

Exécution de Mandats

Table des matières

Avant-propos.....	4
1 Je pensais que vous vous vouliez	5
Problématique	5
Pratiques	5
Outils.....	6
SMART	6
Definition of Done (DoD)	6
Fonctionnalités	7
User Story	7
Modèle Conceptuel de Donnée (MCD)	8
Maquette	8
Diagramme d'état.....	9
Diagramme de flux	12
2 Qu'est-ce qui est le plus important ?.....	15
Problématique	15
Pratique.....	15
Outils.....	15
MoSCoW	15
Comparaison par paire	15
3 Comment faire ?.....	18
Problématique	18
Pratiques	18
Outils.....	18
Brainstorming	18
PoC.....	19
4 Vous voulez ça pour quand ?	20
Problématique	20
Pratiques	20
Outils.....	22
Planning Poker	22
Kanban	23
IceScrum	24
5 Ne me dis pas qu'on a perdu ces fichiers ?	32
Problématique	32
Pratique.....	32
Outils.....	32
Github	32
6 Est-ce que ça marche vraiment ?	36
Problématique	36
Pratiques	36
Niveau de test.....	36
Type de test	37
Stratégie de test	38
Outils.....	39
7 Qui a fait quoi ?.....	40
Problématique	40

Pratiques	40
Outils.....	40
Timesheet.html.....	40
8 Mais qui a décidé ça ? et quand ?	41
Problématique	41
Pratiques	41
9 C'est documenté où ?	42
Problématique	42
Pratiques	42
10 On en est où ?.....	43
Problématique	43
Pratiques	43
Outils.....	43
Comparaison de documents dans Word	43
11 Où est-ce que je trouve vos livrables ?	45
Problématique	45
Pratiques	45
Outils.....	45

Ce document sert de support de cours au module [ICT-431](#) « Exécuter des mandats de manière autonome dans un environnement informatique » donné au CPNV dans les filières CFC. Il ne se veut ni exhaustif ni source de vérité. L'approche consiste à identifier une série de problématiques rencontrées durant l'exécution d'un mandat, de décrire une (ou quelques) pratiques permettant d'adresser la problématique et de proposer un ou plusieurs outils (parmi beaucoup d'autres) permettant de mettre en œuvre la pratique. La majorité de ces outils sont méthodologique et ne nécessitent donc aucun outil informatique. Toutefois, certaines sections montrent comment mettre en œuvre certaines pratiques dans l'application [IceScrum](#) que nous utilisons au CPNV

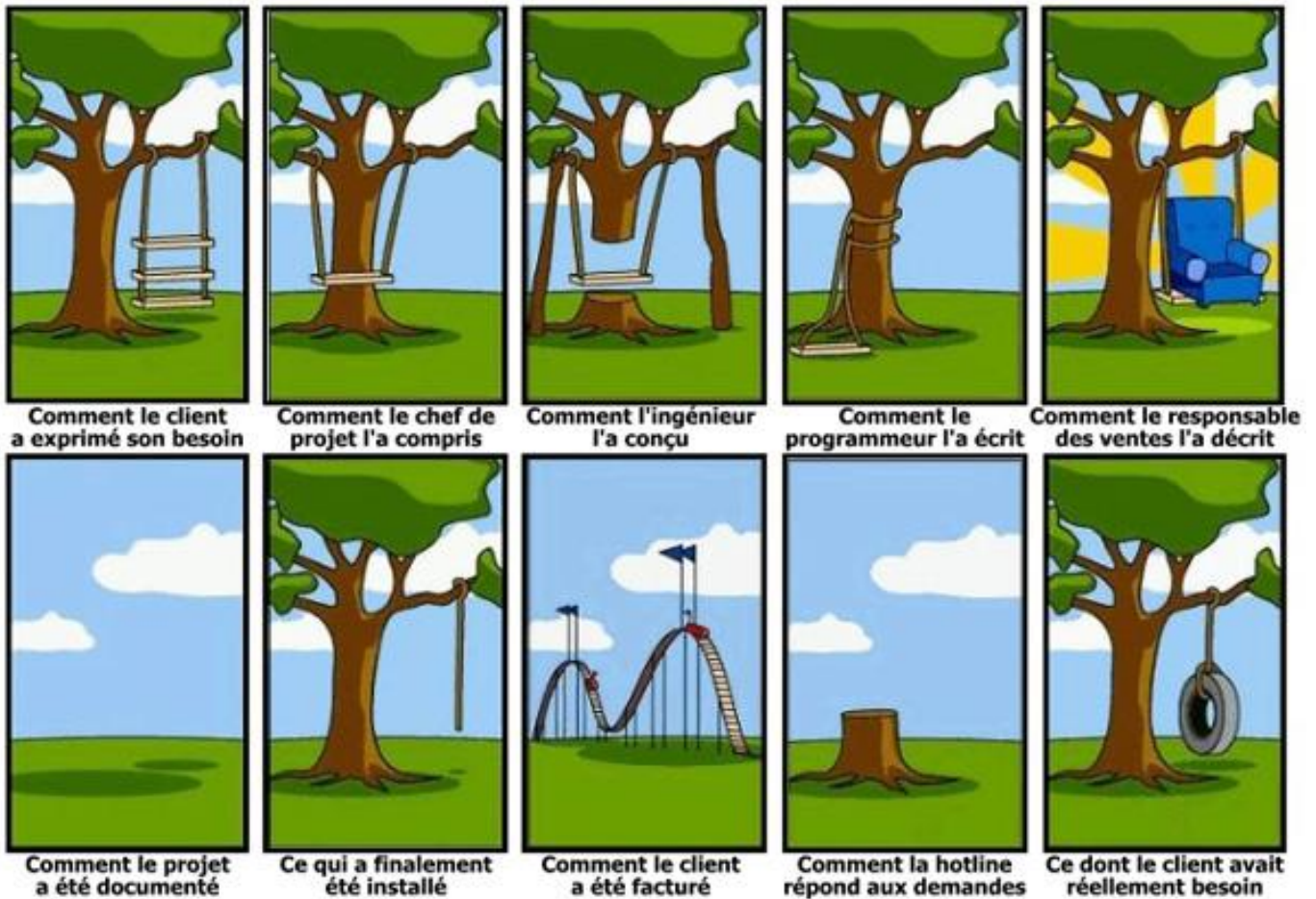
Chaque problématique fait l'objet d'un chapitre séparé

Les pratiques que nous préconisons dans la filière informatique sont basées sur les valeurs de l'agilité :

1. **Les Individus et leurs interactions** plus que les processus et les outils
2. **Des logiciels opérationnels** plus qu'une documentation exhaustive
3. **La collaboration avec les clients** plus que la négociation contractuelle (transparence)
4. **L'adaptation au changement** plus que le suivi d'un plan (adaptation)

A ce propos, nous vous invitons à (re-)lire le [Manifeste Agile](#). Il a certes été rédigé dans un cadre de développement de logiciel, mais ses valeurs et ses principes vont au-delà de ce contexte.

1 Je pensais que vous vous vouliez ...



Problématique

Avant de se mettre au travail, il est capital de s'assurer que ce que nous (mandataires) prévoyons de faire correspond aux attentes du mandant.

Comme l'illustre le gag très connu ci-dessus, la perception de ce qui doit être réalisé peut varier grandement d'une personne à l'autre. On risque donc de se retrouver face à notre mandant et commencer une phrase par 'Je pensais que ...' ou quelque chose comme ça.

Dès qu'on a articulé ces mots, on sait qu'on a un problème, parce qu'on a consacré du temps et de l'énergie à faire quelque chose que le mandant ne veut pas et qu'on va en jeter tout ou partie.

Pratiques

Il y a de nombreux outils qui aident à s'assurer de notre bonne compréhension du mandat. Ils reposent tous en priorité sur le dialogue avec l'utilisateur.

Selon le contexte du projet, nous n'avons pas forcément de contact direct avec l'utilisateur final. Mais nous avons toujours un interlocuteur dont le rôle est d'être ou de représenter cet utilisateur. Nous appelons cette personne le **Product Owner (PO)**.

Nous en abordons quelques-uns dans ce document. Chacun d'entre eux permet de diriger la discussion selon un certain point de vue. Ils sont donc complémentaires les uns des autres. Ils peuvent être applicables ou pas en fonction du contexte du mandat.

Chacun produit des traces tangibles qui serviront de référence en cours de réalisation du mandat.

Il s'agit de :

- SMART, une checklist simple qui permet d'inspecter la bonne formulation d'un objectif
- La liste de fonctionnalités (ou « features ») permet de définir les cas d'utilisation du système dans les grandes lignes
- Les User Story permettent de préciser le comportement du système
- Le modèle conceptuel de données (MCD) permet de s'assurer que l'on a pris en compte toutes (et rien que) les données à gérer
- Les maquettes, parce qu'une image vaut mille mots
- Le diagramme d'état permet d'explicitier ce cycle de vie d'une certaine entité à l'intérieur

Cette liste est naturellement non-exhaustive. Il existe nombre d'autres outils qui fonctionnent tout aussi bien – voire mieux selon le contexte.

N'excluez donc jamais un outil proposé sous prétexte qu'il ne figure pas dans cette liste.

Outils

SMART

SMART est un acronyme qui permet d'inspecter la formulation d'un objectif.¹

On peut dire qu'un objectif est SMART si on répond par l'affirmative aux cinq questions suivantes :

- Est-il **S**pécifique ? L'énoncé est-il clair et précis. Quand on le lit, sait-on ce que l'on doit faire ? Définit-il bien une seule chose à atteindre ou faire ?
- Est-il **M**esurable ? Est-il possible de vérifier de manière non ambiguë que la tâche est réalisée ou non ?
- Est-il **A**mbitieux ? Représente-t-il un travail et une avancée du projet significatif ?²
- Est-il **R**éaliste ? La personne en charge de l'objectif a-t-elle les moyens et les compétences pour l'effectuer ?
- Est-il **T**emporel ? Sait-on quand cet objectif doit être atteint ?

Definition of Done (DoD)

La « définition de fini » est un ensemble de critères (SMART mais sans T) qui peut être appliqués à différentes situations pour s'assurer que tout est fait. Exemple : DoD pour la mise en ordre d'une salle, que l'on pourrait appliquer aux salles de classe, aux salles de réunion et à la salle des maîtres :

- Les chaises sont sur les tables
- Les fenêtres sont fermées
- La lumière est éteinte
- La poubelle est vide

¹ SMART est un outil très répandu. Il en existe plusieurs variantes, en particulier en ce qui concerne le A. Nous en avons choisi une ici, cela ne veut pas dire que les autres variantes sont fausses.

² Une autre interprétation du A qui est répandue est « Acceptable » (selon des critères légaux ou moraux par exemple)

Fonctionnalités

Nous procédons tout d'abord à une découpe du mandat en fonctionnalités (Features) que l'on nomme parfois aussi cas d'utilisation (Use Case)

Pour ce faire, nous complétons la phrase :

« On utilise [le système] pour ... ».

Exemple : si on nous demande de réaliser un smartphone simplifié, on peut dire :

1. On utilise un SmartPhone pour **téléphoner**
2. On utilise un SmartPhone pour **envoyer et recevoir des SMS**
3. On utilise un SmartPhone pour **prendre des photos**

Nous avons ainsi identifié trois features : « Téléphoner », « Envoyer et recevoir des SMS » et « Prendre des photos ». Au passage, nous avons également établi le fait que notre smartphone simplifié ne permettra pas de surfer sur Internet, puisqu'on n'a pas listé cette feature !

User Story

Une User story est un énoncé court qui décrit quelque chose qu'un utilisateur va faire avec le système.

Il est important que – en plus de l'action elle-même – la story indique qui la fait et avec quelle intention. C'est pour cela que l'on donne généralement la structure de base de la story suivante :

En tant que ...
Je veux ...
Pour ...

Ne soyons pas dogmatiques, cette structure n'a pas besoin d'être conservée à la lettre ! Elle doit servir avant tout de rappel à ce qui doit figurer dans la story

Exemple 'à la lettre' :

En tant que responsable de département, je veux que les données de travail de tous les postes soient backupées tous les jours pour être sûr de ne rien perdre en cas de panne.

Variante 1 : Il est évident que l'intention du backup est de ne pas perdre de données, il n'est donc pas indispensable de le dire :

En tant que responsable de département, je veux que les données de travail de tous les postes soient backupées tous les jours

Variante 2 : en s'affranchissant de la forme et en introduisant une intention non-évidente

Le responsable de département veut que les données de travail de tous les postes soient backupées tous les jours pour satisfaire aux exigences ISO-9001

Un ensemble de User stories (que l'on peut également nommer « scénarios ») nous permet de détailler chaque feature.

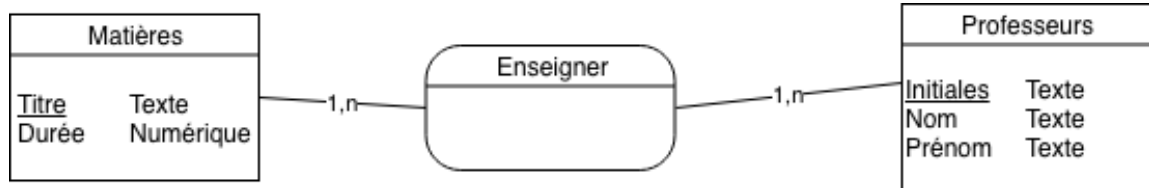
Modèle Conceptuel de Donnée (MCD)

Dans le cas d'un projet incluant la gestion de données, un MCD est un bon moyen de vérifier que l'on a bien compris l'ensemble des informations que le système va gérer.

A noter : le fait qu'il y ait des données à gérer ne signifie pas nécessairement qu'il y a une base de données !

Bref rappel sur Le MCD³ :

- Il est défini dans la langue du mandant.
- Il ne contient pas d'éléments techniques (clés primaires, tables de liaison, ...)
- Il sert à s'assurer que le système correspondra bien aux attentes du mandant
- La notation que nous utilisons est issue de la méthode Merise:



- Il est fréquemment nécessaire de donner des informations supplémentaires concernant les entités. Cela se fait de manière textuelle dans un document que l'on joint au MCD. Exemple :

Professeurs :

- Les initiales sont toujours formées de trois lettres : la première du prénom, puis la première et la dernière du nom

Matières :

- La durée est exprimée en nombre de périodes d'enseignement prévues

Le MCD – lorsqu'il y en a un – doit impérativement être validé avec le mandant. Le moment de la validation est un événement majeur du projet, qui doit absolument être consigné dans le journal de bord du projet.

