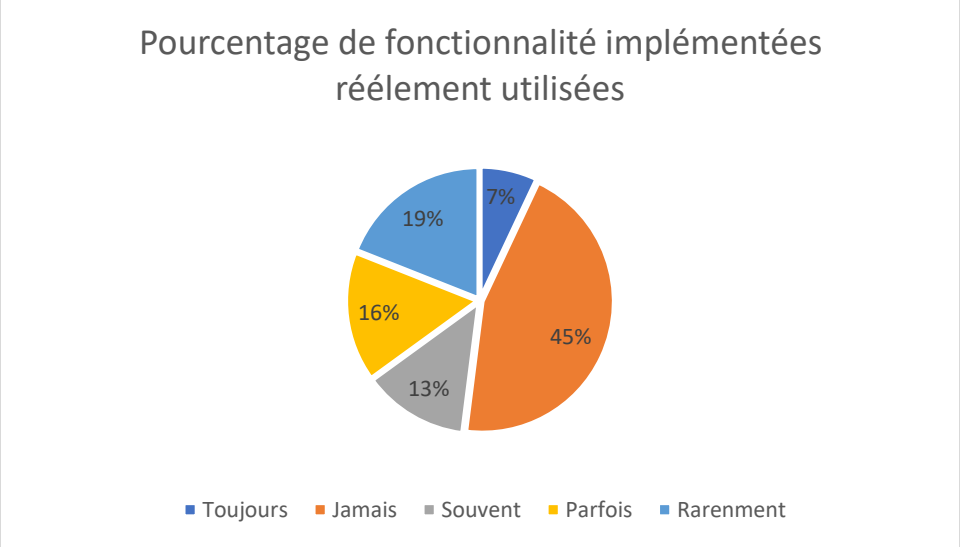


Figure 1, (Source : Gestion de projet agile (3e Edition, Véronique Messager Rota)

Par ailleurs, on constate que plus de 50% des défauts logiciels sont liés aux besoins. Les besoins du client ne sont pas bien compris (voir figure2)

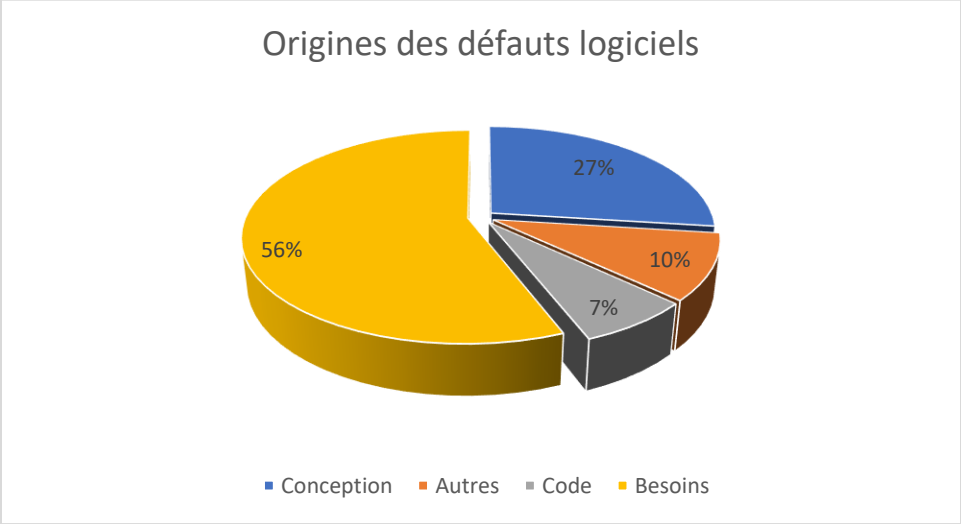


Figure 2, Origines des défauts logiciels (Source : Gestion de projet agile (3e Edition, Véronique Messager Rota)

Comment est-il possible de développer des fonctionnalités qui ne seront jamais utilisées ?

- Est-il possible que nous ayons mal communiqué avec le client ?
- Est-il possible que nous ayons mal compris les besoins du client?
- Le client a-t-il bien exprimé ses besoins ?
- Les attentes du client sont-elles clairement exprimées

Partager une vision

La vision d'un produit ou d'un projet est l'orientation générale donnée à l'équipe, l'objectif global à atteindre. Sans objectif, sans vision l'équipe de projet ne va nulle part. *« il n'est pas de vent favorable à celui qui ne sait où il va »*, Séneque

Cette vision repose sur une étude d'opportunité et/ou une analyse de l'existant qui font émerger un besoin, une nouvelle opportunité, un lancement d'un nouveau produit. Puis une étude de faisabilité analyse techniquement et financièrement si le projet peut être démarré. (Ou s'il existe un produit sur le marché qui peut être paramétré).

Exemple : Exercice de l'ascenseur de Geoffry Moore :

Vous rencontrez votre patron dans l'ascenseur et vous demande de décrire un peu le projet sur lequel vous êtes en train de travailler. Vous avez environ deux minutes pour le faire.

Votre réponse :

L'utilisateur …..a un besoin ……. Que le produit que nous allons développer, qui est un produit de type …… va satisfaire ainsi … …contrairement à d'autres solutions existantes, notre produit se positionne de cette façon.

1. For (target customers)
2. Who are dissatisfied with (the current market alternative)
3. Our product is a (new product category)
4. That provides (key problem-solving capability).
5. Unlike (the product alternative),
6. Our product (describe the key product features).

Faire émerger les besoins :

Définition : Un besoin :

Un besoin est défini à partir des objectifs de la vision du projet

Un besoin n'est pas uniquement une fonction assurée par le système, comme par exemple, imprimer un bulletin, enregistrer une commande etc,… C'est aussi la capacité du système d'assurer cette fonction (la disponibilité, la performance, l'évolutivité). Un besoin se définit

également par les services associés (modalités d'exploitation, support utilisateurs), les contraintes d'utilisation (ergonomie, organisation des utilisateurs…)

Émergence des besoins

Les besoins émergent au fur et à mesure pour les raisons suivantes :

- Entre la représentation qu'a le client de son futur produit, la difficulté parfois à le décrire, l'interprétation possible de cette description par le l'équipe de réalisation et le produit qui est livré au final, il y a de nombreux risques de perdre le besoin initial.
- Une idée initiale peut s'avérer inutile, trop coûteuse, ou trop « secondaire » après analyse.
- Tous les besoins n'ont pas la même priorité.
- Le client pourrait introduire une autre demande, renoncer à une demande.

La meilleure façon de recueillir les besoins et de les prioriser est de le faire de manière itérative et incrémentale

Les besoins du client doivent être recueillis, estimés et priorisés de manière itérative et incrémentale.

Recueillir les besoins :

Plusieurs techniques sont utilisées pour recueillir les besoins.

Le Brainstorming :

Technique idéale pour « défricher » les besoins encore flous ou mal organisés par les utilisateurs lors du démarrage du projet. Le brainstorming peut se faire sous forme de petites rencontres (1 à 2) durant lesquelles les utilisateurs peuvent exprimer ce qui leur parait important dans le projet. Personne ne se censure. Un facilitateur pourra guider le groupe à hiérarchiser les besoins.

L'interview :

C'est la technique la plus directe pour approfondir un besoin. Un utilisateur à la fois. C'est une technique simple mais qui doit être préparée d'avance. L'interview doit se planifier. Fixer une date, une heure début et une durée. Les questions de l'interview doivent être minutieusement préparées. C'est à vous de guider l'interview. Lors de l'interview, un utilisateur qui s'ennuie peut vous mener en dehors de votre objectif (il va vous raconter sa vie). C'est à vous de veiller à ce que l'objectif de l'interview soit atteint. À la fin de l'interview, vous devez peut-être valider avec le supérieur hiérarchique de l'utilisateur.

Le Questionnaire :

On utilise un questionnaire lorsque nous voulons avoir l'avis d'un grand nombre d'utilisateurs sur un certains nombres de besoins particuliers. Combinant des questions fermées, ciblées et questions ouvertes le questionnaire **ne doit pas** induire (guider) les réponses.

Les questions du questionnaire doivent être claires et précises. Le questionnaire peut être anonyme ou non.

Exemple pour améliorer l'utilisation de Colnet par les étudiants au CLG, on pourrait envoyer un questionnaire à tous les étudiant en ciblant des problèmes bien précis. Il serait inutile de rencontrer les étudiants un à la fois.

