# Des fonctionnalités aux user stories

## Une grosse fonctionnalité

Le traitement de texte devrait être capable de gérer des styles de formatage de texte.

## Une plus petite story

L'utilisateur devrait être capable de sélectionner un mot et d'indiquer que c'est de l'italique.

## Le modèle canonique de Mike Cohn

En tant que <un certain type d'utilisateur>, je veux <réaliser un certain objectif> afin de <pour une certaine raison>

*«* En tant *qu'utilisateur d'un traitement de texte,* je veux *sélectionner un mot et indiquer que c'est de l'italique,* afin que *je puisse mettre en évidence mon texte. »*

## Une simple suggestion

Vous n'êtes pas obligé d'écrire la story selon le modèle proposé par Mike Cohn (cela devient très monotone), mais il est bon de prendre en compte les trois points suivants :

1. Qui utilisera cette fonctionnalité ?
2. Que veux-t-on que le système fasse ?
3. Quel dessein plus large cette fonctionnalité sert-elle pour l'utilisateur ?

Une question importante supplémentaire

4. Qu'est-ce qui me dira que cette fonctionnalité est terminée ?

## Le modèle « Carte, conversation, confirmation » de Ron Jeffries

**Carte :** La carte de la story est juste le signal d'un besoin. Elle en est que le titre ; elle ne contient pas l'ensemble des détails. Vous pouvez écrire des notes sur tout élément que vous risqueriez d'oublier autrement, mais *ne vous attendez pas à ce que la story écrite puisse à elle seule transférer la compréhension du métier au développeur.* C'est l'objectif de la conversation.

**Conversation—Communication** : c'est un processus à double-sens, même quand nous essayons de transmettre une information dans un seul sens. Les développeurs ont besoin de poser des questions pour clarifier certains éléments. Les représentants du métiers ont besoin de poser des questions permettant de confirmer la compréhension de leurs besoins. Certaines fois, quelque chose surviendra qui remettra en cause la totalité de la story en question.

**Confirmation —** Comment saurons-nous que nous avons réalisé cette story ? Pouvez-vous montrez quelques exemples concrets ? Transformez ces exemples en tests d'acceptances automatisés.

### Le modèle INVEST de William Wake

De bonne stories sont : Indépendantes, Négociables, (de) Valeurs, Estimables, (Suffisament) petites, Testables

### Ne vous laissez pas embarquer

Toutes les tâches qui doivent être faites n'ont pas à être coulées dans le moule des user stories. Mais lorsque vous regardez quelque chose qui n'est pas une user story, demandez-vous si elle doit être *vraiment* faite. Ajoute-t-elle de la valeur ?

Et n'oubliez pas les autres utilisateurs : administrateur système, service client, homme d'affaires examinant le retour sur investissement ...

# Fractionner les stories

Les stories dans le backlog devraient être grosses mailles. Elles sont souvent appelées fonctionnalités ou épiques, elles sont trop grosses et vagues pour bénéficier d'un effort quelconque d'estimation précise. Généralement, elles contiendront à la fois des composants essentiels et souhaitables. Elles n'ont généralement aucun critère d'acceptance spécifique et testable. Au fur et à mesure, elles seront fractionnées en stories plus petites, plus concrètes pour le développement.

Les stories sélectionnées pour le développement devraient être généralement assez petites pour être implémentées en quelques heures ou en quelques jours tout au plus. Cela permet de suivre l'avancement efficacement pendant l'itération. Lors de leurs fractionnements à partir d'une fonctionnalité ou d'une épique, il est habituel de trouver quelques stories qui peuvent ou devraient être différées après la réalisation d'autres travaux plus importants.

Les développeurs peuvent vouloir fractionner des stories davantage lorsqu'ils les implémentent. Cela leur permet de vérifier l'avancée de leurs travaux et d'éviter les impasses ou les sur-implémentations. À ce niveau, un scénario de test est équivalent grosso modo à une story.

## Quelques suggestions pour le fractionnement

Gérez les cas simples avant les plus détaillés ou les plus complexes. Pensez en terme de raffinement progressif de la fonctionnalité.

- Zéro, un, plusieurs. Gérer d'abord le scénario vierge.
- D'abord le chemin optimal, puis chaque flux alternatif et les conditions exceptionnelles.
- D'abord la fonction principale, ensuite les prérequis pour un usage grand public
- D'abord une seule option, ajoutez ensuite chaque option supplémentaire.
- D'abord une interface utilisateur simple (ou bien aucune), ajoutez ensuite le superflu.
- D'abord le cas transitoire (aucune sauvegarde entre les sessions) avant de gérer la persistance.
- D'abord les éléments statiques, ensuite les éléments dynamiques selon le contexte.
- D'abord les actions manuelles des utilisateurs, ensuite les automatisations.

## Ressources


- William Wake, “*Twenty Ways to Split Stories”* http://xp123.com/xplor/split-summary/
- J.B. Rainsberger, “*Splitting stories: an example”* http://jbrains.ca/permalink/5
- Lasse Koskela, “*Ways to split user stories”* http://radio.javaranch.com/lasse/2008/06/13/1213375107328.html
- James Grenning “*Story Weight Reduction Toolkit”* http://www.renaissancesoftware.net/blog/archives/48





Traduction : Nicolas Mereaux – http://www.les-traducteurs-agiles.org