Maquette

Quand on a imaginé une solution qui – on le pense – répond au besoin du mandant, il faut qu'on la lui explique pour qu'il puisse nous dire si ça lui convient ou pas.

On peut facilement se perdre dans une description textuelle d'écran/de page. Exemple « dans la page contact, on a une liste de personnes avec juste le nom et la photo (ou avatar) à droite du nom ».

Une image valant mille mots, il est parfois préférable de faire des maquettes.

³ Voir support de cours du module ICT-104 pour plus de détails



01-Accueil

02-Contacts

03-Contact

04-Appel

05-Résumé

Faire des maquettes à l'aide d'un logiciel permet d'avoir une présentation soignée et uniforme. Mais cela peut être une tâche très chronophage.

N'hésitez pas à avoir recours à des photos prises d'un tableau blanc ou à des scans de dessins faits à la main si le contexte le permet !!!

L'important est souvent d'être sûr qu'on se comprend, pas que ce soit joli.

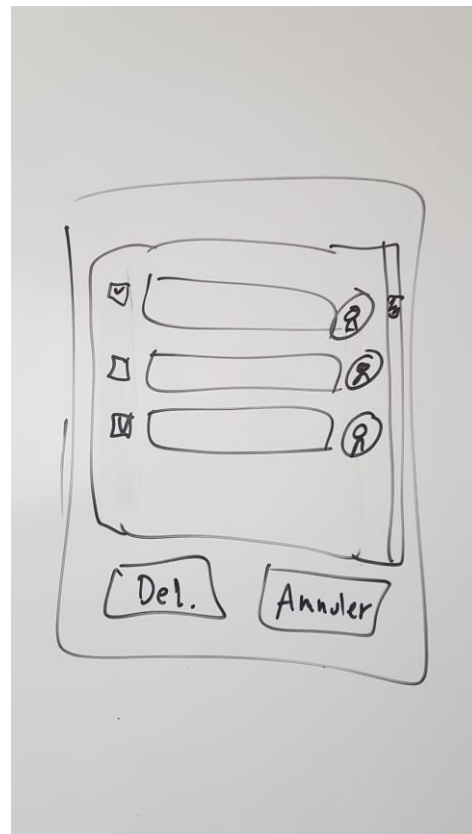
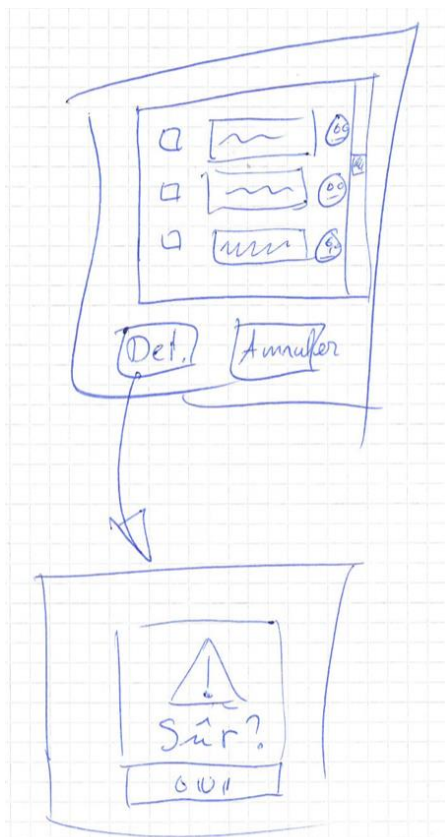


Diagramme d'état

Une entité (de notre MCD) est caractérisée par un certain nombre d'attributs. Les combinaisons de valeurs de certains attributs (pas forcément tous) définissent des États de l'entité.

Prenons par exemple un produit vendu par un site de E-commerce : un vélo.

Ses attributs (parmi d'autres) :

- Marque
- Modèle
- Couleur
- Prix de vente
- Nombre en stock
- Commandes
- Ventes
- Visible (sur le site)

Nous commençons par identifier, nommer et décrire les états sur la base des valeurs

Valeurs	Nom de l'état	Description
<ul style="list-style-type: none">• Stock= 0• Commandes = 0• Ventes = 0• Visible= false	Prévision	On prévoit de vendre ce type de vélo. On a enregistré les informations, mais on n'en pas encore commandé

Valeurs	Nom de l'état	Description
<ul style="list-style-type: none">• Stock= 0• Commandes = 20• Ventes = 0• Visible= false	Commande initiale	On a commandé des vélos au fournisseur. Le fait que le stock et les ventes sont à 0 indique qu'il s'agit d'une commande initiale

Valeurs	Nom de l'état	Description
<ul style="list-style-type: none">• Stock= 14• Visible= true	En vente	Les vélos sont en vente.

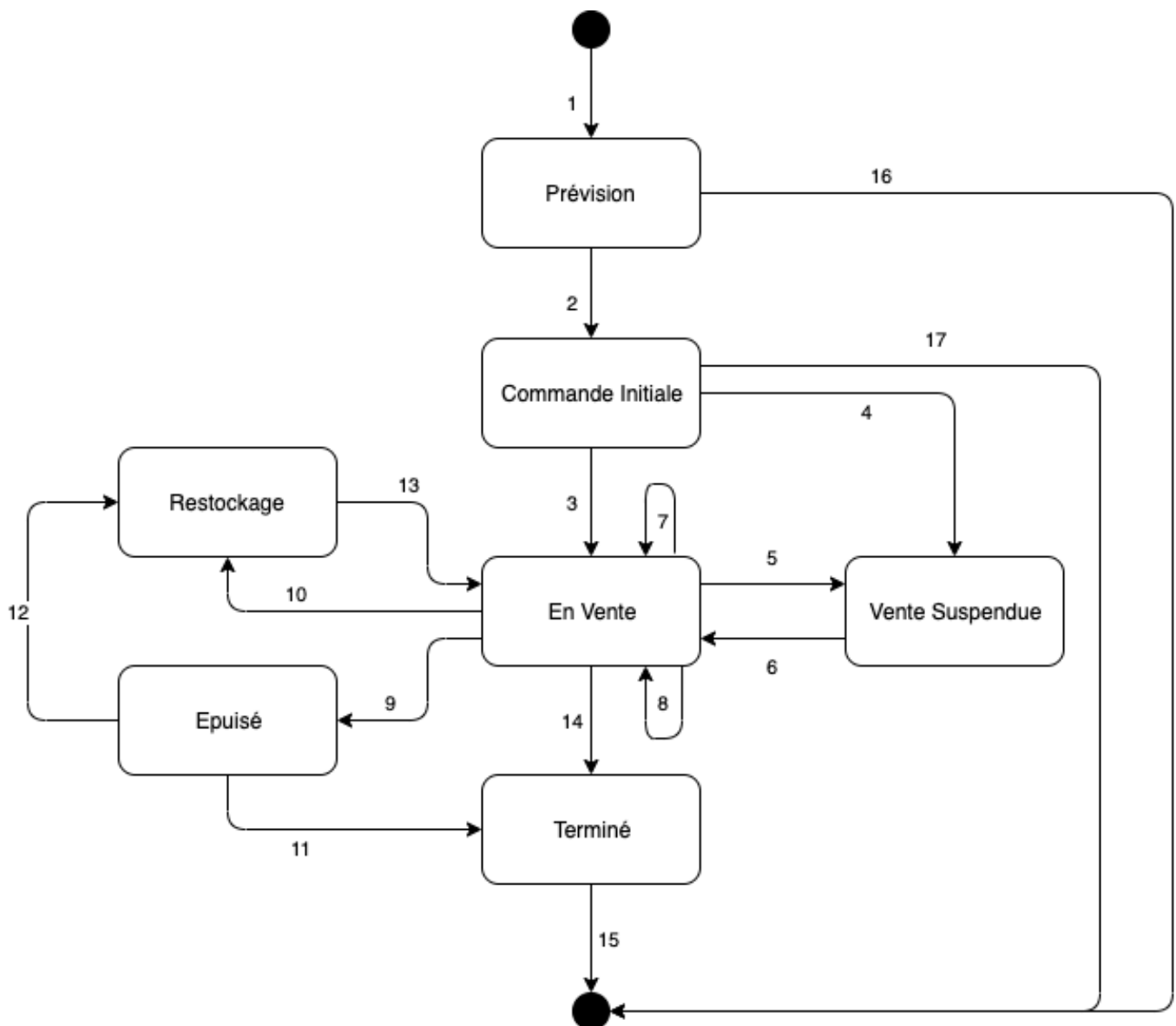
Valeurs	Nom de l'état	Description
<ul style="list-style-type: none">• Stock= 5• Visible= false	Vente suspendue	Par exemple parce qu'un défaut a été constaté sur les premiers vélos vendus

Valeurs	Nom de l'état	Description
<ul style="list-style-type: none">• Stock= 0• Commandes = 0• Visible= true	Epuisé	Momentanément épuisé

Valeurs	Nom de l'état	Description
<ul style="list-style-type: none">• Stock= 0• Commandes = 10• Visible= true	Restockage	On a commandé des vélos

Valeurs	Nom de l'état	Description
<ul style="list-style-type: none"> • Stock= 0 • Commandes = 0 • Ventes = 132 • Visible= false 	Terminé	On ne vend plus cet article

On peut ensuite placer ces états dans un diagramme état/transitions :



Les points noirs marquent les points de début et de fin du diagramme. Il y en a exactement deux.

Nous pouvons maintenant exprimer des règles métier en décrivant les transitions d'état numérotées :

Transition	A lieu parce que
1	Nous avons identifié un produit que nous voulons vendre
2	Nous avons placé une commande auprès du fabriquant

Transition	A lieu parce que
3	Nous avons reçu la commande initiale et nous avons publié l'article sur le site
4	Nous avons reçu la commande initiale, mais nous ne sommes pas prêts à mettre le produit en vente
5	Nous avons du stock, mais nous ne voulons pas vendre ce produit pour le moment, parce que ce n'est pas la saison (par exemple)
6	Nous remettons en vente le produit
7	Nous avons placé une commande de restockage préventif avant que le stock n'arrive à zéro
8	Nous avons reçu la commande de restockage
9	Nous avons vendu la dernière pièce en stock et il n'y a pas de commande en attente
10	Nous avons vendu la dernière pièce en stock et il y a déjà une commande en attente
11	Nous avons décidé de ne pas recommander ce produit et de l'abandonner
12	Nous avons placé une nouvelle commande
13	La commande est arrivée
14	Les ventes stagnent. Nous avons décidé d'abandonner ce produit. Nous avons liquidé le stock sans le vendre.
15	Effacement complet de ce produit (hautement improbable, car nous voulons conserver l'historique des ventes)
16	Nous avons abandonné l'idée de vendre ce produit
17	Nous sommes en litige avec le fournisseur. Nous avons annulé notre commande et nous renonçons à vendre ce produit

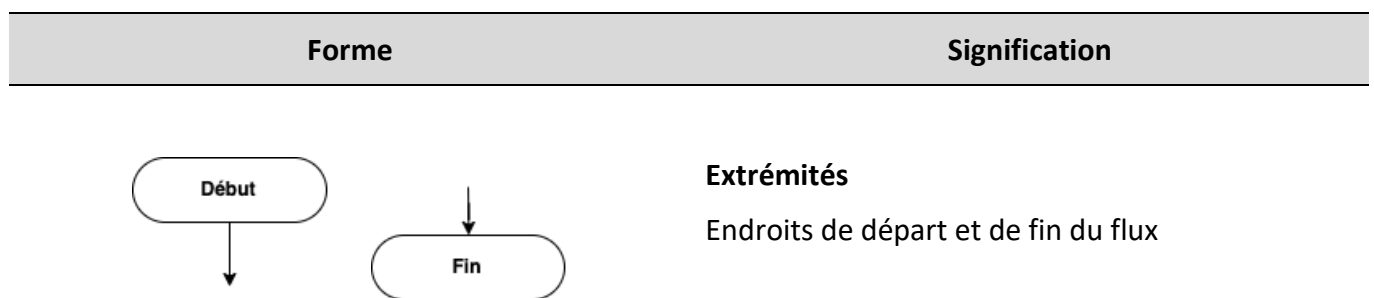
Cet outil peut faire un peu peur de prime abord. Le diagramme ci-dessus n'est pas des plus simples.

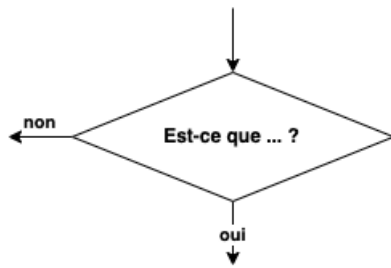
Il présente toutefois l'avantage de donner une vue globale pour une série de cas particuliers qui auraient très certainement fait surface en cours de projet.

Diagramme de flux

Le diagramme de flux est un moyen graphique permet d'illustrer les processus métiers – et par conséquent le comportement attendu d'un système informatique.

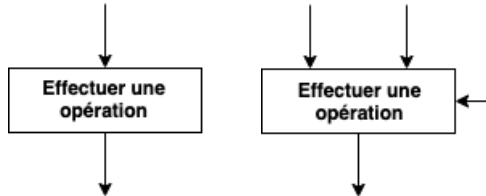
Il existe de nombreux formalismes pour faire des diagrammes de flux. Nous en utilisons une version très simple :





Condition

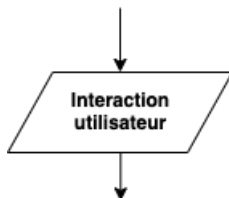
On « arrive » toujours par le haut, mais on peut « repartir » comme on veut, par n'importe quelle pointe libre



Opération

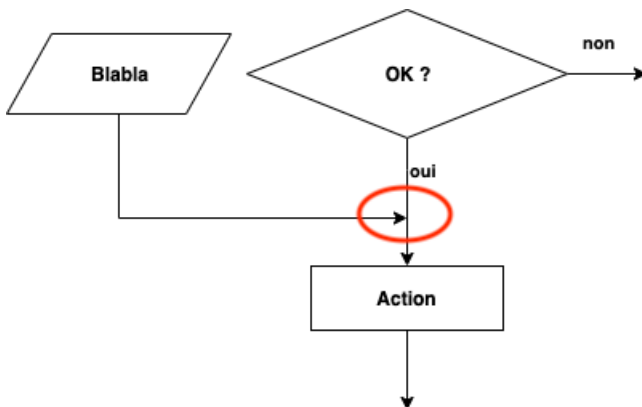
On « arrive » toujours par le haut ou par les côtés, on « repart » toujours par le bas ou par les côtés.