L'observation des utilisateurs en action ou en situation

Cette technique est souvent utilisée par les ergonomes pour comprendre le comportement des utilisateurs sur leur poste de travail. Exemple, nbre de cliques faits pour atteindre un menu, les astuces que note l'utilisateur pour palier au manquement de son application (post-it, prise de notes, etc.…)

L'analyse de l'existant :

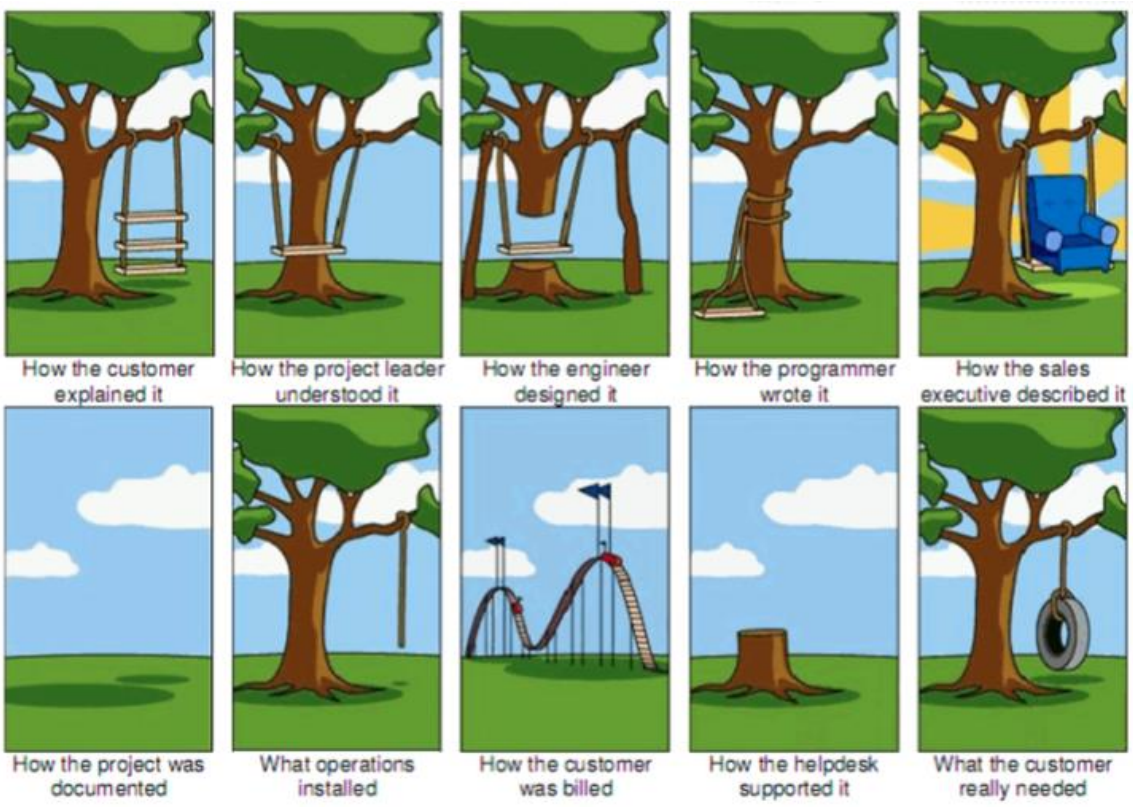
Ici, il s'agit d'examiner le système en place et de voir ses forces et ses faiblesses.

Formaliser les besoins :

Une fois les besoins recensés, la question est de comment « les garder » de façon à les retrouver facilement ?

Traditionnellement, l'équipe les consigne dans un document linéaire nommé « cahier de charge » qui est exprimé dans un langage de niveau métier en termes d'usage ou de services attendus. Ces besoins sont ensuite introduits dans un cycle de développement en passant par les différentes étapes.

Comment garder un lien entre les besoins initiaux et les besoins des étapes intermédiaires ? Comment s'assurer que tous les besoins sont traités ? En d'autres mots, comment limiter la rupture dans le processus de développement ?



Source de l'image : https://www.supinfo.com/articles/single/7225--definition-besoin

Il faudra trouver, des manières de formaliser des besoins qui va garantir que les besoins exprimés seront ceux produits. Il faudra avoir un langage commun entre les différents intervenant dans le projet. On privilégiera le langage de l'utilisateur orienté vers les besoins et les fonctionnalités. On s'intéressera à **QUOI ?.** On laissera le COMMENT à l'étape de conception.

Il existe trois approches pour la formalisation des besoins : La nome IEEE830 (Institute of Electrical and Electronics Engineers) , les Use-Case, et les User-Stories. Dans ce cours, on va privilégier les **user stories**.

La norme IEEE830-1998 :

La norme IEEE830 définie un document de spécification des exigences logiciels (SEL), en décrivant les exigences de la manière suivantes :

1. Doit être exprimée de manière claire, concise et cohérente
2. Doit être non ambiguë
3. Doit être valide
4. Doit avoir un bénéfice qui l'emporte sur le coût
5. Doit être vérifiable
6. Doit être identifiable de manière unique
7. Doit être modifiable (car elle évolue)

8. Doit être importante dans la résolution du problème
9. Doit être réaliste en considérant les ressources disponibles.

Exemple :

À la demande de l'étudiant le système affichera ses résultats dans le cours de KBE pour la session H2020.

Les use-case ou les cas d'utilisation d'UML (Unified Modeling Langage)

Les cas d'utilisation est une technique de formalisation des besoins utilisée dans le processus unifié. Le diagramme de cas d'utilisation d'UML est utilisé pour représenter les besoins du système ou ce que le système doit faire. Ils décrivent le comportement du système du point de vue de l'utilisateur.

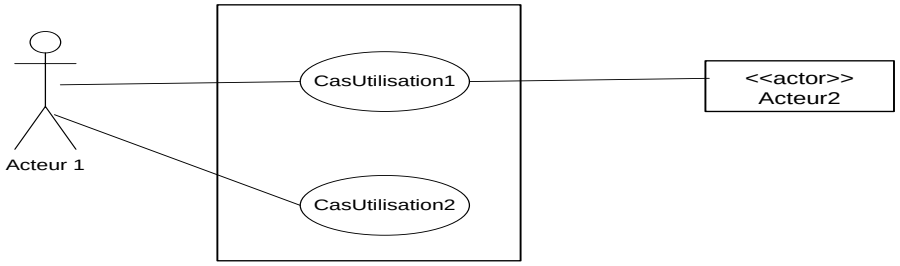
Un cas d'utilisation est un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable pour un acteur particulier du système. Il permet de décrire ce que le futur système DOIT faire sans spécifier comment il le fera. Il permet de voir les échanges entre le système et les acteurs du système.

Un acteur est une entité externe qui agit sur le système. Un acteur peut consulter et /ou modifier l'état du système en émettant et/ou en recevant des messages susceptibles d'être porteurs de données.

Objectifs des cas d'utilisation :

- Permettent de structurer les besoins des utilisateurs et les objectifs correspondants d'un système.
- Ils centrent l'expression des exigences du système sur ses utilisateurs Ils se limitent aux préoccupations "réelles" des utilisateurs ; ils ne présentent pas de solutions d'implémentation et ne forment pas un inventaire fonctionnel du système.
- Ils identifient les utilisateurs du système et leur interaction avec celui-ci

Représentation d'un cas d'utilisation

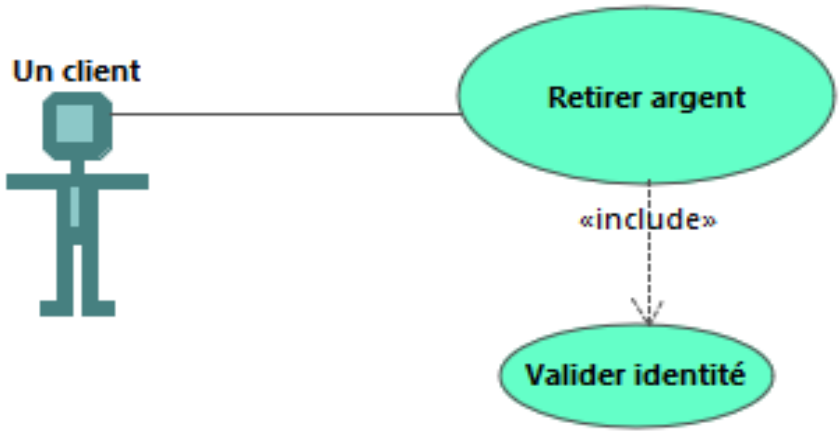


Relation entre cas d'utilisation :

Relation d'inclusion:«include» ou «utilise» ou «use»

Un cas d'utilisation de base (Emprunter un livre) incorpore explicitement un autre cas d'utilisation (s'identifier) de **façon obligatoire** à un endroit précis de son enchaînement. Le but est d'éviter de décrire plusieurs fois le même cas d'utilisation

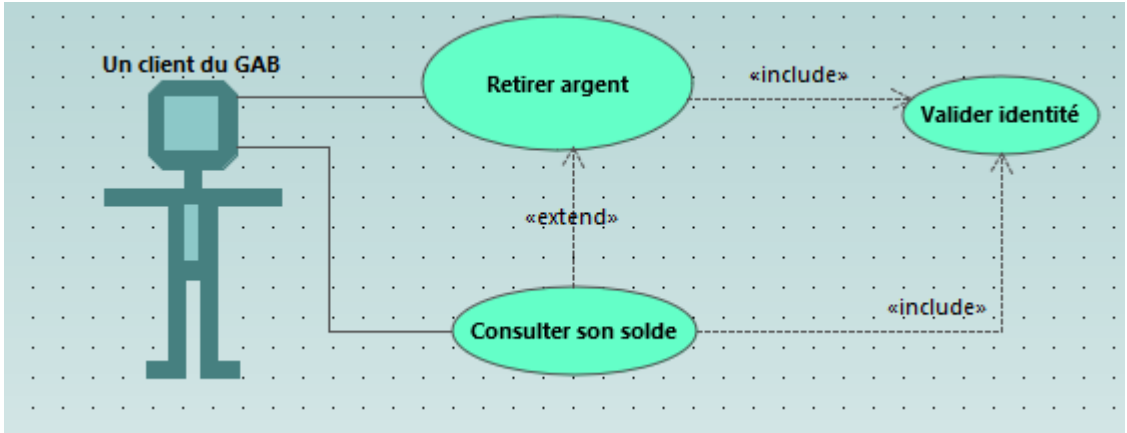
Exemple:



Relation d'extension: B est la suite logique de A

Un cas d'utilisation A (consulter la liste des abonnés) peut étendre son action, pas obligatoire, sur un autre cas d'utilisation B (imprimer la liste des abonnés). **Les deux cas peuvent s'exécuter de manière indépendante.**

Exemple:



Les cas d'utilisation *Retirer argent* et *Consulter son solde* sont deux cas indépendants, en ce sens qu'un client du GAB peut consulter son solde sans retirer de l'argent ou qu'il peut retirer de l'argent sans effectuer d'interrogation de solde.