Si plusieurs chemins arrivent sur la même opération, on essaie au maximum de séparer les flèches.



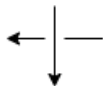
Interaction

On demande un input à l'utilisateur ou on lui fournit une information, un résultat.



Connexion

Une ligne peut en rejoindre une autre, mais la pointe de la flèche doit absolument être présente pour qu'il n'y ait pas de doute quant au chemin à suivre.

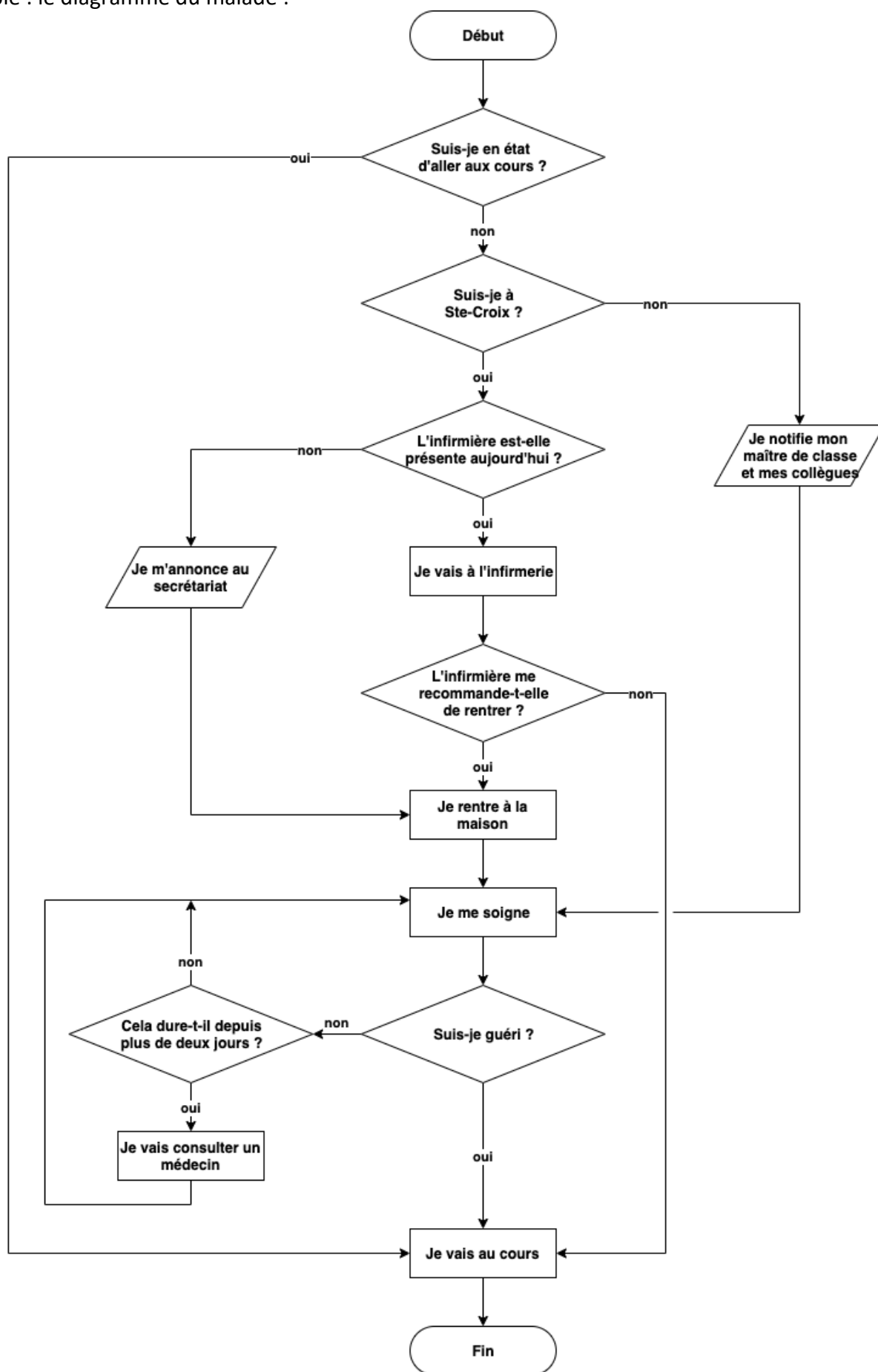


Croisement

Il faut les éviter au maximum, mais ce n'est pas toujours possible !

Dans ce cas, il faut qu'une des deux lignes soit discontinuée, à nouveau pour qu'il n'y ait pas de doute quant au chemin à suivre.

Exemple : le diagramme du malade :



2 Qu'est-ce qui est le plus important ?

Une fois que l'on est d'accord sur les attentes du mandant, nous pouvons procéder à une analyse technique pour déterminer l'ensemble des choses à faire pour les satisfaire.

Cette analyse dépend fortement de la nature du mandat et ne fait par conséquent pas l'objet d'un chapitre de ce document.

Mais maintenant que nous avons une liste de chose à faire se pose la question : y en a-t-il de plus prioritaires que d'autres ?

Problématique

Notre mandant aimerait bien sûr tout avoir tout de suite, c'est naturel.

Mais nous, nous ne pourrons pas tout faire simultanément d'une part et d'autre part certaines choses ne peuvent être faites sans que d'autres n'aient été réalisées au préalable. Si certaines priorités sont évidentes et ne nécessitent pas de long débat (il faut construire les murs d'une maison avant son toit), il peut y en avoir qui ne vont pas de soi ou qui sont sujettes à interprétation.

Pratique

Les priorités sont généralement définies par le mandant, mais ce n'est pas obligatoire (rappelez-vous qu'il veut tout tout de suite).

Elles peuvent donc également être proposées par le mandataire, mais elles doivent alors être validées par le mandant.

Outils

MoSCoW

Une approche classique consiste à attribuer une des quatre lettres M, S, C ou W à chaque tâche selon l'échelle suivante

- **M**ust = Vital
- **S**hould = Essentiel
- **C**ould = Confort
- **W**ould = Luxe

Comparaison par paire

On est parfois confronté à une grande liste d'éléments complexes, c'est-à-dire qui peuvent être comparés selon plusieurs critères. Le fait d'avoir beaucoup de critères pour beaucoup d'éléments rend la tâche très difficile et peut mener à une situation où on « tourne en rond » :

- A est plus important que B
- B est plus important que C
- C est plus important que D
- Oui mais ... je trouve D plus important que A !

L'outil de comparaison par paire permet de poser des questions ciblées qui vont vous permettre d'éliminer une partie de la complexité et ainsi briser le cercle infernal.

Prenons par exemple une liste de villes que vous aimeriez visiter dans votre vie :

- Paris
- New York

- Lucerne
- Berlin
- Lisbonne
- Rome
- Jérusalem
- Marrakech
- Tokyo
- Pékin
- Buenos Aires

Vous placez ces villes dans un « demi-tableau » :

	Buenos Aires	Pékin	Tokyo	Marrakech	Jérusalem	Rome	Lisbonne	Berlin	Lucerne	New York
Paris										
New York										
Lucerne										
Berlin										
Lisbonne										
Rome										
Jérusalem										
Marrakech										
Tokyo										
Pékin										
Buenos Aires										

Attention à l'ordre ! On Remplit les en-têtes de colonnes en reprenant les en-têtes de ligne à partir de la dernière ligne et en remontant.

Il faut ensuite remplir chaque case du tableau en ne prenant en considération que deux villes à la fois, en se posant la question « Plutôt ... que ... ? » :

	Buenos Aires	Pékin	Tokyo	Marrakech	Jérusalem	Rome	Lisbonne	Berlin	Lucerne	New York
Paris	Oui	Non	Oui	Oui	Oui	Non	Oui	Oui	Non	Oui
New York	Oui	Oui	Oui	Oui	Non	Oui	Oui	Oui	Oui	
Lucerne	Oui	Oui	Oui	Oui	Oui	Non	Oui	Oui		
Berlin	Oui	Non	Non	Non	Oui	Non	Oui			
Lisbonne	Non	Non	Oui	Non	Non	Non				
Rome	Non	Non	Oui	Non	Non					
Jérusalem	Oui	Non	Non	Non						
Marrakech	Non	Non	Non							
Tokyo	Non	Non								
Pékin	Oui									
Buenos Aires										

Réponse à la question:
« Plutôt Rome que Tokyo ? »

Réponse à la question:
« Plutôt Tokyo que Pékin ? »

Une fois qu'on a terminé, on compte les points :

Points		Buenos Aires	Pékin	Tokyo	Marrakech	Jérusalem	Rome	Lisbonne	Berlin	Lucerne	New York
7	Paris	Oui	Non	Oui	Oui	Oui	Non	Oui	Oui	Non	Oui
8	New York	Oui	Oui	Oui	Oui	Non	Oui	Oui	Oui	Oui	
8	Lucerne	Oui	Oui	Oui	Oui	Oui	Non	Oui	Oui		
3	Berlin	Oui	Non	Non	Non	Oui	Non	Oui			
1	Lisbonne	Non	Non	Oui	Non	Non	Non				
5	Rome	Non	Non	Oui	Non	Non					
4	Jérusalem	Oui	Non	Non	Non						
4	Marrakech	Non	Non	Non							
3	Tokyo	Non	Non								
8	Pékin	Oui									
4	Buenos Aires										
Total											

3 « Non » dans la colonne donnent 3 points à Jérusalem

1 « Oui » dans la ligne donne 1 point à Jérusalem

Il ne reste plus qu'à classer par nombre de points. Bien sûr on va avoir des égalités (New York, Lucerne et Pékin) et il faudra peut-être les départager, mais on aura déjà bien restreint le nombre de possibilités.

3 Comment faire ?

Problématique

Le mandant a explicité son but et nous pensons en avoir une bonne compréhension générale. Il ne va pas tarder à nous poser LA question : « Alors, vous pouvez me faire ça pour quand ? »

Le problème c'est que pour pouvoir commencer à répondre à cette question, il faut que l'on se fasse nous-même une idée de comment nous comptons nous y prendre.

Pratiques

Nous procédons à une **Analyse Préliminaire**.

Elle est indispensable et son résultat est généralement un bref rapport décrivant :

1. Le **cadre** dans lequel le mandat est réalisé. On décrira ce cadre à partir de multiples points de vue :
 - **Humain** : Qui sont les personnes concernées ? Combien sont-elles ? Quel est leur degré de familiarité avec l'informatique ? ...
 - **Géographique** : Où sont les utilisateurs ? Y a-t-il plusieurs sites ? Les utilisateurs seront-ils mobiles ? ...
 - **Technique** : Quelle est l'infrastructure existante ? Quelles modifications pensons-nous devoir y apporter ? Quels projets ont été réalisés précédemment dans le même contexte ? ...
 - **Concurrentiel** : Est-ce qu'il existe quelque chose de similaire ? Quels avantages notre projet pourrait donner à nos utilisateurs ? ...
 - **Budgétaire** : Quels sont les moyens financiers à disposition ? D'où proviennent-ils ? ...
 - ...
2. La ou les **technologies** que l'on pense utiliser pour réaliser le mandat. Nous élaborons une liste précise (inclus les versions) et nous expliquons pourquoi nous avons fait ces choix
3. Une **planification initiale** qui définit grossièrement les moments clés du projet que l'on est capable d'identifier à ce stade précoce.
4. Une **Analyse de risque**, dans laquelle liste un maximum de risques potentiels en détaillant :
 - Le risque lui-même : Mon PC est victime d'une attaque de Ransomware
 - La probabilité que le risque se réalise : moyenne
 - L'impact sur le projet si le risque se réalise : le projet est gravement retardé car je dois tout recommencer.
 - Le ou les moyens d'empêcher le risque de se produire et le coût que cela engendrerait : mettre mes données sous contrôle de version (Github), ça me prend un tout petit peu de temps.
 - Ce qu'on a décidé de faire par rapport à ce risque : J'utilise Github
5. Une **stratégie de test** comme décrit plus loin dans ce document

Outils

Brainstorming

Le brainstorming peut être utile dans beaucoup de situations diverses et variées. Il a pour but de générer des idées nouvelles. Dans le cas où on analyse les moyens de réaliser notre mandat, il peut s'avérer intéressant pour éviter de foncer tête baissée dans une voie qui nous semble évidente, mais qui pourrait au final ne pas être la meilleure.

Le brainstorming est un outil très répandu et peut prendre des formes différentes d'une organisation à l'autre.

Dans le cadre de ce cours, nous le définissons comme suit :

- C'est une réunion d'au moins trois personnes et d'au plus huit personnes
- Une personne a le rôle d'animateur, une autre a le rôle de scribe
- Le brainstorming porte sur un seul sujet
- Il a une durée courte : maximum 10 minutes
- L'animateur est responsable :
 - De faire respecter la durée
 - De maintenir la discussion sur le sujet
- Le scribe est responsable de noter les idées retenues
- Son déroulement est :
 - Une première personne émet une idée ou un commentaire
 - Une autre personne rebondit sur cette idée, l'enrichit, la pousse plus loin
 - Il est interdit de contredire ou d'objecter. Il est interdit de dire non.
 - Une seule personne parle à la fois
 - Lorsque la discussion se tarit, l'animateur indique au scribe s'il faut noter quelque chose ou pas et relance la discussion en invitant chacun-e à proposer une autre idée
- Le seul produit du meeting est la liste des idées notées par le scribe

PoC

De l'anglais "Proof of Concept", un PoC est quelque chose que l'on réalise dans le but de démontrer la faisabilité.

Un PoC peut typiquement s'avérer nécessaire dans l'établissement de la liste des technologies présentée dans l'analyse préliminaire.

Attention, il s'agit d'une réalisation, pas d'une étude théorique ! Faire un PoC peut prendre du temps et nécessiter des moyens techniques supplémentaires

4 Vous voulez ça pour quand ?

Problématique

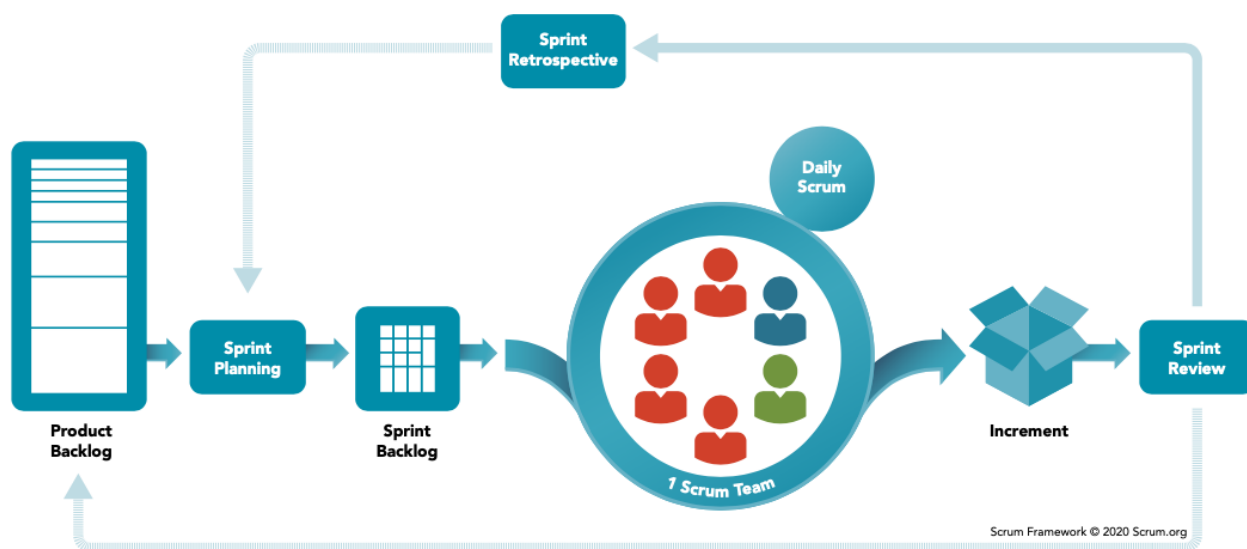
Le mandant nous a maintenant expliqué ce qu'il voulait et quelles étaient ses priorités. Nous nous sommes préparés la moindre et nous allons pouvoir nous attaquer à la grande question de la planification.

C'est un moment difficile car nous sommes face à beaucoup d'inconnues et il est donc compliqué de pouvoir dire avec confiance quand nous aurons terminé. Et pourtant c'est ce que le mandant attend de nous...

Pratiques

Nos pratiques sont tirées de SCRUM, lui-même directement inspiré des principes de l'agilité.

SCRUM FRAMEWORK



SCRUM peut être résumé par la définition du terme « Sprint », de trois artefacts et de cinq cérémonies.

Le Sprint

Un sprint est un intervalle de temps court (1 mois maximum, souvent appelé itération), pendant lequel on va concevoir, réaliser et tester un nombre généralement petit de stories pour produire un incrément produit.

Les artefacts

Commençons par définir le terme : un artefact est un « produit de l'art ou de l'industrie humaine »⁴. Il s'agit donc de quelque chose de concret, que nous produisons durant le projet.

Les trois artefacts sont :

⁴ <https://dictionnaire.lerobert.com/definition/artefact>

1. **Le Product Backlog** : (ou « backlog » tout court) : l'ensemble des stories que le PO veut voir réalisées, mais qui n'ont pas encore été planifiées
2. **Le Sprint Backlog** : l'ensemble des stories que l'on doit encore réaliser dans le cadre d'un sprint
3. **L'incrément produit** : l'ensemble des stories que l'on a réalisées dans le cadre d'un sprint

Bien qu'il ne figure pas dans le framework SCRUM, nous utilisons un quatrième artefact :

- La **SandBox** : il s'agit de user stories proposées par le PO ou par nous, mais qui ne sont pas encore acceptées par le PO comme devant être réalisées

Les cérémonies

Un mot un peu pompeux pour désigner ce qui est en fait simplement une réunion.

Les cinq cérémonies sont :

1. **Le Backlog Grooming** : Réunion de travail pendant laquelle on passe en revue les éléments du backlog pour s'assurer qu'il contient les éléments appropriés, priorisés, et que les éléments en haut du backlog sont prêts à être planifiés.
2. **Le Sprint Planning** : Réunion de travail durant laquelle on construit le sprint backlog en choisissant les stories que l'on va réaliser durant le prochain sprint. Pour ce faire, on se base sur :
 - Le product backlog
 - Les estimations de charge
 - Les priorités établies précédemment (s'il y en a)
 - Les inputs du PO
 - L'expérience accumulée durant les sprints précédents
3. **La Sprint Review** : Réunion de travail consistant à présenter aux parties prenantes les stories terminées au cours du Sprint afin de recueillir leurs feedbacks et à faire le point sur l'avancement global du projet.
4. **La Sprint Retrospective** : Réunion de travail durant laquelle on porte un regard critique sur notre manière de travailler et – le cas échéant – on décide de changements pour l'améliorer.
5. **Le Daily Scrum** : La « mêlée » quotidienne. Il s'agit d'une réunion particulièrement courte, mais néanmoins essentielle. Durant la réunion, chaque personne communique :
 - Ce qu'elle a fait le jour précédent
 - Ce qu'elle compte faire aujourd'hui
 - Les problèmes bloquants auxquels elle est confrontée (s'il y en a)

Le daily scrum doit rester bref pour laisser le maximum de temps pour travailler. Il est fréquemment tenu debout (« stand-up meeting »). Il faut faire particulièrement attention à ne pas commencer à débattre ou à résoudre les problèmes. Si un intervenant a une idée de solution à un problème présenté par un autre, les deux conviennent de discuter après le daily scrum.

Et s'il n'y a qu'une personne qui travaille ? Eh bien on le fait quand même ! C'est un moment de réflexion, de prise de recul qui sera bénéfique à la marche du projet

Time Box

Toutes ces cérémonies sont sujettes à une même pratique : le Time Boxing. Il s'agit là avant tout d'éviter le syndrome de la réunionite⁵.

On va donc définir à l'avance la durée de la cérémonie et on va faire très attention à ne pas dépasser cette durée.

Outils

Planning Poker⁶

Il est très difficile d'estimer le temps de travail que demande la réalisation d'une tâche. Cela reste vrai quand on a des années d'expérience, ça l'est donc encore plus quand on n'en a pas. On va donc accepter notre inaptitude à chiffrer en heures et se contenter de comparer des tâches entre elles, par exemple : « J'estime que cette tâche va me prendre environ quatre fois plus de temps que celle-ci »

Le planning poker est une façon ludique de produire des estimations sur l'effort nécessaire à la réalisation de tâches.



L'avantage principal du planning poker est de permettre à tous de s'exprimer librement. L'estimation serait meilleure parce que plusieurs personnes l'auront validée : des participants avec des niveaux d'expérience et d'expertise différents. De plus, cette technique favorise les échanges entre le responsable de produits et l'équipe de développement.

L'estimation se fait en points et non en temps.

Les points permettent d'obtenir une mesure relative de l'effort car les scénarios sont comparés entre eux. L'équivalent en temps est propre à chacun, selon ses compétences, son expérience et sa connaissance du domaine. L'avantage d'utiliser des points réside surtout dans le fait que l'échelle utilisée restera stable tout au long du projet. Peu importe la vitesse (vélocité) à laquelle l'équipe de développement accomplira ces tâches, nul besoin de réviser les estimations : c'est le rapport entre le temps réel et les points qui évoluera.

La suite de Fibonacci est utilisée pour les évaluations. Comme nous cherchons un dimensionnement de l'effort, le message est clair : plus le scénario est gros, moins l'évaluation est précise. Le paquet de cartes utilisé pour le planning poker doit donc comporter les valeurs suivantes : 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144. Certains simplifient les grandes valeurs en les transformant en 20, 40, 100... puisqu'il s'agit d'être globalement bon plutôt que précisément erroné. On y ajoute généralement les valeurs 0 et 1/2.

⁵ Exemple de définition : <https://www.universalis.fr/dictionnaire/reunionite/>

⁶ Extrait de https://fr.wikipedia.org/wiki/Planning_poker

Il est préconisé de se mettre d'accord sur un scénario de référence, traditionnellement équivalent à la valeur 1 (ou 2).

Déroulement :

1. Les participants s'installent autour d'une table, placés de façon que tout le monde puisse se voir.
2. Le PO présente à l'équipe une user story.
3. Les participants posent des questions au PO, discutent du périmètre du scénario, évoquent la DoD et les tests d'acceptation.
4. Chacun des participants évalue l'effort de développement de ce scénario, choisit la carte qui correspond à son estimation et la dépose, face vers le bas, sur la table devant lui.
5. Au signal du facilitateur, les cartes sont retournées en même temps.
6. S'il n'y a pas unanimité, la discussion reprend.
7. On répète le processus d'estimation jusqu'à l'obtention de l'unanimité.

Une procédure optimisée consiste, après la première "donne", à demander aux deux acteurs ayant produit les évaluations extrêmes d'expliquer leurs points de vue respectifs. Ces explications achevées et comprises de tous, une nouvelle estimation est produite et c'est alors la moyenne arithmétique de ces estimations qui est prise en compte.

Kanban

La méthode Kanban⁷ est communément associée à l'agilité.

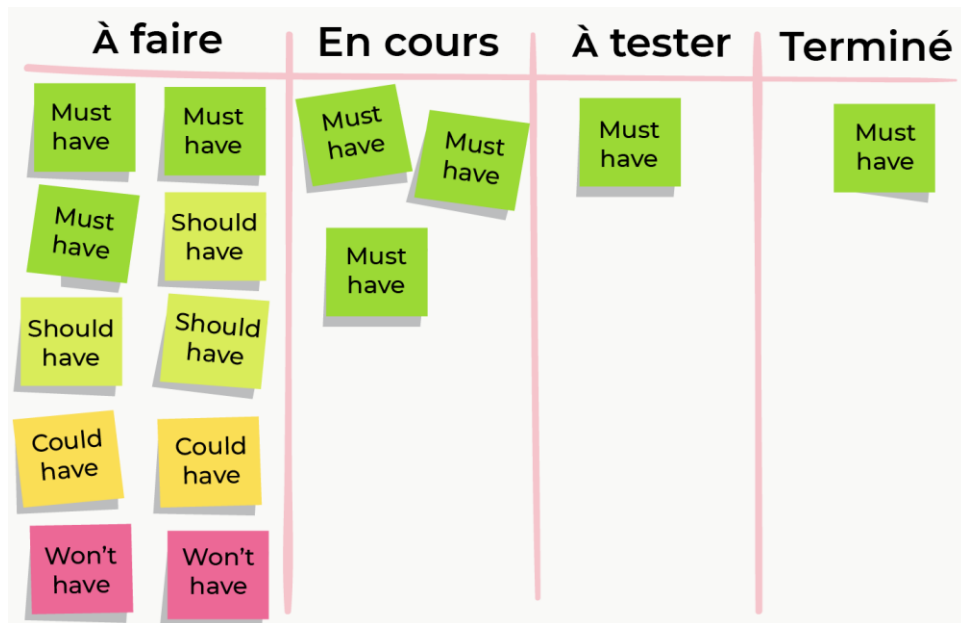
Par abus de langage, on désigne fréquemment par le terme Kanban un simple tableau qui permet le management visuel des tâches.

Le Kanban « de base » ne contient que trois colonnes :



⁷ [https://fr.wikipedia.org/wiki/Kanban_\(développement\)](https://fr.wikipedia.org/wiki/Kanban_(développement))

Mais rien n'interdit de l'adapter selon les besoins:



En veillant à ce que la colonne 'en cours' ne soit pas surchargée et en ayant recours à une bonne DoD, on assure que notre travail avance de manière efficace et contrôlée.

IceScrum

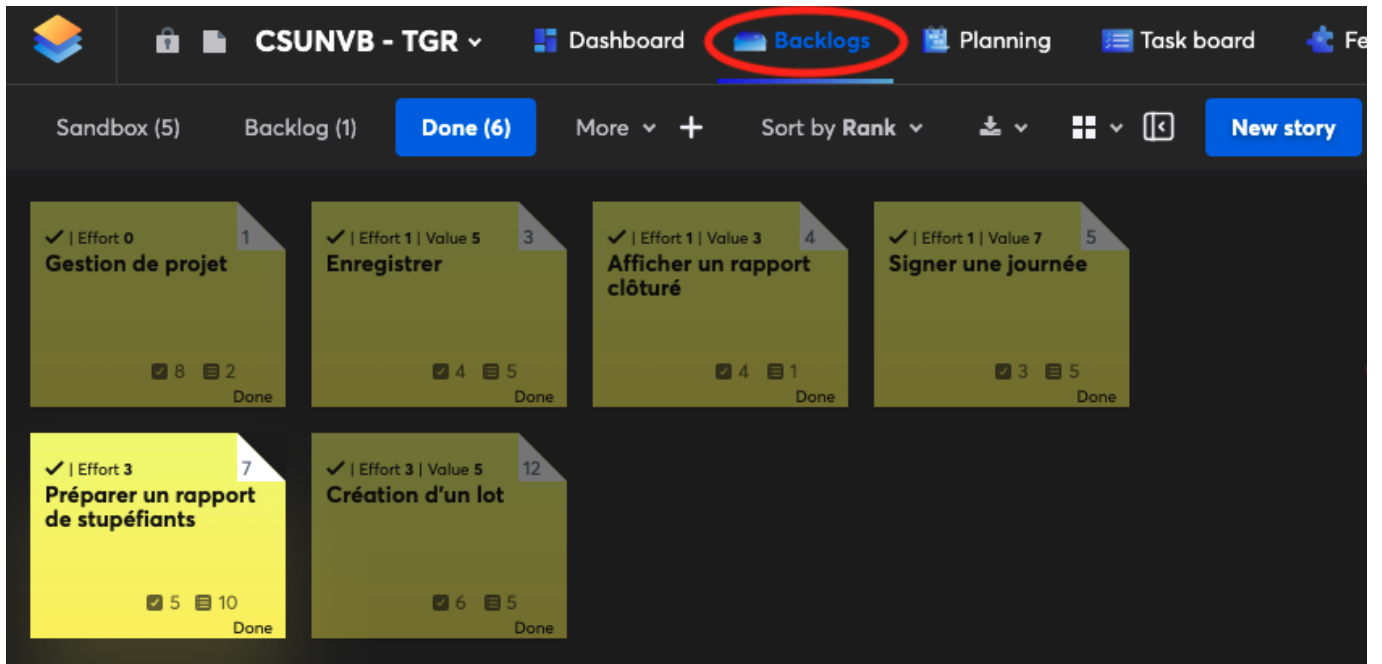
Dans cette section nous abordons dans les grandes lignes comment mettre en œuvre les pratiques et outils (méthodologiques) vus précédemment avec l'application IceScrum.