Cependant le cas *Consulter son solde* peut etendre son action au cas *Retirer de l'argent*. Un client après avoir consulter son solde peut décider de retirer de l'argent.

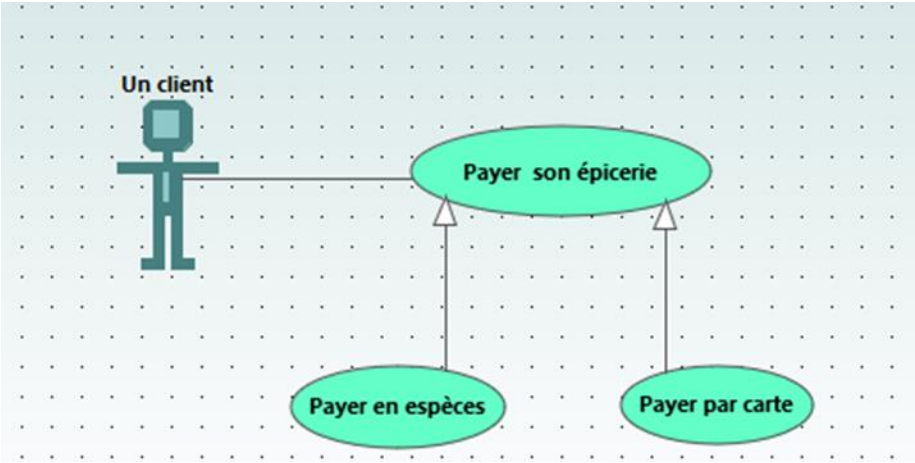
Quel que soit le cas d'utilisation à exécuter (*Retirer argent* ou *Consulter son solde*), ils doivent faire appel (utilise) au d'utilisation *Valider identité* (carte débit et nip). On ne peut retirer de l'argent si nous n'avons pas de carte bancaire valide et un nip valide

La relation de généralisation.

Un cas A est une généralisation d'un cas B si B est un cas particulier de B

Cette relation de généralisation/spécialisation est présente dans la plupart des diagrammes UML et se traduit par le concept d'héritage dans les langages orientés objet.

Exemple :



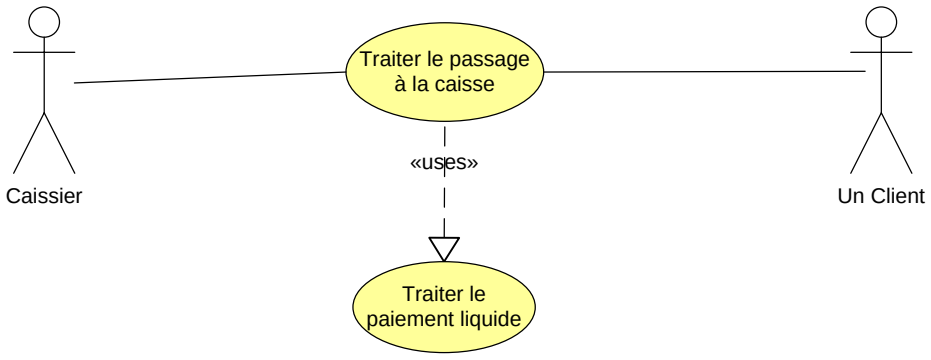
Exemple. Caisse enregistreuse[1]

Le déroulement normal des opérations d'une caisse enregistreuse (point de vente) est le suivant :

- Un client arrive à la caisse avec des articles à payer
- Le caissier enregistre le n° d'identification de chaque article, ainsi que la quantité si elle est supérieure à 1
- La caisse affiche le prix de chaque article et son libellé.
- Lorsque tous les achats sont enregistrés, le caissier signale la fin de la vente.
- La caisse affiche le total des achats.
- Le client choisit son mode de paiement.
    1. Liquide: le caissier encaisse l'argent reçu, la caisse indique la monnaie à rendre au client.
    2. Carte de débit
    3. Carte de crédit : un terminal bancaire fait partie de la caisse. Il transmet une demande d'autorisation en fonction du type de la carte.
- La caisse enregistre la vente et imprime un ticket.
- Le caissier donne le ticket de caisse au client.

Lorsque le paiement est terminé, la caisse transmet les informations sur le nombre d'articles vendus au système de gestion des stocks.
Tous les matins, le responsable du magasin initialise les caisses pour la journée.

Exemple de représentation : (le diagramme n'est pas complété)



[1] **Sources UML** 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles

Description des cas d'utilisation :

Les cas d'utilisation ont besoin d'être décrits soit textuellement, soit en utilisant un autre diagramme. Deux parties sont importantes lors de la description d'un cas d'utilisation.

1) Le sommaire d'identification
    a. Le titre
    b. Résumé
    c. Acteurs
    d. Date de création
    e. Version
    f. Date de mise à jour
    g. Responsable
2) Le scénario nominal.
    a. Préconditions
    b. Scenario nominal (enchaînement des opérations dans le cas où le cas d'utilisation se déroule normalement.)
    c. Scénario alternatif
    d. Enchaînement des erreurs
    e. Postconditions

Exemple de Description des cas d'utilisation[2]

Titre: Traiter le passage à la caisse

Brève description: Un client arrive à la caisse avec des articles qu'il souhaite acheter.

Acteurs: Caissier (principal), Le client (secondaire)

Préconditions:

- Le terminal de point de vente est ouvert
- Un caissier y est connecté
- La base de données des produits est disponible

Postcondition.

La vente est enregistrée dans le terminal de vente.

[2] **Sources UML** 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles

Enchaînement des opérations (scénario nominal ou déroulement normal des opérations)

Acteurs	Système
1. Le cas d'utilisation débute lorsqu'un client arrive à la caisse avec des articles à acheter.	
2. Le caissier enregistre chaque article. Le caissier indique également la quantité pour chaque article s'il y a lieu.	3. Le terminal de vente valide le numéro d'identification de chaque article. Le terminal de vente affiche la description et le prix de chaque article.
4. Après avoir enregistré tous les articles, le caissier indique que la vente est terminée.	5. Le terminal de vente calcule et affiche le montant de la vente.
6. Le caissier annonce le prix au client.	
7. Le client choisi de payer en liquide. Exécuter le cas « paiement en espèces	
	8. Le terminal de vente enregistre la vente et imprime un ticket.
9. Le caissier remet le ticket de caisse	
10. Le client s'en va	

Scénarios alternatifs

A1: Numéro d'identification du produit inconnu.

L'enchaînement de A1 démarre au point 3 du scénario nominal

Le terminal de vente indique que le numéro du produit est inconnu. L'article ne peut être prix en vente. Le scénario reprend au point 2 s'il y a d'autres produit ou au point 4 s'il n'y en a pas

A2: le client demande l'annulation de l'article (article trop cher)

L'enchaînement de A2 démarre au point 2 du scénario nominal

2- Le caissier demande l'annulation de l'article
3- Le terminal de vente supprime l'article de la vente.

Le scénario nominal reprend au point 2, s'il y a d'autres articles ou au point 4 sinon.

Les user-stories :

Une **user-story** ou un **scénario** est une exigence du système à développer formulée en une ou deux phrases dans le langage des utilisateurs pour servir un but. Sa granularité doit permettre à

l'équipe de réalisation d'estimer son coût et de la réaliser entièrement à l'intérieur d'un sprint (ou une itération).

L'avantage des user-stories est qu'elles facilitent la démarche en deux temps : générales et grossières au début, elles s'enrichissent ensuite d'exemples, de tests d'acceptation. Elles facilitent la communication, l'ajout ou la suppression de détails.

Une user-story s'écrit comme suit;

En tant que <rôle>

Je veux <liste de tâches>

Afin de <valeur ajoutée ou résultat>

Exemples

1. En tant qu'acheteur en ligne, je veux pouvoir ajouter des items à mon panier, supprimer les items afin de pouvoir n'acheter que ce dont je suis vraiment certain.
2. En tant que client, je peux consulter la liste des factures émises.
3. En tant que client (du projet), je peux consulter la liste de mes clients.
4. En tant que client, je peux connaître le montant total des factures impayées

Propriétés INVEST d'une User story

Une bonne user story doit respecter l'acronyme INVEST

I pour **Indépendante** : lorsque le client peut en toute liberté décider de l'ordre dans lequel les scénarios (story) est implémenté sans qu'interviennent des contraintes techniques. Si des stories sont fortement liées, alors il faut les combiner

Bon exemple :

En tant que joueur, je veux obtenir une question aléatoire d'une catégorie afin de pouvoir y répondre.

Mauvais exemple :

En tant que joueur je veux répondre à une question (qui dépend d'obtenir une question)

N pour **Négociable** : L'équipe de développement n'est pas contrainte par la manière dont sera implémenté le scénario : Elle a la latitude d'imaginer une solution efficace

Bon exemple :

En tant que client, je peux connaître le montant total des factures impayées.

Mauvais exemple

En tant que client, je veux, lorsque je clique sur le bouton « calculer » une ligne s'ajoute et on affiche sur cette ligne le montant total des factures impayées.