Features

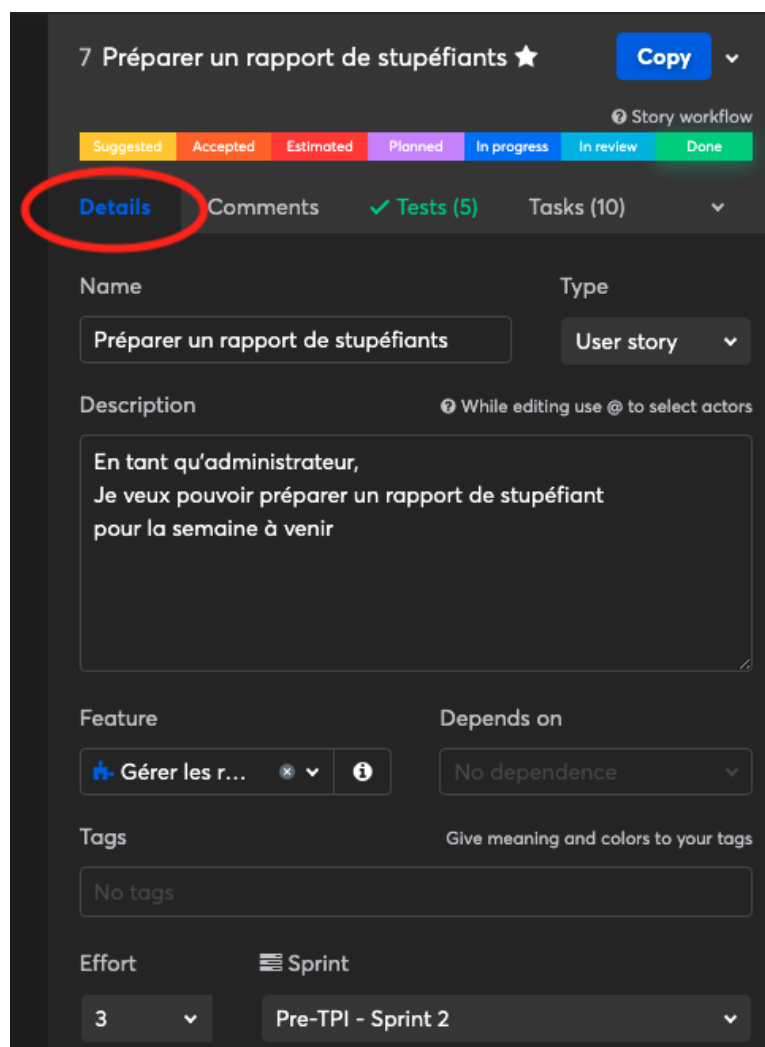
The screenshot shows the IceScrum application interface. The top navigation bar includes links to Dashboard, Backlogs, Planning, Task board, and Features. The main section is titled 'Features (3)' and shows three feature cards. The first card, 'M'authentifier', is in progress (0/5). The second card, 'Consulter mes rapports', is a todo item (3/3). The third card, 'Introduire un check', is a todo item (4/4). The right sidebar shows the details for the selected feature 'Introduire un check'. It includes a 'Create an epic story' button, tabs for Details, Comments, Stories (4), and History. The Name field is 'Introduire un check'. The Type is 'Functional' and the Business value is '5'. The Description is: 'Les pharmacheck et novacheck sont des contrôles dont les valeurs sont collectées en dehors du bureau. Il est donc particulièrement utile de pouvoir reporter ces valeurs directement au moyen de l'application mobile plutôt que de les rapatrier jusqu'à l'ordinateur du bureau.'

Stories

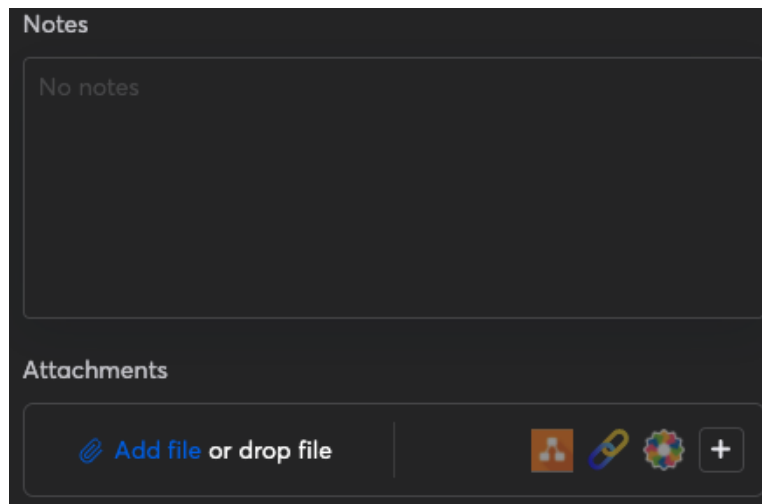
Elles sont regroupées dans les différents backlogs



L'onglet Détails permet de formuler la story, de l'associer à une feature et d'en estimer l'effort :



Cet onglet permet également d'enregistrer toutes sortes d'informations utiles à la bonne compréhension de la story et notamment une ou plusieurs maquettes.



L'onglet Tests permet de formuler la DoD de la story. Il est nécessaire que le test soit formulé de telle manière qu'il soit vérifiable de manière non-ambiguë (une fois de plus, pensez « SMART » !).

Pour une utilisation fluide d'IceScrum, veillez à ce que le titre du test ne dépasse pas la taille du champ de saisie.

Ce n'est que quand le PO valide la liste des tests que la story peut passer de la SandBox au Product Backlog.

7 Préparer un rapport de stupéfiants ★ Copy

Story workflow

Suggested Accepted Estim **Acceptance tests** Press In review Done

Details Comments **✓ Tests (5)** Tasks (10)

31 ✓ Success

Le rapport en préparation est calqué sur le principe de préparation des rapports de todo:

- Une liste déroulante pour choisir un médicament
- Une liste déroulante pour sélectionner un des lots du médicament choisi
- Un bouton 'Ajout'

Quand on clique sur 'ajout', le lot choisi est ajouté au rapport

32 ✓ Success

Le rapport en préparation est calqué sur le principe de préparation des rapports de todo:

- Sous chaque lot, il y a un bouton 'poubelle'
- Quand on clique sur la poubelle, le lot est supprimé du rapport

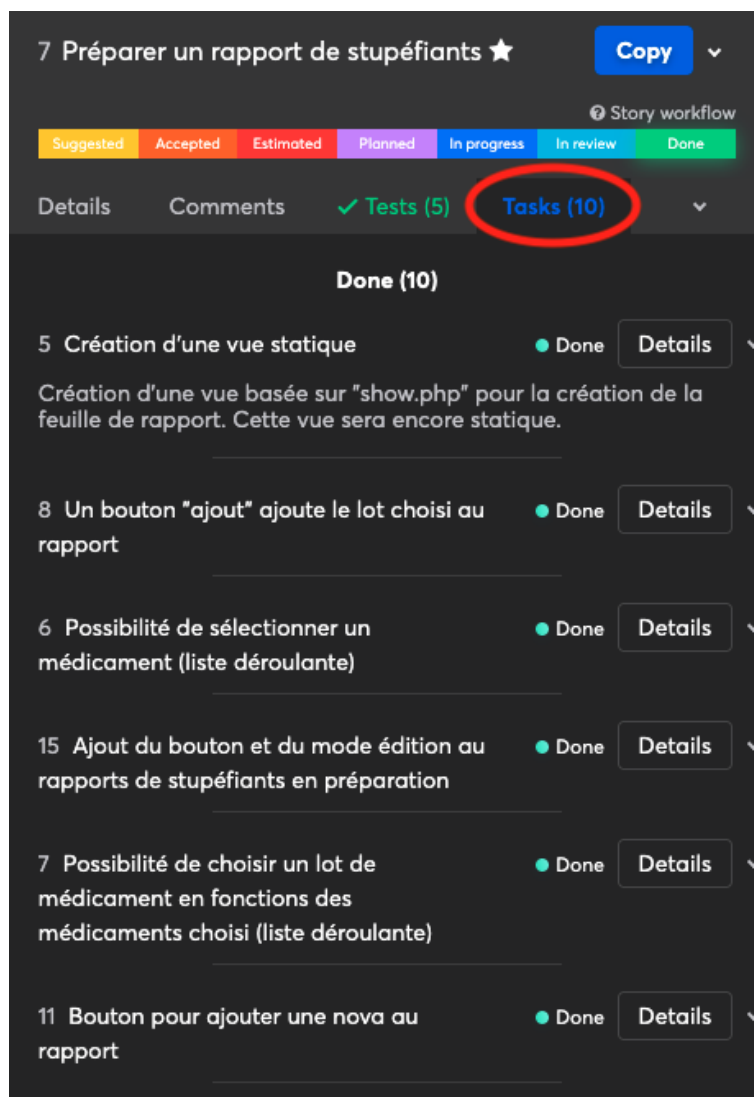
33 ✓ Success

Le rapport en préparation est calqué sur le principe de préparation des rapports de todo:

Attention : le PO est la seule personne qui est habilitée à confirmer qu'une story est prête pour sa réalisation. IceScrum applique cette règle. Malheureusement, IceScrum considère également que l'utilisateur qui a créé le projet a tous les droits sur ce dernier. Il est donc fortement conseillé que ce soit le PO qui crée le projet IceScrum, sinon un simple membre de l'équipe pourrait valider une story.

L'onglet Tasks permet de formuler les tâches qui devront être effectuées pour arriver à satisfaire les critères d'acceptation de la story. Plus cette liste de tâches est précise avant de commencer le travail, plus il sera possible d'évaluer l'effort requis par la story.

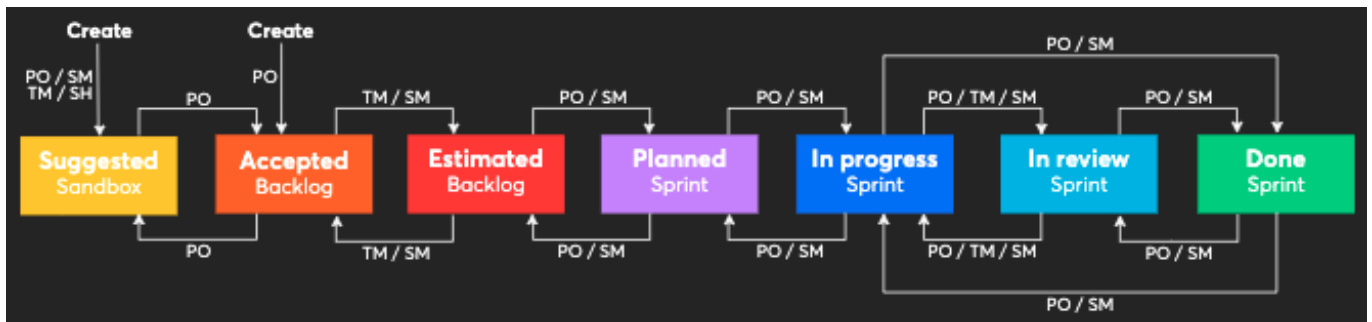
Le PO n'est pas impliqué dans la validation de ces tâches. Elles ne concernent que l'équipe de développement.



Une fois un premier Product Backlog constitué, nous pouvons commencer la planification proprement dite.

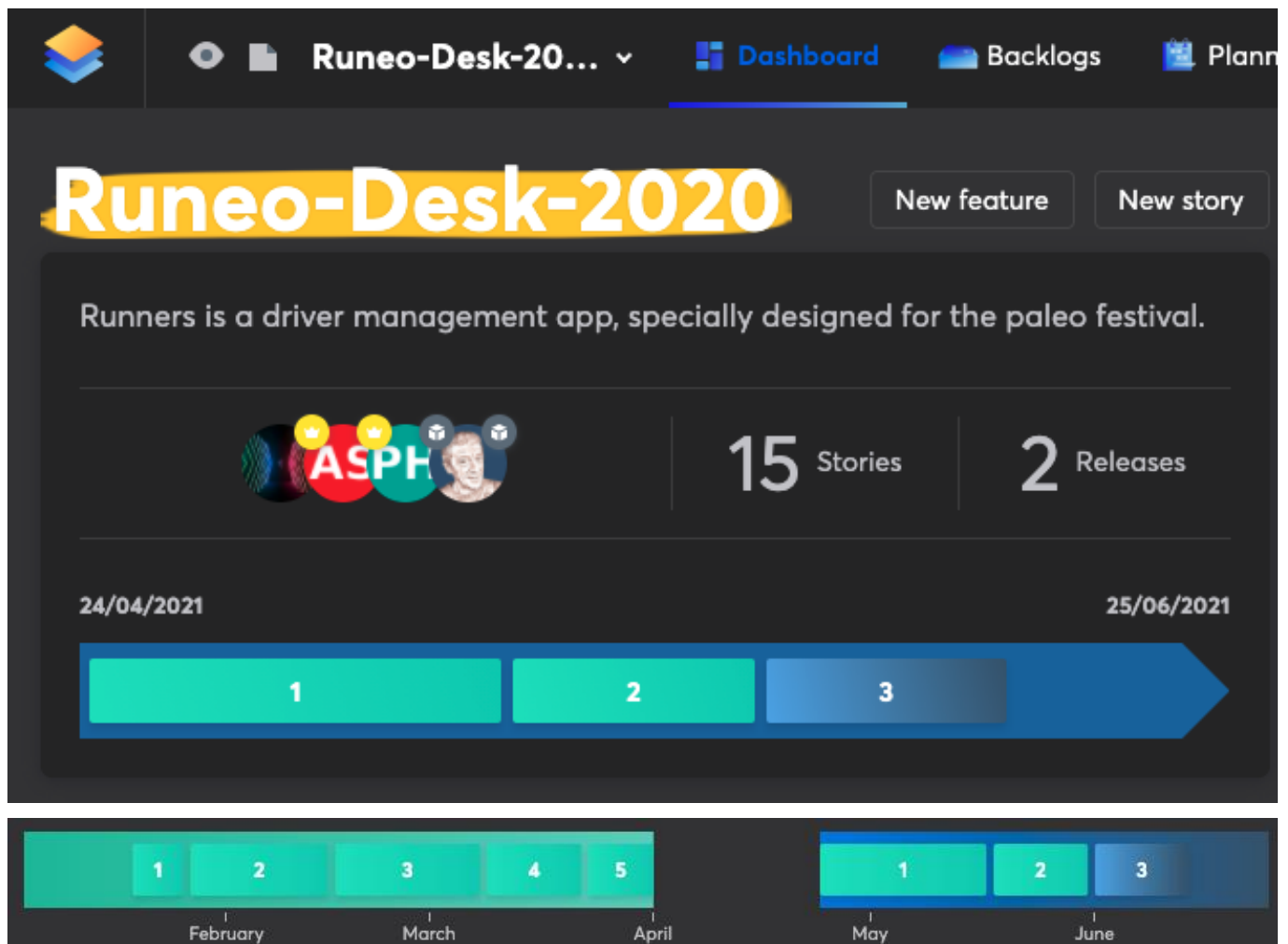
Pour bien travailler avec IceScrum, il est important de bien comprendre ce qu'est la vie d'une story. Et pour bien se comprendre, un moyen efficace de clarification est ... le diagramme d'état !