V pour **Vertical** : (V pour Valuable , a une valeur ajoutée) un bon scénario ou story décrit une fonctionnalité complète de l'application dont le client apprécie l'intérêt à l'intérieur d'une itération.

Bon exemple :

Un bon découpage qui fait apparaitre les scénarios distincts rendus par le service de facturation :

- **Consulter la liste des factures émises**
- **Afficher la liste des factures ordonnées par date**
- **Afficher la liste des factures par adresse de livraison**
- **Consulter une facture client**
- **…**

Mauvais exemple :

Créer la base de données pour le module de facturation. Créer l'interface graphique pour la facturation

E : pour **Estimé** : pour que le scénario puisse servir de base à la planification, on doit connaître au moins une estimation du coût d'implémentation.

Bon exemple :

Implémenter les règles métier R1, et R2

R1 : si le nom de l'utilisateur existe déjà, lorsque l'acheteur en ligne essaye de créer son compte, alors le message « ce compte existe déjà » doit être affiché.

R2, un rabais de 5% est octroyé pour tous les clients dont le montant total de la facture dépasse les 1000$.

Mauvais exemple :

Garantir le respect des règles de gestion en vigueur.

S pour **Suffisamment petit** (Size appropriately): toujours, pour planifier sa réalisation un scénario doit être réalisable en peu de temps pour que l'équipe de projet puisse en planifier dans un même sprint (itération)

Mauvais exemple : réaliser le module de facturation – il faudra découper –

Bon exemple : voir V pour vertical

T pour **Testable** : planifier un test d'acceptation est un excellent moyen pour vérifier que tout le monde a bien compris le scénario ou la story.

Exemple de test d'acceptation :

En tant que client, je veux connaître la liste des factures par date. **Test d'acceptation : afficher la liste des factures ordonnées par la date de facturation**

Les user-stories : Détails et tests d'acceptation

Ron Jeffries définit une story par trois composants :

- La carte pour enregistrer son titre;
- La conversation pour la raconter,
- La confirmation pour s'assurer qu'elle est finie : **Le critère d'acceptation**

Une story a besoin d'être d'écrite par des détails pertinents par l'équipe de développement afin de pouvoir l'estimer.

Une story doit être terminée par un test d'acceptation.

Un test d'acceptation est la pratique qui permet d'accepter une story à la fin d'une itération. Il en consiste en :

- **Décrire le comportement attendu avec les conditions de satisfaction**
- Transformer ces conditions en cas de test : **storytest**
- Écrire le code applicatif qui répond au comportement attendu
- Passer la storytest sur le code applicatif. En cas d'échec, corriger les tests ou le code. En cas de succès la story est terminée

D'autres informations pourraient être ajouter pour compléter la présentation d'une story, par exemple il est important d'indiquer un **test d'acceptation** pour la story. On verra les tests d'acceptation plus loin.

Exemple 1

Story 10 En tant que client, je veux connaître la liste des factures par date.
Test d'acceptation : afficher la liste des factures ordonnées par la date de facturation

Exemple 2

Story 100 En tant que candidat potentiel pour un poste chez l'entreprise SigmaPlus, je veux pouvoir créer mon profil pour pouvoir postuler à des postes disponibles.
Test d'acceptation : • Tous les champs du formulaire ont été validés. • Les informations sont enregistrées. • Un courriel de confirmation est envoyé au candidat.

Détails : liste des tâches à faire pour réaliser la story 100 (le candidat doit créer son profil)

1. Créer le formulaire d'inscription chez l'entreprise SigmaPlus avec tous les champs : en d'autres mots créer le profil du candidat.
2. Indiquer les champs obligatoires
3. Valider toutes les entrées du formulaire.
4. En envoyant le formulaire celui-ci sera enregistré correctement.
5. Envoyer une confirmation par courriel au candidat.

Exemple 3:

En tant qu'étudiant du CLG, je veux me connecter au site ColNet afin d'accéder à mon dossier scolaire.
Test d'acceptation: • L'identifiant saisi est dans le bon format (on peut préciser le format : ex: email). • Le mot de passe est dans le bon format (on peut préciser le nb caractères max ou format). • L'étudiant est identifié ou non.

Étudiant identifié :
Étant donné que la combinaison de l'identifiant et du mot de passe est bonne, l'étudiant est identifié et la page d'accueil du site ColNet lui est présentée.

Étudiant non-identifié
Étant donné que la combinaison de l'identifiant et du mot de passe n'est pas bonne, un message informant l'étudiant s'affiche et il doit ressaisir les informations de connexion.
Exemple 4 :

En tant qu'intéressée, je veux m'inscrire à la conférence sur le climat « SOS Terre » déroule à la salle « Talbot »
Test d'acceptation : Inscription acceptée ou refusée.

Inscription acceptée :

Étant donnée l'utilisateur Primogene connecté à la conférence « SOS Terre » qui se déroule à la salle Talbot d'une capacité de 200 places et dont le nombre d'inscrit actuels est 152. Quand Primogene s'inscrit à « SOS Terre » alors, l'inscription est acceptée, un message de confirmation est envoyé à Primogene et le nombre d'inscrits sera augmenté de 1.

Inscription refusée

Étant donnée l'utilisateur Patoche connecté à la conférence « SOS Terre » qui se déroule à la salle Talbot d'une capacité de 200 places et dont le nombre d'inscrit actuels est 200. Quand Patoche s'inscrit à « SOS Terre » alors, l'inscription est refusée, un message de refus est envoyé à Patoche

Conclusion : comparaison user-stories et use-case

User Story	**Use case**
C'est une brève description d'une fonctionnalité telle que vue par l'utilisateur.	Représente une séquence d'action qu'un système peut accomplir en interagissant avec les acteurs du système
Mode orale et **collaboratif**	Mode écrit et souvent **distant**
Grande lisibilité vu sa simplicité	Pourrait souvent manquer de lisibilité (volume)
Format écrit court laissant part à de belles discussions orales	**Format écrit très riche en information** (postcondition, scénario …) peu de place à l'oral
Utilisée pour spécification des exigences mais surtout pour **estimation de la planification**	Utilisé seulement en tant que spécification

Émergence rapide au travers des ateliers de collaboration	Long **travail d'analyse et de formalisation**
Implémentée et testé en une itération seulement	Implémenté et testé en plusieurs opérations
Contient des tests d'acceptation	Ne contient pas les cas de test qui en découle.
Difficile à lier les unes aux autres: Absence de vue globale	**Liaison et vue globale faciles** au travers du diagramme des cas d'utilisation qui lient les use case
Associée au méthodes agiles comme **SCRUM, XP**	Associée généralement au **Processus Unifié**
Très facile à maintenir	Plus difficile à maintenir

Chapitre 3, Introduction aux méthodes agiles :

Utilisées pour développer des produits en particulier à base de logiciels, les méthodes agiles visent à apporter plus de valeurs aux clients et aux utilisateurs ainsi qu'une plus grande satisfaction dans leur travail aux membres de l'équipe.

Le développement s'effectue par itérations successives, il est possible à la fin de chaque itération de changer les priorités pour faire en sorte que les éléments apportant le plus de valeurs soient réalisées en premier. Cela permet de maximiser la valeur ajoutée.

Définition selon Scott Ambler « *Une méthode agile est une approche itérative et incrémentale pour le développement de logiciel, réalisé de manière très collaborative par des équipes responsabilisées appliquant un cérémonial minimal, qui produisent, dans un délai contraint, un logiciel de grande qualité répondant aux besoins changeants des utilisateurs* »

Manifeste agile

Le *Manifeste* agile ensemble de valeurs et de principes qui régissent les méthodes agiles

Valeurs : publié au début des années 2000 et inchangé depuis, le *Manifeste agile* définit une attitude de réaction par rapport au processus lourds et bureaucratiques en vogue à l'époque (et même aujourd'hui). La position prise par rapport à ces processus ne définit pas les valeurs intrinsèques de l'agilité mais des valeurs relatives :

Valeurs agiles

- Les personnes et leurs interactions sont plus importantes que le processus et les outils
- Un logiciel qui fonctionne prime sur la documentation
- La collaboration avec les clients est préférable à la négociation contractuelle
- La réponse au changement passe avant le suivi d'un plan.

Les principes :

Le Manifeste annonce douze (12) principes, qui ne définissent pas une méthode agile.

1. Satisfaire le client en livrant tôt et régulièrement des logiciels utiles qui offre une véritable valeur ajoutée.
2. Accepter les changements même tard dans le développement.
3. Livrer fréquemment une application qui fonctionne.
4. Collaborer quotidiennement entre clients et développeurs.
5. Bâtir le projet autour de personnes motivées en leur fournissant environnement et support, et en leur faisant confiance.
6. Communiquer par des conversations face à face.
7. Mesurer la progression avec le logiciel qui fonctionne.
8. Garder un rythme de travail durable.

9. Rechercher l’excellence technique et la qualité de conception
10. Laisser l’équipe s’auto-organiser.
11. Rechercher la simplicité.
12. À intervalle régulier, réfléchir aux moyens de devenir plus efficace.

Si les principes et les valeurs sont universels, la façon de les mettre en œuvre sur des projets varie. Cette application se fait par l’intermédiaire de ce qu’on appelle une pratique.

Une pratique est une approche concrète et éprouvée qui permet de résoudre un ou plusieurs problèmes courants ou d’améliorer la façon de travailler lors d’un développement.