IceScrum propose donc un « story workflow » :



Remarque : par défaut, l'état « In review » est désactivé à la création du projet. Nous recommandons de l'activer car il permet une distinction intéressante des stories en cours de sprint : celles sur lesquelles on estime avoir fini de travailler (puisqu'elles sont prêtes à être revues) mais sans que le résultat n'ait été validé par le PO (puisque la sprint review n'a pas encore eu lieu).

Avant de pouvoir planifier des stories (durant le Sprint Planning), il est tout d'abord nécessaire de définir au moins une Release dans le projet. Chaque Release contiendra un certain nombre sprints :

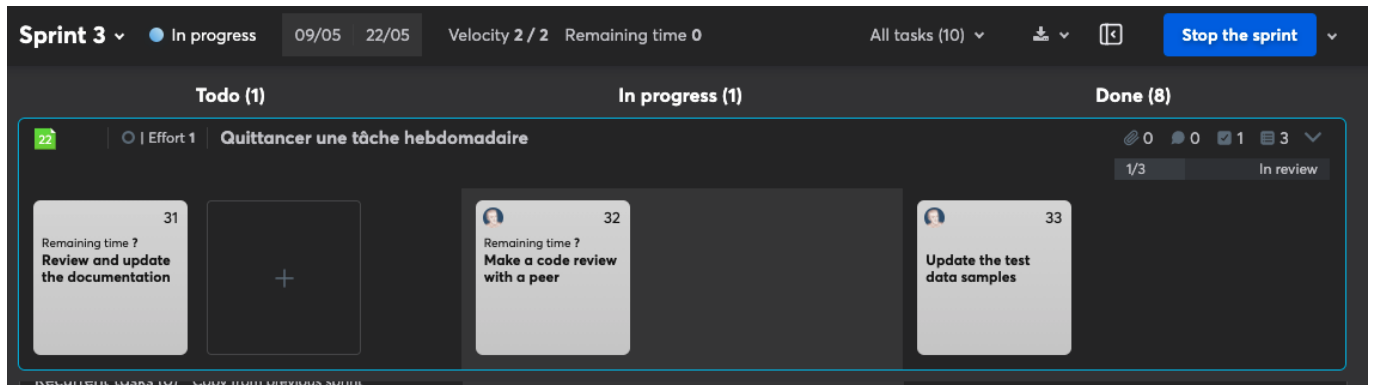


Lors du sprint planning :

- On choisit des stories du product backlog et on les passe dans le sprint backlog
- On active le sprint

A tout moment dans un projet IceScrum, il y a zéro ou un sprint actif. Attention : lorsque vous activez un sprint, il n'est plus possible de revenir en arrière.

Durant le sprint, IceScrum permet de travailler sur le sprint actif dans une vue sous forme de KanBan ('Task Board'). Chaque tâche définie lors de l'analyse technique (ou rajoutée par la suite) :



Quand toutes les tâches de la story ont été faite, on met la story dans l'état « In review ».

Durant la Sprint Review, on commence par montrer le résultat au PO :

- Si le PO est satisfait du résultat (= il valide que les tests d'acceptance sont réussis), on passe la story en « Done » et on a augmenté l'incrément produit du sprint !

Attention : dans IceScrum, le fait de mettre une story en « Done » met automatiquement le résultat de tous ses tests d'acceptance à « Success ». Cela peut mener à la situation embarrassante décrite ci-dessous.

- Si un (ou plusieurs) test d'acceptance échoue, on remet la story dans l'état « In Progress »

A la fin de la Sprint Review, on va clore le sprint. Les stories qui sont en état « In Progress » sont automatiquement transférées au backlog du sprint suivant.

Une situation embarrassante...

On a une story qui a nécessité pas mal de travail.

Un des tests d'acceptance dit « Le bouton qui permet de confirmer l'envoi de la demande est automatiquement sélectionné (focus) lors de l'ouverture du formulaire ».

La SR arrive et on a fait un gros et bon travail sur cette story, mais on a juste zappé ce test là ! ☐

Ça nous embête parce que tout ce travail va être ignoré (pour le sprint en cours). On choisit donc de mettre la story en « In Review » bien que l'on sache pertinemment qu'un test ne passe pas.

Durant la SR, le PO est donc confronté à un choix difficile :

- Soit il rejette la story et on perd un incrément produit conséquent pour une brouille
- Soit il accepte la story, mais IceScrum mettra ce test à « Success » et c'est un mensonge

Dans une telle situation, nous proposons au PO la démarche suivante :

1. On supprime le test en question
2. On accepte la story (donc sans mentir)
3. On enregistre le focus automatique comme étant un des bugs connus du système

Le PO accepte ou non la démarche en fonction de l'importance qu'il accorde à ce test et de la taille de la liste de bugs connus (qu'il ne veut pas voir grossir continuellement).

Tâches urgentes et récurrentes

Il arrive régulièrement que l'on doive faire quelque chose de nécessaire à la réalisation de notre mandat mais qui n'est pas directement lié à une story.

Exemples :

- Participer à une des cérémonies SCRUM
- Mettre en place une infrastructure de test

Icscrum permet de garder la trace de ce genre d'activité dans le task board au moyen de tâches classées comme soit urgente soit récurrente:



5 Ne me dis pas qu'on a perdu ces fichiers ?

Problématique

Quelque soit la nature du mandat qui nous est confié (développement, système, réseau, PoC, ...), nous allons produire des documents numériques : code source, diagrammes, rapports, procédures, ...

Mais ni le matériel, ni le logiciel et encore moins les humains qui les manipulent ne sont infaillibles.

Et donc fatalement arrive le jour funeste où le fruit de notre travail s'est évaporé.

Il est d'autre part tout-à-fait possible de faire fausse route au beau milieu d'un projet. On doit donc recommencer, mais pas forcément à partir de zéro.

Pratique

La pratique la plus répandue pour adresser ces problèmes consiste à mettre notre travail sous contrôle de version (VCS : Version Control System)

Outils

Il existe [de très nombreux VCS](#).

Nous avons choisi de nous servir de Git avec Github.

Il est important de ne pas confondre les deux :

- Git peut être assimilé à une technologie, définissant des structures de données et des concepts qui permettent de gérer les version
- Github est un outil qui nous permet de gérer nos documents avec la technologie Git

Github

Cette section se contente de donner une introduction à l'utilisation de Github. Le net regorge d'informations, de tutoriels et de documents sur le sujet qu'il serait vain et inutile de répéter ici.

Repository

L'entité principale dans Git est le Repository ('Dépôt' en français), souvent abrégé 'Repo'.

Formellement, un repo Git est le dossier `.git/` qui se trouve dans le dossier d'un projet. Ce dépôt suit toutes les modifications apportées aux fichiers de votre projet (dossier et sous-dossiers), en construisant un historique au fil du temps. Autrement dit, si vous supprimez le dossier `.git/` vous supprimez l'historique de votre projet.

Par abus de langage, on utilise souvent le terme de repo pour désigner l'ensemble formé du dossier de projet et du `.git/` qu'il contient.

Local vs Remote

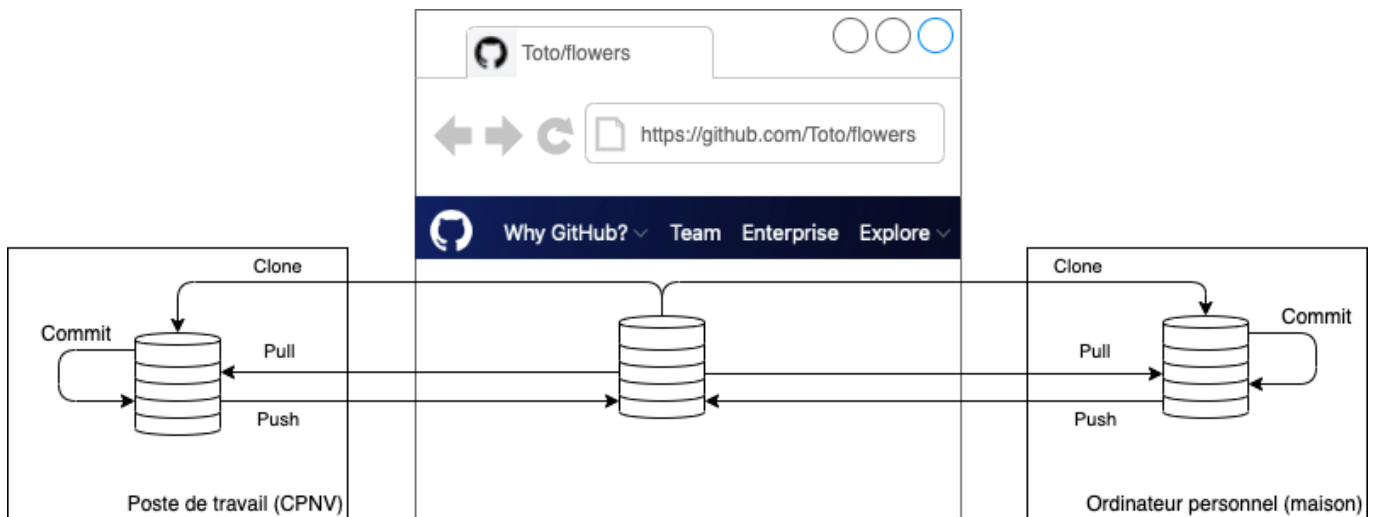
Git est un système de gestion de version décentralisé. Cela veut dire qu'il peut exister de nombreuses copies du repo disséminées sur plusieurs machines.

Il y a une et une seule copie d'un repo qui existe sur Github : on la nomme 'Remote'

Chaque copie du repo qui est faite sur une machine est un 'repo local' (à la machine en question).

Ce système donne beaucoup de flexibilité de travail. Son inconvénient est qu'il faut bien gérer la synchronisation entre le remote et les repos locaux.

Commandes



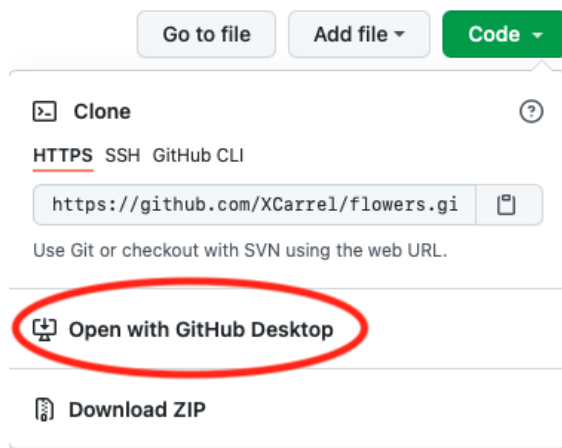
Cette figure montre les principales commandes qui vous permettront de commencer à travailler avec Github.

- Clone : crée un repo local à partir du repo remote.
- Pull : met à jour le repo local à partir du remote
- Push : met à jour le repo remote à partir du local
- Commit : enregistre dans le repo tous les changements effectués dans le projet depuis le dernier Pull, Commit ou Clone

Exemple d'utilisation :

Sur mon poste de travail CPNV et sur mon ordinateur personnel à la maison, j'ai installé Github Desktop et j'ai accès à Github.com.

1. Depuis le CPNV, je crée un nouveau repo sur Github. Je le nomme 'flowers'. C'est le repo remote. (Il est très fortement recommandé de créer le repo avec un README)
2. Depuis le site Github.com, je crée un repo local sur mon poste de travail. Je peux le mettre où je veux sur mon poste.



Attention : ne mettez pas vos repos locaux sur les lecteurs réseau. Cela ne sert à rien et ne peut que vous causer des problèmes.

3. Je travaille sur mon projet flowers dans le repo local: je crée des documents, des images, des sous-dossiers, ...

4. J'ai terminé une tâche : je fais un **commit**. Cela a pour effet d'enregistrer tout mon travail dans le repo local.
5. Je fais un **push**. Cela a pour effet de mettre à jour le remote sur Github.
6. Je continue à travailler. Je crée de nouveaux fichiers, dossiers, ... Je fais également certaines modifications à des fichiers que j'ai déjà commités/pushés précédemment.
7. J'ai terminé ma deuxième tâche : je fais un **commit**. Cela a pour effet d'enregistrer tout le travail que j'ai fait depuis le dernier commit dans le repo local.
8. Je fais un **push**. Cela a pour effet de mettre à jour le remote sur Github.
9. J'ai terminé ma journée, je rentre chez moi. Mais il y a encore une chose que j'aimerais bien terminer aujourd'hui...
10. Arrivé chez moi, je vais sur github.com et je crée un repo local sur mon ordinateur personnel. L'état de mon projet est alors exactement le même que celui sur mon poste au CPNV.
11. Je fais cette fameuse chose que je voulais terminer dans le repo local de mon ordinateur.
12. Quand j'ai terminé, je fais un **commit**. Cela a pour effet d'enregistrer mon travail dans le repo local.
13. Je fais un **push**. Cela a pour effet de mettre à jour le remote sur Github.
14. Le lendemain, j'arrive au CPNV. Mon projet dans mon repo local est en retard puisqu'il ne contient pas mon travail d'hier soir.
15. Je fais un **pull**. Mon repo local est maintenant à jour, je peux continuer à travailler sur ce poste.
16. Je fais plusieurs **commit** / **push** pendant la journée.
17. Le soir chez moi, le repo local de mon ordi est en retard. Je fais **pull**, il est mis à jour et je peux continuer (ou pas).

Conflits

Il est très important de faire pull quand on commence à travailler sur un poste et push quand on a fini. Ne pas le faire peut mener à des conflits difficiles, parfois impossibles à résoudre.

En effet, Git se rappelle de (presque) tout !

Admettons que j'aie un fichier 'tulipe.png' qui est à jour partout (remote, local CPNV, local maison).

Je le modifie au CPNV et je fais **commit** / **push**. Le tulipe.png du local à la maison est en retard. Si j'oublie de faire **pull** à la maison, que je modifie le fichier et que je fais commit puis push, Github va refuser le push parce que mes modifications n'ont pas été faites sur la dernière version de tulipe.png.

Nommage et contenu des commits

Pour profiter pleinement des avantages de l'outil, il convient de bien faire ses commits. Voici quelques règles à suivre :

- Faire des commits qui couvrent uniquement et entièrement une tâche.
Exemple :
Vous devez ajouter une information à saisir dans un formulaire de l'application que vous développez. Cela implique de modifier plusieurs fichiers : MLD, script de base de données, formulaire, requêtes SQL. Faites un seul commit qui contient tout ces fichiers et rien que ces fichiers.

- Nommez vos commits dans une forme active.

Un bon moyen d'aide à la formulation est de compléter l'une des deux phrases suivantes

1. « Si on l'applique, ce commit va ... »
2. « Si on applique ce commit, le système va ... »

Le nom de votre commit est ce que vous mettriez à la place des points de suspension.

On utilise la première phrase si le commit est de nature technique :

- Si on l'applique, ce commit va **déplacer les images dans un sous-dossier**
- Si on l'applique, ce commit va **publier la version 1.2**
- Si on l'applique, ce commit va **supprimer des dépendances inutiles**

On utilise la deuxième s'il est de nature fonctionnelle :

- Si on applique ce commit, le système va **permettre de saisir l'adresse du site internet**
 - Si on applique ce commit, le système va **permettre d'effacer les données périmées**
 - Si on applique ce commit, le système va **gérer la réactivation des comptes**
- Il peut arriver qu'un commit n'apporte rien de véritablement significatif : correction de fautes d'orthographe, mise en page, détail cosmétique, ... Dans ce cas, épargnez-vous la peine de trouver un nom significatif et mettez un simple terme entre parenthèses : (orthographe), (cosmétique), ...

6 Est-ce que ça marche vraiment ?

Problématique

Pendant la réalisation de notre mandat, nous travaillons dans notre environnement de travail que nous connaissons bien. Plus ou moins consciemment, nous adaptons nos choix à cet environnement. Pire encore (nous sommes humains): nous avons tendance à favoriser les tests que l'on sait qu'ils vont passer.

Le moment venu de mettre en service ou de publier notre système, l'environnement change, l'utilisateur change, et soudain les problèmes surgissent.

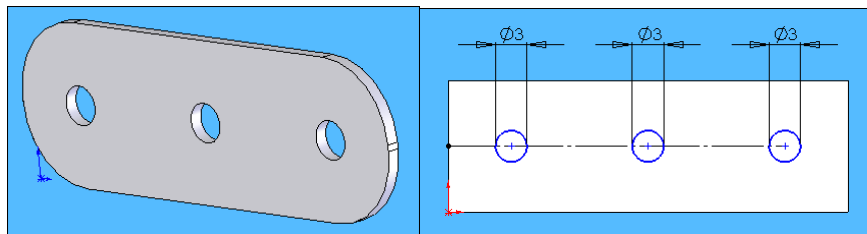
Comment faire pour minimiser ces problèmes ?

Pratiques

Niveau de test

Dans la mesure du temps et des moyens à disposition, nous essayons de faire des tests de chacun des trois niveaux de test suivants :

- **Le Test unitaire** : Il vérifie le bon fonctionnement d'un composant réalisé par une personne.



Exemple : on vérifie qu'une pièce de Mécano respecte les dimensions voulues

- **Le Test d'intégration** : Il vérifie que deux ou plusieurs composants testés unitairement fonctionnent correctement ensemble.



Exemple : on vérifie que diverses pièces de Mécano peuvent bien être assemblées au moyen des vis et écrous.

- **Le Test Système** : Il vérifie le bon fonctionnement du système complet dans l'environnement où il va être vraiment utilisé.



Exemple : on vérifie le bon fonctionnement de notre construction complète.

Type de test

Dans la mesure du temps et des moyens à disposition, nous essayons de faire des tests de chacun des quatre types de tests suivants :

- Le Test fonctionnel, qui vérifie que l'élément testé fait ce qu'on attend de lui ;
- Le Test de performance, qui vérifie que l'élément testé peut traiter la quantité d'information voulue dans le temps voulu ;
- Le Test de robustesse, qui vérifie que l'élément testé peut reprendre un fonctionnement normal après avoir passé par une situation anormale.
- Le Test de non-régression, qui vérifie qu'un changement effectué à un endroit du système n'a pas affecté le bon fonctionnement du système dans son ensemble.



Exemple : considérons la machine ci-dessus.

Elle sert à imprimer des flyers, les plier, les insérer dans une enveloppe, fermer l'enveloppe et faire des paquets de 100 enveloppes fermées.

Un test fonctionnel vérifiera que les flyers sont bien pliés en trois volets égaux et qu'il y a bien 100 enveloppes par paquet au final

Un test de performance vérifiera que la machine est capable de produire 250 paquets de 100 enveloppes en une heure

Un test de robustesse vérifiera que si on fournit des enveloppes qui sont déjà remplies, le système les écarte et reprend son fonctionnement dès que les enveloppes fournies sont vides

Stratégie de test

Établir une stratégie de test permet d'analyser et d'anticiper les risques de problèmes au moment de la livraison.

Pour tout mandat/projet, nous définissons une stratégie de test, qui consiste à décrire :

1. Le matériel et logiciel tiers.
2. Les données de test.
3. Les personnes qui vont participer aux tests : camarades de classe, amis, famille, profs, ...
4. Le timing des activités de test
5. Les types et niveaux de tests effectués

Exemple (site web pour un yearbook):

Pour le développement du site web, le serveur sera un wamp installé sur le même PC que celui sur lequel j'écrirai le code. Sur ce poste, je ne ferai que des tests fonctionnels.

Je préparerai

- Un script SQL qui créera 60 élèves répartis dans 4 classes.
- Un ZIP contenant des photos fictives pour les 60 élèves

J'utiliserai ensuite une machine virtuelle VMWare avec Debian 9 installé dessus et un serveur LAMP. Cette machine virtuelle sera également sur mon PC de développement. Elle ne servira qu'à faire des tests fonctionnels. Elle sera visible sur le réseau de l'école. Mon chef de projet ainsi que deux de mes camarades de classe l'utiliseront pour faire quelques tests.

Je déploierai mon site du l'hébergeur 'Swisscenter' pour faire les tests fonctionnels finaux. Avec l'aide d'autres camarades, nous effectuerons également des tests de robustesse et de performance en se connectant à plusieurs simultanément.

Les tests fonctionnels seront faits avec les navigateurs Google Chrome et Firefox sur Windows, Google Chrome, Firefox et Safari sur Mac. Aucun test n'est prévu sur Edge.

Outils

Étant donné la démarche prescrite (validation des tests par le PO), une grande quantité de tests a forcément été exécutée lorsque l'on referme le dernier sprint de notre projet.

Mais rien ne garantit que la couverture de test soit bonne. Le PO ne peut pas énoncer tous les cas de figure avant de confirmer une story pour le backlog. Il ne peut pas non plus essayer toutes les combinaisons possibles et imaginables durant la Sprint Review. Il convient donc de rester prudent quant à la qualité véritable de notre produit quand il a été accepté par le PO.

D'autre part, durant la SR, on ne teste que les stories du sprint et rien ne garantit de nouveau que la réalisation d'une story de ce sprint n'a pas « cassé » une story du sprint précédent (régression).

Le seul moyen de faire monter et de conserver un bon niveau de test tout au long de la vie d'un produit est d'avoir recours à un framework de test.

Nous n'en abordons aucun dans ce document.

7 Qui a fait quoi ?

Problématique

La réalisation complète d'un mandat demande des compétences diverses et variées. Si le mandat est réalisé par une équipe, le groupe de travail possède des compétences diverses et variées. La réalisation d'une même tâche dans un mandat ne se déroulera donc pas toujours de la même manière d'une personne à une autre ou d'un groupe à un autre.

Comment faire pour s'améliorer et augmenter notre efficacité – et de là notre satisfaction et celle du client ?

D'autre part, dans le monde réel, il est souvent important de connaître le détail des heures de travail, soit pour savoir ce que nous a coûté un projet, soit pour facturer des heures à un client.

Pratiques

La Sprint Restrospective est là pour ça. Nous allons chercher à identifier quelles sont les tâches qui se sont bien déroulées et lesquelles nous ont mises en difficulté. Cela nous permet d'en analyser le déroulement, puis d'identifier et promouvoir de bonnes pratiques, et de corriger nos moins bonnes pratiques.

Un moyen d'identifier les tâches coûteuses est de faire le total du temps passé sur chacune des tâches et de les mettre en rapport avec leur véritable difficulté (selon un avis d'expert, ou selon l'expérience acquise). Il est également intéressant de comparer la durée effective avec celle que l'on avait estimée.

Outils

Timesheet.html

Le fichier [Timesheet.html](#) vous aide à collecter et à trier les données enregistrées dans IceScrum.

Téléchargez-le et ouvrez-le avec votre navigateur. Ce dernier vous indiquera alors comment finaliser la configuration.

Vous devez créer un fichier nommé `_icescrum_api.js` dans le dossier où se trouve `Timesheet.html`

Dans ce fichier, vous devez mettre:

1. Votre token d'accès à l'API Icescrum. Cela donne quelque chose genre: `var iceScrumToken = '91030e0ea9a9ab58...'`
Vous générez ou retrouvez votre token dans votre profil IceScrum ("My Account") sous l'onglet 'API token'
2. Un tableau contenant les codes Icescrum des projets que vous voulez suivre, genre: `projectIds = ['XXXXXX', 'YYYYYY']`
Vous trouverez la valeur à mettre à la place des XXXXXX dans la barre d'adresse de votre navigateur quand vous êtes sur votre projet Icescrum (<https://icescrum.cpnv.ch/p/XXXXXX/#/project>)

Attention: ajoutez `*_icescrum_api*` dans le fichier `.gitignore` de votre repository.

Si vous ne le faites pas, votre token sera publié sur Github!

Une fois la configuration faite, sélectionnez un de vos projets et cliquez le bouton 'Aide' pour vous guider.

8 Mais qui a décidé ça ? et quand ?

Problématique

Dans un projet, il y a des moments marquants planifiés, mais il y a aussi toujours des événements imprévus qui surviennent. Toujours.

On est donc appelé à prendre des décisions en réaction à ces événements.

Il arrive parfois que l'on oublie les motivations d'une décision et on ne parvient pas à justifier un choix qui est remis en question tardivement :

« Mais enfin, pourquoi est-ce que vous avez abandonné la librairie TrucMachin ? Elle fait pourtant le job, non ? »

Pratiques

Le moyen le plus simple est de tenir un journal de bord, lequel a pour but de retracer l'histoire du projet au travers de ses faits marquants uniquement

Exemple :

Date	Événement
8.9.16	Revue des Use cases avec M. Gates : des ajustements sont demandés au niveau de la gestion des membres
9.9.16	Revue des Use cases avec M. Gates : ils sont validés
9.9.16	Rétro-planification de planning : aucun problème en vue
14.9.16	M. Gates demande (par mail) l'ajout d'une page d'historique des achats

9 C'est documenté où ?

Problématique

Un projet de longue durée a toutes les chances de voir passer plusieurs personnes en cours de route.

Où alors, quelqu'un démarre un nouveau projet qui s'appuie sur le vôtre.

Dans un cas comme dans l'autre les nouvelles personnes impliquées auront besoin de plus d'informations que ce qu'ils trouveront dans le produit livré.

Pratiques

Faire de la doc...

La documentation d'un projet réalisé dans la filière informatique du CPNV consiste au final en un document (livré en format PDF) contenant :

- Une introduction avec :
 - Une description générale du projet
 - Les informations sur le mandant du projet
 - Les informations sur le mandataire (nous)
- Des références (pas de copie !) aux documents fournis en guise d'énoncé (cahier des charges, projet précédent, emails explicatifs, ...)
- L'analyse préliminaire
- Les détails de réalisation qui méritent explications. On entend par là les choses que l'on a faites dans le projet qui ne sont pas « habituelles ».
Exemple : mon projet requiert l'installation d'un serveur DHCP. S'il n'y a rien de particulier à son propos, je me contente de citer le fait qu'il y a un serveur DHCP. Le professionnel qui reprendra le projet à partir de ma doc saura le faire. Admettons maintenant que mon serveur DHCP fonctionne sur un port non-standard. Je dois maintenant décrire dans la doc sur quel port mon serveur fonctionne et expliquer pourquoi je l'ai fait ainsi.
- Les détails de la livraison, telle que décrite au chapitre plus loin dans ce document
- Le Journal de Bord, tel que décrit précédemment
- Une conclusion, contenant entre autres :
 - Un commentaire critique de comparaison entre ce qui était demandé et ce qui a été réalisé
 - La liste des problèmes connus
 - Un commentaire personnel sur l'ensemble du projet

10 On en est où ?

Problématique

En plus du PO, il peut y avoir des personnes qui portent un intérêt à notre projet et à son avancement (on utilise parfois le terme anglais « stakeholder »).

Comment les tenir informés sans que cela ne nous demande trop de travail supplémentaire ?

Pratiques

Étant donné que nous travaillons dans un mode agile, nous valorisons la transparence.

Nous communiquons donc à intervalles réguliers les artefacts que nous maintenons de toute manière :

- Le Journal de Bord
- Le Journal de Travail,
- Le Document de Projet.
- Le contenu actuel du projet (Github)

Sauf cas exceptionnel demandé spécifiquement par le mandant, tous les documents sont toujours transmis en format PDF.

Outils

Comparaison de documents dans Word

Dans les journaux, on ne fait que rajouter des lignes. Il est donc facile pour le mandant de détecter les changements intervenus entre deux envois : il suffit d'aller consulter le bas du tableau, qui est organisé par ordre chronologique.

Il n'en va pas de même pour le Document de Projet, dans lequel des modifications difficilement décelables peuvent se situer dans des endroits que le mandant a déjà lu plusieurs jours auparavant. Du coup il y a de fortes chances pour que certains changements passent inaperçus.