Présentation de quelques méthodes agiles :

XP pour eXtreme Programming:

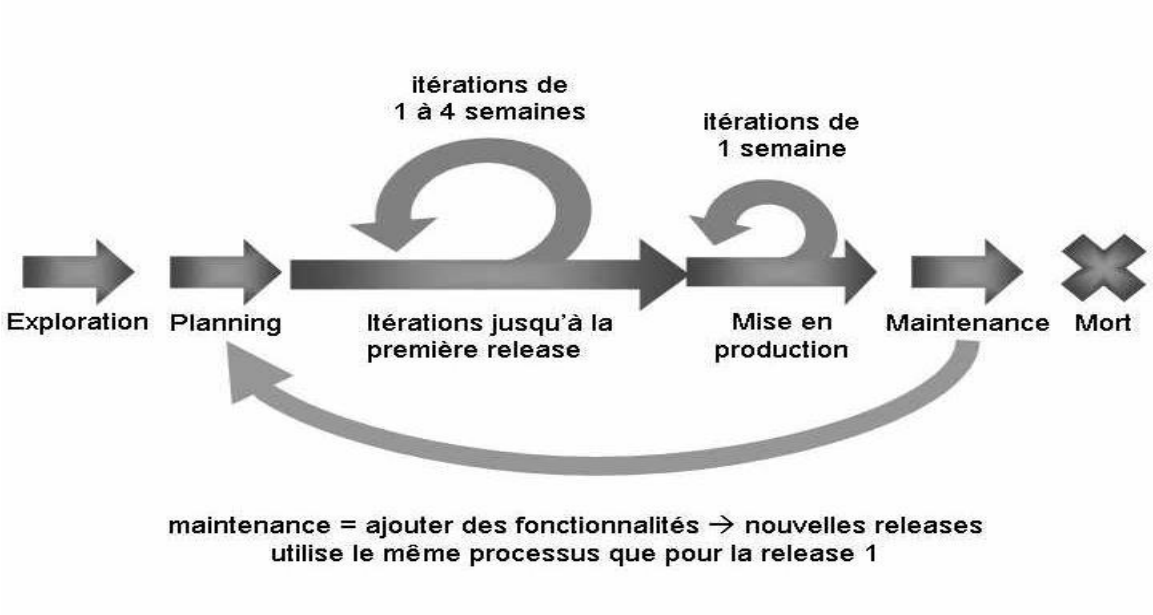
Adaptée aux équipes réduites avec des besoins changeants. Elle pousse à l'extrême des principes simples. Son but principal est de réduire les coûts du changement. XP s´attache à rendre le projet plus flexible et ouvert au changement en introduisant des valeurs de base, des principes et des pratiques.

Les principes de cette méthode ne sont pas nouveaux : ils existent dans l'industrie du logiciel depuis des dizaines d'années et dans les méthodes de management depuis encore plus longtemps. L'originalité de la méthode est de les pousser à l'extrême :

Pratique de l’XP

- Planning game
  - ✓ phase d'exploration : écriture des besoins sous forme de "user stories" et estimation de leur durée
  - ✓ phase d'engagement : classement des user stories par ordre de priorité.
  - ✓ phase de direction : mise à jour du planning
- Petites releases
- Utilisation de métaphores pour décrire l'architecture du système
- Conception simple : toujours développer la solution le plus simple possible
- Tests (unitaires et fonctionnels)
- Refactoring du code : retravailler le code pour le rendre plus lisible et plus robuste
- Programmation en binôme
- Propriété collective du code
- Intégration continue (plusieurs fois par jour)
- Pas de surcharge de travail : ne pas dépasser 40 heures de travail par semaine
- Client sur site : présence sur site d'une personne minimum à temps plein pendant toute la durée du projet
- Standards de code (normes de nommage et de programmation)

Cycle de vie de XP :



Source de l'image: https://fr.slideshare.net/MedAymen/method-xp

Kanban [3]

À la base, Kanban a été développé à la fin des années 1940 chez Toyota afin d'optimiser l'efficacité de ses chaînes de montages. Basé sur la méthode de remplissage des supermarchés, cette méthode visait à la base à optimiser les niveaux de stocks pour les garder aux niveaux nécessaires et éviter tout genre de « sur-stockage », sans pour autant manquer de pièces.

Pour être capable de communiquer les niveaux de stock dans l'usine, les employés utilisaient des cartes ou des signaux visuels afin d'indiquer un manque d'une certaine pièce et indiquer à la production d'en faire et quels matériaux utiliser.

Les avantages de Kanban / Comment l'utiliser durant le développement logiciel?

Premièrement, il faut savoir que la méthode Kanban pour le développement logiciel est basée sur les principes et les valeurs Agile. Quels avantages cela nous apporte-il?

Planification flexible : Il faut savoir accepter les changements en tout temps. Le client change d'idée durant le développement logiciel et il faut savoir intégrer ces changements tant que cela

[3] Alexandre André-Lespérance

n'a pas d'impact sur les tâches en cours. De plus, le Product Owner (nous verrons ce rôle un peu plus tard, mais on peut le voir comme le décideur ici) peut se permettre de changer l'ordre des tâches futurs en toute temps sans jamais impacter le travail en cours.

Durée de cycle raccourcies : La durée du cycle est une des métriques les plus importantes dans la méthode Kanban. Elle représente la durée pour qu'une équipe de travail passe à travers son « workflow » ou sa charge de travail avant la prochaine livraison. Les équipes suivent de bonnes pratiques de bases lors de la production et ensuite applique le « code review » pour s'assurer que la qualité ne sera pas impactée par ce laps de temps raccourcie.

De plus, nous devons nous assurer que les équipes ont des compétences partagées, c'est-à-dire que les compétences se chevauchent et que plus d'une personne est capable d'effectué chacune des tâches. Cela nous permet d'être plus constant dans la livraison des livrables et de s'assurer que le travail est protégé du départ d'un employé.

Réduction des goulots d'étranglement : Avec l'utilisation du Kanban Board (nous le verrons plus tard), nous pouvons suivre l'état de chacune des tâches en état. Par exemple : « À faire (Backlog) », « En cours », « En révision/Test » et « Complété ». La méthode Kanban propose de limiter le nombre maximal de tâches pouvant être dans un certain état à tout moment. Par exemple : On peut décider de mettre un maximum de 4 tâches en cours et 2 tâches en révision en même temps afin d'éviter un bouchon et que certains employés n'aient rien à faire.

Métriques visuelles : L'une des valeurs de Kanban est la constante amélioration de l'efficacité de l'équipe. Pour y arriver Kanban utilise plusieurs métriques visuelles afin de s'assurer de l'amélioration constante des processus. Un des principes dicte que si l'information est facile d'accès par l'équipe, il sera facile d'identifier les goulots d'étranglement et de les éviter/supprimer. (Des exemple sont disponibles ici : https://www.atlassian.com/fr/agile/kanban)

Livraison continue : Kanban est une méthode de travail qui prône l'intégration continue : C'est-à-dire, les builds automatisés et les tests incrémentiels tout au long du développement. Ceci nous permettra de livrer des fonctionnalités au client régulièrement.

Chapitre 4, La méthode agile SCRUM : Présentation

Présentation :

SCRUM signifie mêlée en rugby. Scrum utilise les valeurs et l'esprit du rugby et les adapte aux projets de développement. Comme le *pack* lors d'un ballon porté au rugby, l'équipe chargée de développement travaille de façon collective et soudée vers un objectif précis. Comme un demi de mêlée, le Scrum Master aiguillonne les membres de l'équipe les repositionne dans la bonne direction et donne le tempo pour assurer la réussite du projet.

Définitions :

Un sprint est le terme utilisé dans SCRUM pour désigner une itération. Dans le langage SCRUM un sprint est un bloc de temps fixée aboutissant à un incrément de produit potentiellement livrable.

Une release : (selon le dictionnaire du jargon informatique) : Version d'un système, par exemple un logiciel, effectivement publiée, donc lâchée dans la nature.

Une release représente aussi la durée nécessaire pour produire un produit logiciel livrable. En ce sens une release est composée de sprint. C'est cette définition qui sera retenue.

En bref, c'est quoi Scrum?

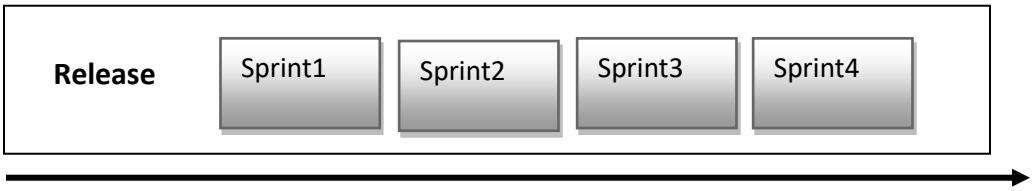
- ✓ Scrum sert à développer des produits, en quelques mois tout au plus. Les fonctionnalités souhaitées sont collectées dans le **backlog du produit** – **liste unique des user-stories**— et classées par priorités. C'est le **Product Owner** qui est tenu responsable de la tenue de ce *backlog.*
- ✓ Une **version (release)** est produite par une série d'itérations d'un mois, parfois même de 15 jours, appelés **sprint**. Le contenu d'un *sprint* est défini par l'équipe avec le Product Owner, en tenant compte des priorités et de la capacité de l'équipe.
- ✓ Pendant un **sprint***,* des points de contrôle sur le déroulement des tâches sont effectués lors des mêlées quotidiennes (*scrums*). Cela permet au **ScrumMaster,** l'animateur chargé de faire appliquer Scrum de déterminer l'avancement par rapport aux engagements et d'appliquer, avec l'équipe, des ajustements pour assurer le succès du *sprint*.
- ✓ À la fin de chaque **sprint***,* l'équipe obtient un **produit partiel**, (qui s'enrichit d'un nouvel incrément à chaque *sprint*) qui fonctionne. Il est potentiellement livrable. Son évaluation et le *feedback* récolté permettent d'ajuster le *backlog* pour le *sprint* suivant.