Pour guider le lecteur, nous fournissons le document de projet en deux exemplaires :

1. Le document lui-même
2. Un exemplaire qui met en évidence les modifications intervenues depuis la version précédente.

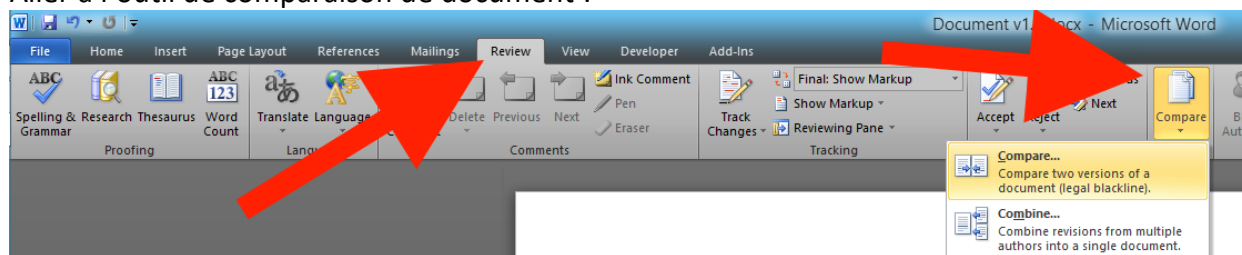
Exemple : si l'on travaille sur un document « Projet Alpha.docx » et que l'on publie la version 1.3, on fournira :

1. « Projet Alpha v1.3.pdf »
2. « Projet Alpha v1.3 – différences.pdf »

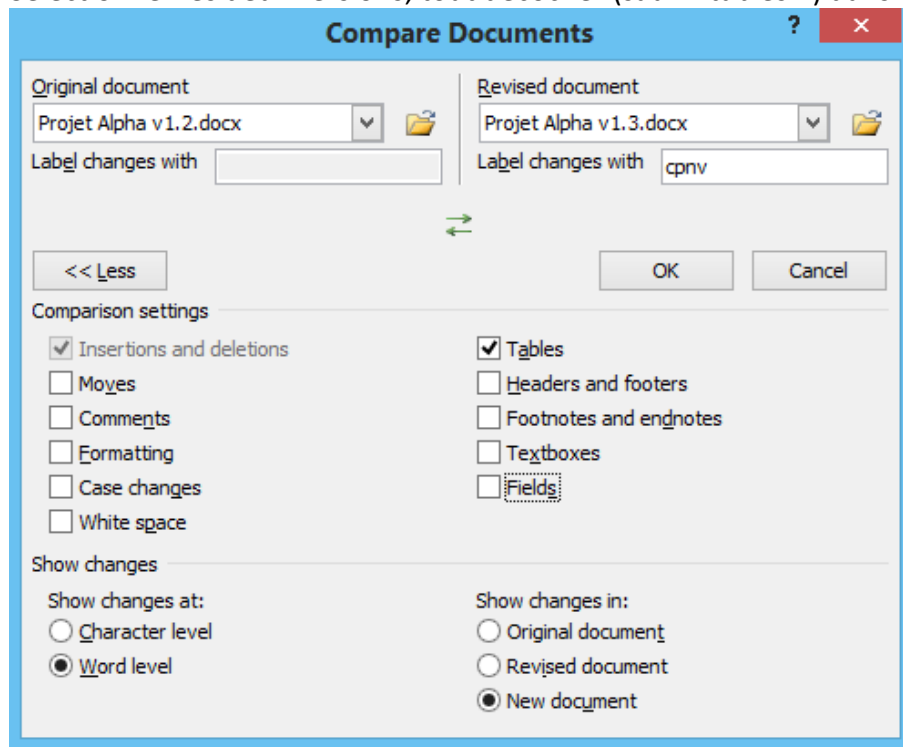
Remarque : ces fichiers n'ont pas besoin d'être enregistrés dans le repository Git, ce dernier contenant les originaux.

Pour créer la version « ... - différences » avec Word :

1. Aller à l'outil de comparaison de document :



1. Sélectionner les deux versions, tout décocher (sauf « tables ») dans les options :



2. Effectuer la comparaison (OK) et sauver le résultat au format pdf.

11 Où est-ce que je trouve vos livrables ?

Problématique

Lorsque nous avons terminé notre travail et que nous sommes prêts à le remettre au mandant, nous devons faire particulièrement attention à être précis et complet.

Nous voulons que le mandant prenne possession du produit et puisse le mettre en œuvre sans avoir besoin de nous.

D'autre part, il est important de s'assurer que le mandant a bien reçu notre livraison et les autres parties prenantes du projet (parmi eux l'ensemble de l'équipe) soient informées que la livraison a eu lieu.

Pratiques

Nous livrons un fichier (ou une référence) unique qui contient tout le nécessaire.

Nous informons le mandant et les personnes concernées de la livraison par un canal de communication commun à tous.

Toute livraison est un moment important dans un projet, elle fait donc l'objet d'une écriture dans le journal de bord.

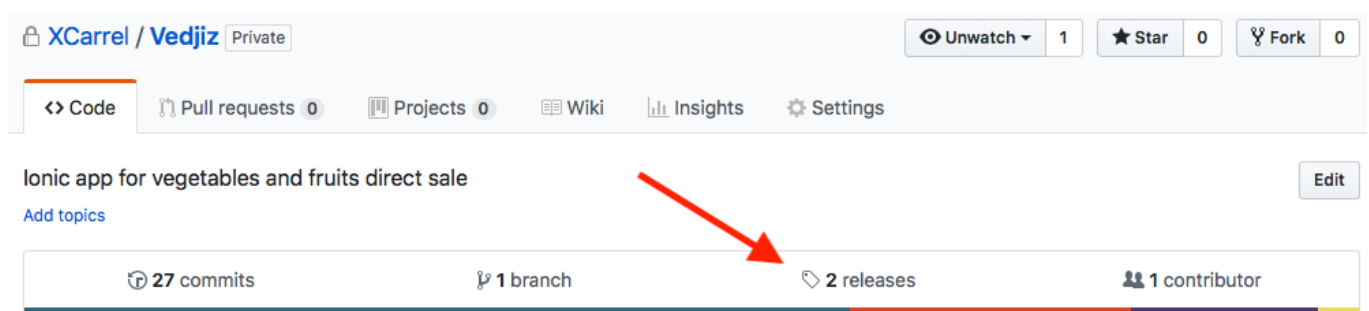
Outils

Nous livrons le résultat de notre travail au moyen « release » dans Github.

Une release est un commit du repository que l'on distingue des autres par des informations supplémentaires :

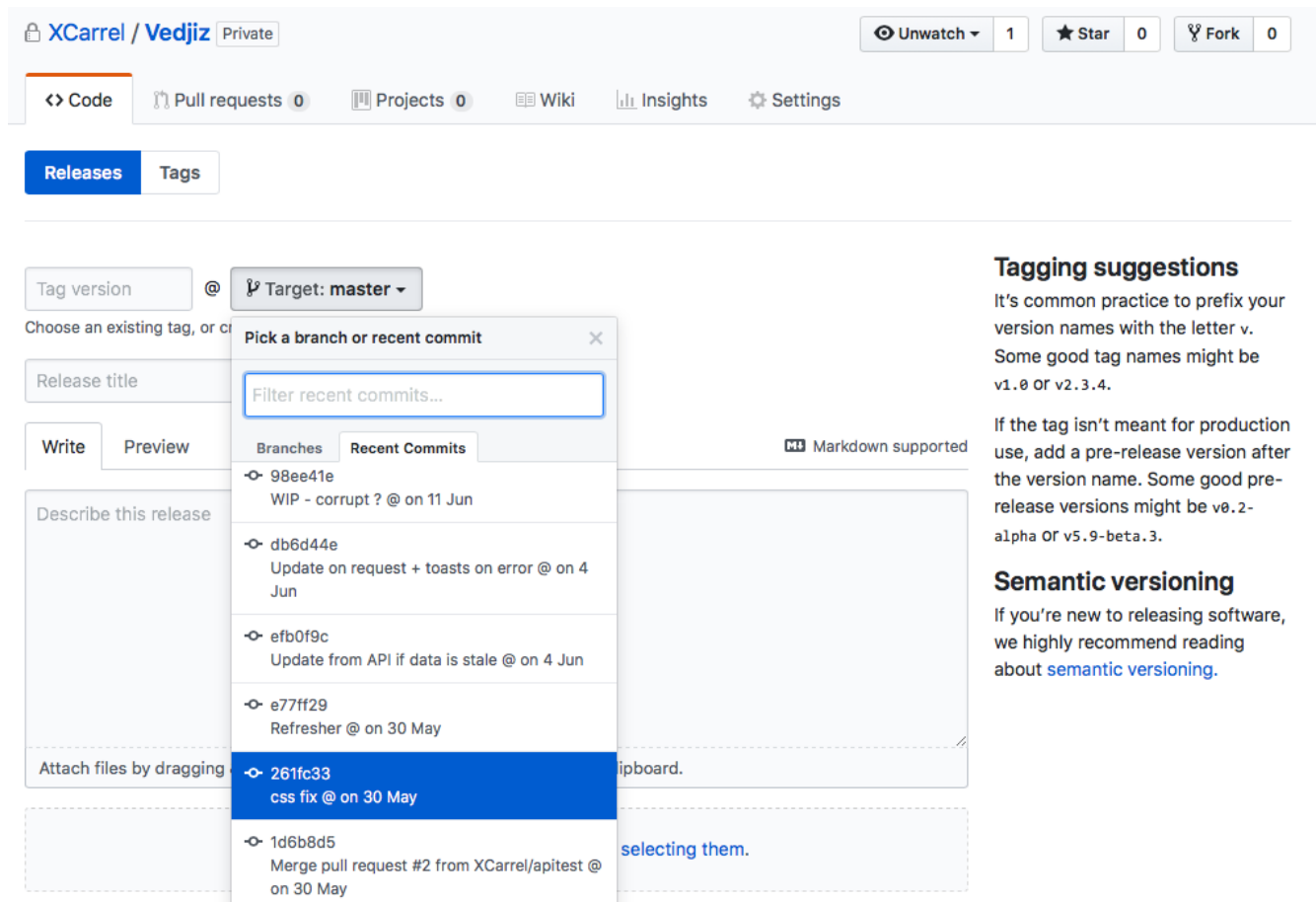
- Un numéro de version
- Un titre
- Une description

Pour créer une release, allez dans l'onglet « release » du repository



et cliquez sur « Draft a new release ».

Sélectionnez le commit que vous voulez livrer :



Tag version @ Target: master

Choose an existing tag, or create a new one

Release title

Write Preview

Describe this release

Attach files by dragging or pasting from clipboard.

Markdown supported

Tagging suggestions

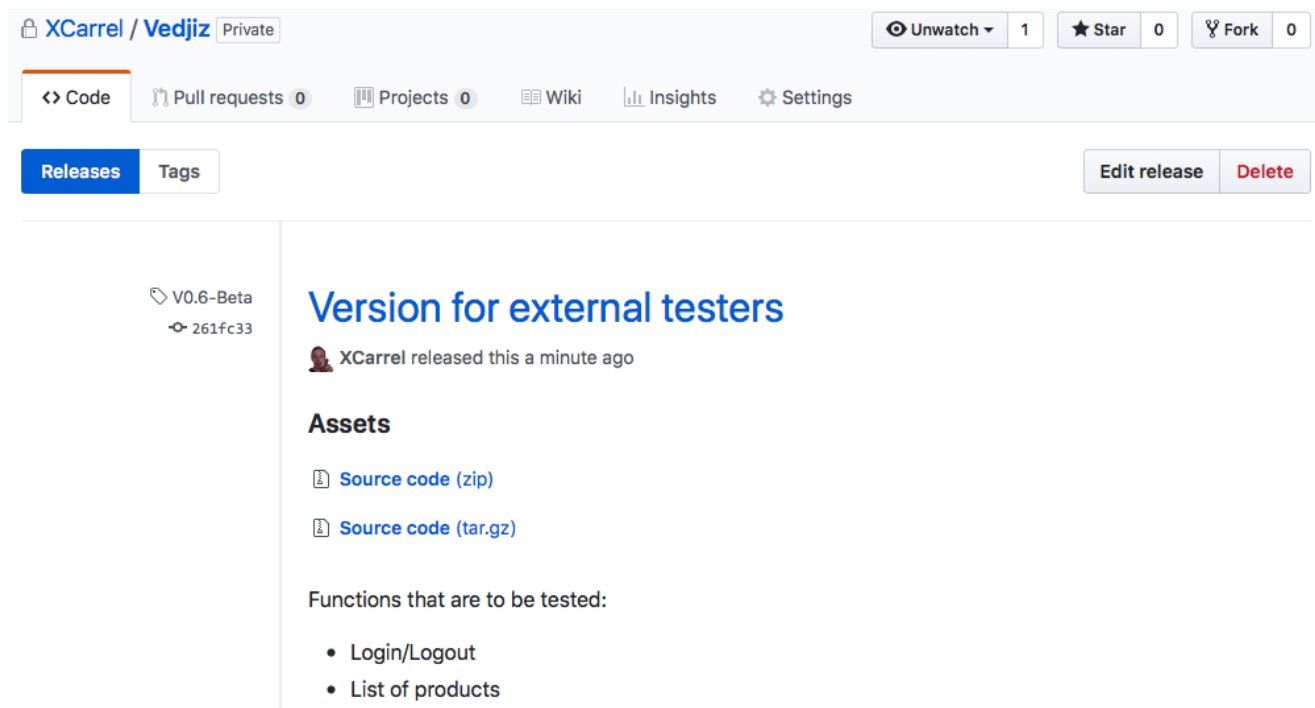
It's common practice to prefix your version names with the letter v. Some good tag names might be v1.0 or v2.3.4.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be v0.2-alpha or v5.9-beta.3.

Semantic versioning

If you're new to releasing software, we highly recommend reading about [semantic versioning](#).

Remplissez le reste du formulaire et publiez la version :



Unwatch 1 Star 0 Fork 0

Code Pull requests 0 Projects 0 Wiki Insights Settings

Releases Tags

Edit release Delete

V0.6-Beta
261fc33

Version for external testers

XCarrel released this a minute ago

Assets

- Source code (zip)
- Source code (tar.gz)

Functions that are to be tested:

- Login/Logout
- List of products

Il ne vous reste plus qu'à communiquer au mandant le tag et le titre de la release.

Si le mandant n'a pas accès à Github, faites-lui parvenir le .zip