Cycle de développement SCRUM :

Aspect temporel :

Phases et Jalons (un jalon est une balise)

Dans le processus de développement SCRUM, le jalon mineur correspond à la fin d'un *sprint* et le jalon majeur correspond à la production de la release.



Une *release* est un ensemble de *sprints* qui se termine quand les incréments successifs constituent un produit qui présente une valeur à ses utilisateurs.

La durée d'une *release* est définie par l'équipe et le Product Owner. La tendance est de raccourcir ces durées.

Pour une équipe, une *release* dure environs 3 mois avec *des sprints* de deux à trois semaines. Ce qui permet d'avoir de quatre à 6 *sprints* dans une *release*.

Il n'y a pas de chevauchement. Les *sprints* s'enchaînent sans délais. Un *sprint* n'est commencé que si le précédent est terminé. Le nouveau *sprint* démarre immédiatement après le précédent.

La date fine d'un *sprint* est fixée au début de celui-ci. Elle ne change pas même si l'équipe ne réalise pas tout ce qu'elle pensait faire. Lorsque l'équipe se rend compte qu'elle ne peut pas présenter quelque chose à la fin du *sprint*, une possibilité est d'arrêter immédiatement le *sprint* en cours et repart aussitôt dans un nouveau *sprint* qui tient compte des difficultés.

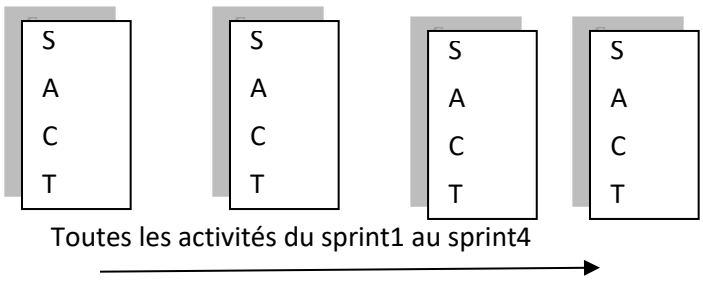
Arrêter un sprint plutôt que de l'étendre.

Activités et cycles de développement

Toutes les méthodes de développement utilisant l'approche agile, présentent un enchaînement de phases qui sont :

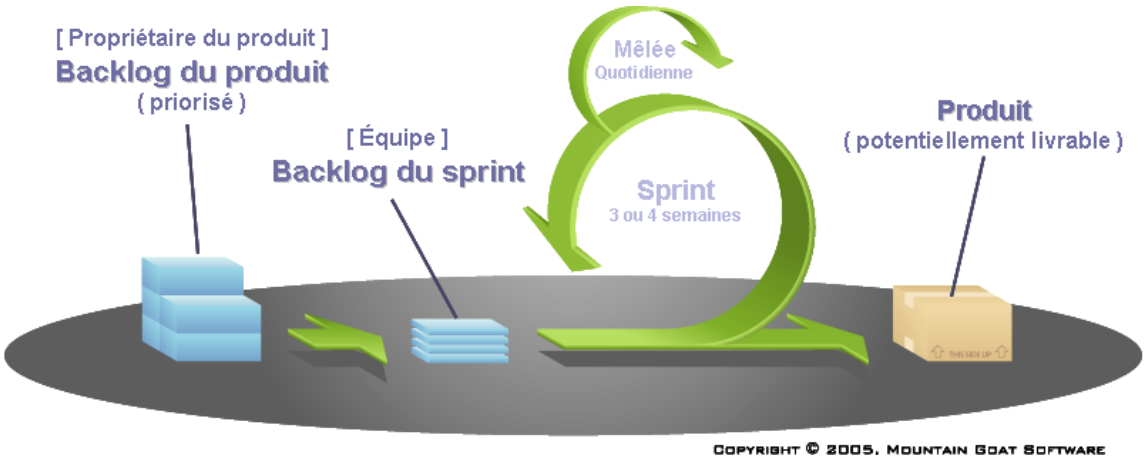
- Spécification fonctionnelle (Requierements ou étude des besoins fonctionnels)
- Architecture (conception)
- Codage (et test unitaire)
- Test (test d'intégration).

Cycle SCRUM : les sprints et leurs phases se déroulent en parallèle.



Le résultat d'un sprint

À la fin d'un sprint le résultat attendu est un incrément du produit final qui est potentiellement livrable. Voir figure suivante.



Source de l'image : https://fr.wikipedia.org/wiki/Scrum_(d%C3%A9veloppement)

Dans le cas d'un développement de logiciel le minimum est d'avoir déployé à la fin d'un sprint le produit potentiellement livrable avec si nécessaire de la documentation permettant son utilisation.

Le résultat d'une release.

Le résultat d'une *release* est le produit livrable fourni à ses utilisateurs. La façon dont il est fourni dépend de son déploiement.

Souvent, le jalon majeur que représente la *release* correspond à une annonce marketing.

Rôles et responsabilités : Le Product Owner et le Scrum Master

Le Product Owner : (PO)

Le PO est l'expert du domaine (niveau métier). En tant que représentant des clients et utilisateurs, il est responsable de définir les caractéristiques du produit développé par l'équipe, en termes de :

- Fonctionnalités offertes. Plus précisément, il identifie chaque exigence que doit satisfaire le produit et la collecte comme élément du backlog de produit. Il est souhaitable d'inclure les tests d'acceptation.
- Priorité. C'est lui qui définit l'ordre dans lequel ces éléments seront développés en fonction de la valeur qu'ils apportent aux clients et utilisateurs. Cela permet d'alimenter l'équipe avec un backlog de produit prêt pour la planification des sprints

- But. C'est lui qui définit l'objectif d'une release et qui prend les décisions concernant le planning de la release.
- Son implication dans le projet est capitale pour la réussite de celui-ci.

Le SCRUM Master :

C'est le coach de l'équipe (ancien chef de projet). Il a pour rôle:

- Dans le cadre du développement d'un produit, d'aider l'équipe à travailler de façon autonome et à s'améliorer constamment. Il est le garant de l'application du processus, Scrum en l'occurrence.
- S'assurer que l'équipe bénéficie des meilleures conditions pour accomplir les tâches
- Éliminer les obstacles : prendre en compte les problèmes qui surviennent à tout moment sur un projet pour les éliminer au plus vite, en évitant qu'ils ralentissent l'équipe. Il protège l'équipe des interférences extérieures.
- Faire en sorte que l'équipe reste concentrée sur le véritable objectif du projet, qui est de réaliser les éléments du Backlog en collaboration étroite avec le Product Owner , et soit productive. Il s'assure que chacun participe pleinement aux travaux de l'équipe.
- Organise et anime les réunions qui constituent le cérémonial.

Démarrer un projet SCRUM

Pour n'importe quel projet, n'importe quelle méthode de développement qui sera utilisé, n'importe quelle méthode de gestion de projet qui sera utilisé pour le mener à bien, la première étape sera de collecter et de formaliser les besoins du client. Ces besoins sont formalisés dans **un Backlog du produit**.

Le backlog du produit :

Après avoir décidé de lancer le développement d'un produit, la difficulté fondamentale est de transformer l'idée de départ (vision) en quelque chose d'assimilable par l'équipe de développement.

Contrairement aux méthodes traditionnelles, une équipe SCRUM ne consacre pas tout le début d'un projet à rédiger un document qui décrit en détail les spécifications fonctionnelles. Elle identifie les fonctions essentielles et les raffine progressivement. L'outil de collecte s'appelle le ***backlog du produit.***

Scrum n'impose pas de pratique pour identifier et nommer les éléments du *backlog*.

L'usage le plus courant est de définir un élément du backlog comme étant une *User Story*

Dans un backlog de produit, les stories sont rangées (Classées) selon l'ordre envisagé pour leur réalisation. Cette notion priorité prend une grande importance dans le développement itératif.

SCRUM est un développement léger et agile (phases de développement)

1. **Création du *backlog* (un to do list) de toutes les fonctionnalités d'un projet. Les éléments du backlog doivent être estimé (temps de réalisation) et priorisé.**
2. **Création d'un sprint backlog : fonctionnalités à compléter durant la durée du sprint (15 jours ou un mois)**
3. **Effectuer des rencontres quotidiennes durant le sprint : des mêlées quotidiennes (*scrums*).**
4. **Finalisation du sprint avec démonstration et évaluation**

Chapitre 5, SCRUM en action

Dans le développent utilisant la méthode SCRUM l'élément central est **le backlog** du produit. Sa production est la tâche la plus importante à réaliser avant de commencer la planification et le développement du produit. La production du backlog du produit n'est pas une tâche facile à faire. La question est comment passer d'une vision à backlog de produit ?

La figure suivante montre les étapes importantes pour la constitution d'un backlog initial.

Construire une bonne vision

Rappel : Une vision est une expression d'une volonté collective à développer un excellent produit. Une vision s'attache **au pourquoi ?** Elle présente ce qui permet de comprendre rapidement ce pourquoi le produit est développé. C'est l'objectif à atteindre ou l'orientation générale donnée à l'équipe.

Les éléments que nous pouvons trouver régulièrement pour décrire une vision sont :

- ✓ Le but ou l'objectif principal
- ✓ Les besoins des utilisateurs et des parties prenantes
- ✓ Une présentation de la solution qui répond à ces besoins
- ✓ Les contraintes importantes imposées au développement.

Il n'y a pas de standard quant au contenu de la vision.

Identifier les éléments d'une vision

La meilleure façon d'identifier les éléments d'une vision est de procéder par des séances de travail en groupe : **un *brainstorming* collectif**

On pourrait présenter une vision comme suit :

Le problème de	Décrire le problème en détail
Affecte	Les intervenants affectés par le problème
Il en résulte	Quel est l'impact du problème
Une solution réussie permettrait de	Donner quelques bénéfices

On peut également décrire le produit en le positionnant par rapport à certains critères (pour?, Différence avec les autres produits…). L'objectif de déterminer la position du produit est de comprendre rapidement quelle est la solution proposée. (rappel, Exercice de l'ascenseur de Geoffry Moore)

1. For (target customers)
2. Who are dissatisfied with (the current market alternative)
3. Our product is a (new product category)
4. That provides (key problem-solving capability).
5. Unlike (the product alternative),
6. Our product (describe the key product features).

En français :

Pour	Public cible concernée par l'outil, le produit
Qui	Leur rôle général
Nom du produit	Ce que c'est : outil, application, système
Qui permet	Utilité
À la différence de	Pratique actuelle, concurrence
Notre produit	Ce qu'il permet de faire

Identifier les utilisateurs (rôles) :

Comme dans toute méthode de développement, il est important de déterminer les utilisateurs du système (ou du produit), pour identifier les utilisateurs appelés également ***acteur***s on se pose les questions suivantes :

- Qui fournira, supprimera, utilisera les informations de l'application?
- Qui utilisera le logiciel?
- Qui est intéressé par une fonction ou un service proposé?
- Qui assurera le support et la maintenance du système?
- Avec quels autres systèmes doit interagir le logiciel à développer ?
- On peut définir un acteur par le nom, la description, sa fréquence d'utilisation du produit, son niveau en utilisation des applications informatique, se critères de satisfaction du produit.)

Les features , définition

Une *feature* est un service fourni par le système, observable de l'extérieur qui répond à un besoin et dont la description se situe à un niveau tel que toutes les parties prenantes comprennent facilement ce dont il s'agit.

Décomposer en user-stories

Comment passer de la vision aux stories ?

À essayer

1. Travailler en équipe

À éviter

1. Avoir trop de stories au début
2. Avoir des stories trop grosses.

Construire le backlog du produit :

Pour produire un backlog initial, il est important de prioriser chacune des stories qui sera contenue dans le backlog.

Prioriser les stories : La méthode Moscow

La technique utilisée pour prioriser les besoins dans un contexte itératif est celle de MoSCoW. L'avantage de la méthode *MoSCoW* réside dans la signification de l'acronyme, qui est plus compréhensible que d'autres techniques de priorisation comme élevé/moyen/faible

M pour **Must Have** : ***DOIT*** être fait. L'exigence est essentielle. Si elle n'est pas faîte le projet échoue. On peut dire également priorité haute.

S pour Should **Have** : Il s'agit d'une exigence essentielle, qu'il faut faire dans la mesure du possible (***DEVRAIT***). Mais si elle n'est pas faîte, on peut la contourner et la livrer plus tard.

C pour **Could Have**: Il s'agit d'une exigence souhaitable. Elle ***POURRAIT être*** faîte dans la mesure où elle n'a pas d'impact sur les autres tâches

W pour **Won't Have** Il s'agit d'une exigence «Luxe». ***NE SERA PAS*** faîte cette fois mais plus tard, mais intéressante et à garder pour la prochaine version

La méthode priority Poker (pour les feature)

- Chaque participant reçoit un lot de neuf cartes numérotées de 1 à 9
- Chaque story est étudiée successivement
- Le premier vote porte sur l'intérêt d'avoir le feature. Chaque participant vote avec une carte. On fait le total des points.

- Le deuxième vote, porte sur la pénalité de ne pas avoir le feature dans le produit. On vote également de 1 à 9.
- On définit l'importance des deux votes. On peut donner 4 au premier et 1 au deuxième.
- En faisant la somme des deux votes pondérés on obtient l'utilité de l'élément. Les stories les plus utiles sont les plus prioritaires.

Exemple de backlog initial, pour e jeu Médiéval

Scénario ou story	**Priorité**
En tant qu'administrateur, je dois pouvoir ajouter des items (armes, armures, potions) pour le jeu « Médiéval »	M
En tant que joueur je dois pouvoir acheter des items	M
En tant que joueur, je dois pouvoir consulter mon historique d'achat	M
En tant qu'administrateur, je dois pouvoir ajouter, supprimer les comptes de joueurs	S

Estimer les stories … comment ?

En demandant à un expert, c'est bien mais le problème c'est qu'il va se basé sur son expérience, sa capacité. Dans les approches agiles, le développement se fait en équipe, il est difficile parfois d'avoir un expert dans l'équipe. L'avantage de demander à un expert est que ce n'est pas long et on a une idée de l'ampleur de la tâche. Un peu comme demander à Usain Bolt le temps à mettre pour courir le 100 m.

Par analogie avec ce que vous connaissez : c'est un peu ce que nous faisons d'habitudes. On dira qu'une story est un peu plus longue que la story précédente. Ou qu'une story A prendra deux fois plus de temps qu'une story B..

Découper ou détailler la story : découper la story en petites fonctionnalités que nous pouvons estimer. C'est ce qui es suggéré dans un backlog détaillé. C'est plus simple de dire qu'une fonctionnalité ou une petite story sera implémentée dans 3 jours que de dire que la story sera implémentée dans 102 jours ??

Estimer les stories : La planning Poker

Chaque story du backlog doit être estimée si on veut en tenir compte dans la planification. Les stories ne sont pas toutes de même taille; ce qui veut dire : pour planifier son projet, donc des sprints, on ne se basera pas sur le nombre de stories à faire dans le sprint. Exemple, le sprint 2 pourrait contenir une seule story alors que le sprint 3 va contenir 3 stories.

Dans SCRUM, l'estimation se fait de manière collective et la technique fréquemment utilisée est celle du planning poker

De nombreux jeux de cartes sont vendus sur internet ou fournis lors de conférences.

- Une valeur de 0 représente une fonctionnalité déjà mise en place, ou ne demandant pas d'effort particulier.
- Les valeurs 0.5 et 1 peuvent être utilisées pour des tâches particulièrement simples,
- 3 et 5 pour des tâches un peu plus complexes.
- Au-delà, la réalisation demande des travaux plus conséquents ou plus complexes.
- La valeur 100 représente une grande complexité et un grand nombre d'heures de travail. Il faudra sans doute découper
- La carte portant le symbole « ∞ » (infini) représente toute tâche valant plus de 100. Il s'agit donc d'un travail particulièrement long ou complexe nécessitant une attention particulière. Généralement, une fonctionnalité recevant cette note a peu de chance de pouvoir être embarquée dans le sprint à venir.

Ces chiffres complètent les premiers éléments d'une suite Fibonacci.

Source de l'image https://www.scrumdesk.com/agile-estimation-principles/planning-poker-cards/

Déroulement d'un planning Poker:

Chaque participant reçoit un jeu de cartes.

Sur chaque carte, il y a une valeur positive pour estimer la story

1. Le PO présente la story.
2. Les membre de l'équipe posent des questions pour clarifier la story.
3. Tous les participants présentent en même temps la carte choisie pour l'estimation
4. L'équipe discute des différences éventuelles entre les estimations.
5. On recommence jusqu'à une convergence des estimations.
6. On passe à la prochaine story.

Pour commencer la séance de planification, il suffit de choisir une story connue de TOUS , (baseline) pour laquelle l'équipe décide en commun de lui fixer une valeur. Et, il est préférable e choisir une story de taille moyenne (3 ou 5) pour laisser une marge vers le bas et vers le haut

Lors de l'estimation d'une story il s'agit de déterminer le nombre de points qu'elle nécessite pour sa réalisation. Il ne s'agit pas de dire si elle est complexe ou non. Au fait est-ce qu'une story complexe av avoir plus de points qu'une story simple ? la réponse est ça dépend

L'exemple de Mike Cohen(un des contributeur à SCRUM) à propos de l'estimation des user-stories est le suivant :

Prenez une équipe composée d'un chirurgien du cerveau et d'un enfant. Le backlog de produit contient les stories suivantes :

0- En tant qu'enfant, je dois coller 1000 timbres sur 1000 enveloppes
1- En tant que chirurgien, je dois faire une opération simple au cerveau.

Quelle est la story qui nécessite le plus de points ?

Il est probable que les deux stories se terminent en même temps. Que les deux stories aient besoin d'un même nombre de points pour se terminer.

Mais, si la question est : Quelle est la story la plus complexe? La réponse et claire : celle du chirurgien

Estimation en effort ou en temps :

Vous pouvez estimer les user-stories, en effort. Dans ce cas il faudra déterminer c'est quoi l'effort.

Exemple :

1 Effort = 5jours/homme :

A une durée de 5 jours si le nombre de ressources qui travaillent en parallèle et à temps plein est égal à 1.

A une durée de 10 jours si le nombre de ressources qui travaillent en parallèle et à mi-temps est égal à 1.

A une durée de 1 jours si le nombre de ressources qui travaillent en parallèle et à temps plein est égal à 5

Conclusion pour le backlog

Ce qu'il faut faire :

- Cultiver le backlog : la backlog évolue dans le temps, il faudra le mettre à jour.
- Partager le backlog avec toute l'équipe
- Surveiller la taille du backlog : ne pas avoir plus de 150 éléments à faire dans le backlog

À Éviter :

- D'avoir plusieurs backlog pour le même de produit
- De ne pas avoir de backlog
- De confondre le backlog de produit avec le backlog de sprint

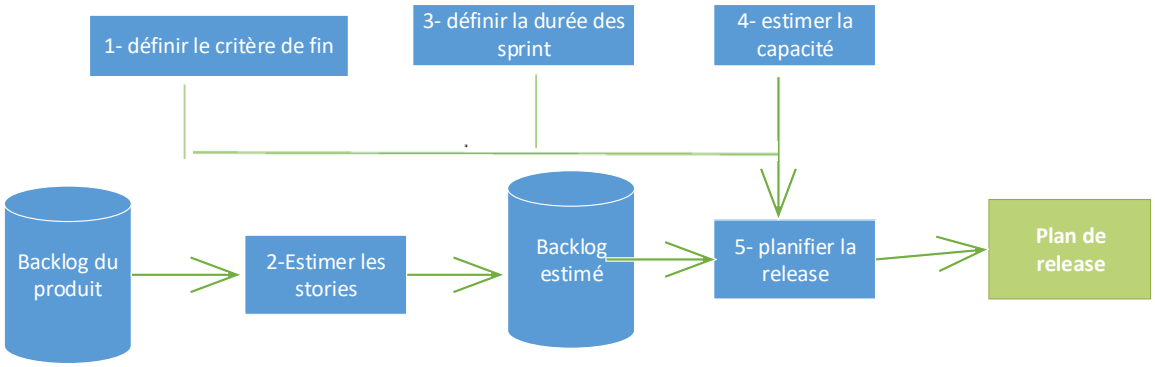
Exemple de backlog détaillé.

Scénario ou story	Tests d'acceptation	Niveau détaillé	Priorité	Effort, estimation, pointage

En tant que candidat potentiel pour un poste chez l'entreprise SigmaPlus, je veux pouvoir créer mon profil pour pouvoir postuler à des postes disponibles.	Tous les champs du formulaire ont été validés. Les informations sont enregistrées. Un courriel de confirmation est envoyé au candidat.	Créer le formulaire d'inscription chez l'entreprise SigmaPlus avec tous les champs : en d'autres mots créer le profil du candidat	M	4
		Indiquer les champs obligatoires(validation)	M	1
		Faire les validations sur le formulaire puis l'enregistrer	M	5
		Envoyer une confirmation par courriel au candidat	M	2

Planifier la release

Une release est une séquence de sprints, mais quand finit-elle? Comment la planifier pour livrer un produit fini et à temps ? Combien de sprints la release contiendra-t-elle ? Quelle est la durée d'un sprint ? Pour planifier une release, nous allons procéder par étapes.



Étape 1 : définir le critère de fin de la release :

1- Finir quand le backlog est vide. Mais le backlog est vivant. Il évolue avec le temps. Il est difficile de figer le backlog quand une amélioration est possible.

   Une variante serait de fixer un sous-ensemble du backlog pour la release. Comme les éléments sont rangés par priorité, il suffit de fixer une limite. « Pour la release courante on livre le sous-ensemble sélectionné, le reste des éléments du backlog seraient dans la prochaine version de la release ». C'est-ce qu'on appelle une release à paramètres fixés. Mais, nous avons toujours le même souci : le backlog est vivant. Il serait stupide de figer le backlog en refusant un changement qui apporte une valeur.

2- Fixer la date de fin à l'avance : la meilleure façon de procéder est de fixer une date de fin et de s'y tenir. L'objectif d'une release à date fixée est d'estimer le contenu qui sera livré à cette date. La release à date fixe présente les avantages suivants :
   a. Elle donne un objectif précis pas lointain ---> motivation de l'équipe
   b. Elle impose au Product Owner d'avoir une réflexion poussée sur les priorités des éléments du backlog
   c. Des éléments du backlog ayant peu d'intérêt ne seront pas développés.
   d. On passe moins de temps à planifier puisque la date de livraison est connue.

   Une release à date fixée et à durée uniforme (exemple une durée de 3 mois) est la formule la plus facile à mettre en œuvre.

Étape 2 : estimer les stories du backlog : La planning Pocker (voir page 37)

Étape 3 : définir la durée des sprints :

Lorsqu'on lance utilise une approche agile pour le développement logiciel, la question fondamentale est de déterminer la durée d'un sprint. Chaque projet est différent. Il n'y a donc pas de réponse universelle.

Pour SCRUM, la pratique est de faire des sprints de maximum UN MOIS. Dans la plupart des cas, des sprints de 2 à 3 semaines sont recommandés.

Pour définir la durée d'un sprint, il faudra tenir compte des facteurs suivants :

- L'implication des clients et des utilisateurs : Il faut tenir compte de leur disponibilité à utiliser les versions partielles produites à la fin de chaque sprint

- Le coût supplémentaire engendré par la préparation du sprint : Un sprint ajoute du travail supplémentaire pour préparer le produit partiel. Faire les tests de non régression, préparer la démonstration pour la revue de sprint

- La taille de l'équipe : plus il y a du monde dans l'équipe, plus il faudra du temps pour se synchroniser.

- La date de fin de la release : idéalement une release comporte au moins quatre sprints pour profiter des bénéfices des avantages de l'itératif
- Un sprint trop long risque de démotiver l'équipe
- La stabilité de l'architecture: il est plus facile d'obtenir un produit qui fonctionne si l'architecture est stable.

Étape 4 : estimer la capacité de l'équipe

La Vélocité: la vélocité de l'équipe **mesure** la partie du backlog **réalisée** par l'équipe à l'intérieur d'un sprint. À la fin d'un sprint, on mesure ce que l'équipe a été capable de réaliser. La vélocité se calcule à la fin d'un sprint après la démonstration et lors de lors de la revue de sprint

La vélocité est volatile, elle peut varier sensiblement entre les sprints.

Si un projet vient de commencer et que l'équipe n'a pas de sprint passé, une façon de faire pour déterminer la vélocité (une mesure) est de simuler une planification du premier sprint.

La vélocité est une mesure de l'équipe et non d'une personne individuelle.

La capacité est une prévision de ce que l'équipe est capable de faire en tenant compte de sa vélocité.

Exemple de prévision: Trois stories sont étudiées qui avaient été estimées à 3, 2 et 5 points. Les tâches identifiées pour ses stories sont estimées à 60 heures.

L'équipe dispose de 300 heures pour le sprint

En 60 heures, l'équipe pourrait réaliser 10 points. (3 +2 +5)

En 300 heures (sprint) l'équipe pourrait réaliser (300/ 60) * 10 points.

La capacité de l'équipe serait 50 points (estimation)

Étape 5 : produire un plan de release

Après avoir fait les étapes précédentes produire un plan de release devient facile, un jeu d'enfant. On procède comme suit :

1. On prend le backlog du produit priorisé et estimé.
2. On commence par le premier sprint de la release. On y associe les stories en commençant par les prioritaires
3. On continue dans ce sprint en additionnant la taille en points de stories jusqu'à atteindre la capacité de l'équipe
4. Quand on y arrive, on passe au sprint suivant.

Si on ne tombe pas exactement sur la capacité de l'équipe dans un sprint en ajoutant une story, il faut décider : Est-ce qu'on va au-dessus ou en en dessous de la capacité de l'équipe ou prendre une story moins prioritaire mais qui se rapproche le plus de la capacité de l'équipe.

Garder du « LOUSSE » ou du mou pour les incertitudes.

Dans le plan de release, tout est basé sur l'estimation des stories du backlog. Même si l'estimation est faîtes par ceux qui réalisent, même si l'estimation est collective il y a une part d'incertitude surtout au début de la release et il faudra en tenir compte.

Pour les release à date fixe, le lousse porte sur les stories.

Le sprint zéro

Qu'est-ce qu'un sprint Zéro :

Le développement agile a besoin d'un sprint de départ, qui ne se termine pas nécessairement par une livraison. D'une durée variable sert à mettre le projet sur de bons rails et d'apprendre à l'équipe de travailler ensemble. Concrètement, ce que l'on doit faire durant le sprint Zéro est :

- Partager une vision claire du projet
- Préparer l'environnement de développement
- Produire un backlog du produit estimé et priorisé
- Roder l'équipe sur le backlog initial
- Définir la posture ergonomique de l'interface
- Déterminer un plan de Release.
- Selon le contexte, travailler sur l'architecture et la BD
- S'offrir une belle rétrospective.

Références bibliographiques:

Gestion de projet agile 3e Édition , Véronique Messager Rota, Eyrolles 2011.

SCRUM : le guide pratique de la méthode agile la plus populaire., Claude Aubrey, DUNOD 2011.

http://www.aubryconseil.com/post/Le-backlog-de-produit

http://www.aubryconseil.com/post/2007/01/16/159-le-backlog-de-sprint

http://www.agiliste.fr/items/bien-demarrer-avec-scrum/

http://fr.wikipedia.org/wiki/M%C3%A9thode_MoSCoW

http://elevatorpitchessentials.com/essays/CrossingTheChasmElevatorPitchTemplate.html

https://www.nutcache.com/fr/blog/quest-ce-que-le-planning-poker-scrum/