Bill Wake, 30 Sept. 2005

Traduit par Fabrice Aimetti, 8 Jan. 2013

20 manières de découper les stories

Découper les stories nous permet de séparer les parties qui présentent une forte valeur de celles qui ont une plus faible valeur, afin de pouvoir consacrer notre temps à traiter les parties de la fonctionnalités présentant la plus forte valeur. De temps en temps, nous devons procéder autrement, en combinant des stories pour qu'elles deviennent vraiment intéressantes. Il y a généralement une grande valeur ajoutée à obtenir une solution minimale de bout en bout, pour ensuite la compléter avec le reste. Ces « stratégies de découpage » sont là pour vous aider à le faire.

Vision d'ensemble		
Plus facile	*Plus compliqué*	*Pourquoi*
Recherche	Action	Il est plus facile de rechercher comment faire une chose plutôt que de le faire (sachant que le faire réclame d'effectuer toutes les recherches nécessaires pour terminer le travail). Donc, si une story est trop dure, un découpage possible est de consacrer un certain temps à rechercher des solutions.
Spike	Implémentation	Les développeurs peuvent ne pas être à l'aise sur la manière de faire quelque chose, ou sur les critères que vous pourriez utiliser pour découper une story. Vous pouvez « acheter » de l'apprentissage pour le prix d'un spike (une expérimentation spécifique et pratique sur un aspect précis du système). Un spike peut durer une heure ou un jour, rarement plus.
Manuel	Automatisé	S'il y a un processus manuel en place, il est plus facile de l'utiliser tel quel. Ce n'est peut-être pas *meilleur* mais c'est moins de travail à automatiser. Par exemple, un système de vente a besoin d'une vérification de solvabilité. L'implémentation initiale orientait ce type de demande vers un groupe de personnes qui réalisait le travail manuellement. Ce qui laissait la possibilité de livrer le système plus tôt; le système automatique de vérification de solvabilité serait développé plus tard. Et de toute façon ça n'a pas été du travail pour rien puisqu'il a toujours existé un processus manuel pour les cas aux limites.
Achat	Implémentation	Parfois, ce que vous voulez existe déjà. Par exemple, vous pouvez trouver un widget spécialisé qui coûte quelques centaines d'euros. Cela pourrait vous coûter beaucoup plus en le développant vous-mêmes.
Implémentation	Achat	D'autres fois, la solution « sur étagère » correspondra peu à votre contexte, et le temps que vous perdrez à la personnaliser aurait pu être consacré à développer votre propre solution.

Expérience utilisateur		
Plus facile	*Plus compliqué*	*Pourquoi*
Batch	En ligne	Un batch n'a pas à interagir directement avec l'utilisateur.

Utilisateur unique	Multi-utilisateurs	Vous ne traitez pas la situation problématique « où deux utilisateurs essayent de faire la même chose en même temps ». Vous pouvez aussi ne pas vous inquiéter des comptes utilisateurs, ni de la traçabilité des actions des utilisateurs.
API seulement	Interface Utilisateur	Il est plus facile de ne pas avoir d'interface utilisateur du tout. Par exemple, si vous testez votre capacité à vous connecter à un autre système, vous pouvez découper la story pour vous contenter de faire un test unitaire pour appeler les objets de connexion.
Interface utilisateur en mode texte ou par scripts	Interface Utilisateur	Une simple interface peut suffire à faire émerger les sujets critiques
Interface Utilisateur Générique	Interface Utilisateur Personnalisée	Dans certains cas, vous pouvez utiliser des widgets basiquement avant de vous amuser avec les différentes options possibles. En allant un peu plus loin, des choses comme les Naked Objects génère une interface utilisateur par défaut à partir d'un ensemble d'objets.

-ités		
Plus facile	*Plus compliqué*	*Pourquoi*
Statique	Dynamique	Il est plus facile de calculer quelque chose une seule fois plutôt que de s'assurer qu'elle a une valeur correcte à chaque fois que quelque chose change. Parfois, vous pourrez utiliser une approche intermédiaire : vérifier périodiquement s'il y a un besoin de mise à jour, mais ne le faites pas avant que l'utilisateur le demande.
Ignorer les erreurs	Gérer les erreurs	Même si cela représente moins de travail d'ignorer les erreurs, cela ne veut pas dire que vous devez ignorer les exceptions. Par contre, le code de reprise doit être minimisé.
Transitoire	Permanent	Essayez de vous concentrer sur le fait d'écrire des objets corrects sans vous inquiéter de la correspondance avec les données persistantes.
Faible fidélité	Haute fidélité	Vous pouvez décomposer certaines fonctionnalités selon la qualité du résultat. Par exemple, « on peut démarrer avec un appareil photo numérique avec 1 pixel blanc et noir, puis l'améliorer selon divers axes : 9 pixels, 256 pixels, 10000 pixels; couleur codée sur 3 bits, 12 bits, 24 bits; fiabilité des couleurs à 75%, 90%, 95%. » (William Pietri)
Peu fiable	Fiable	« La disponibilité totale coûte très cher. Rapprochez-vous en de façon incrémentale, en la mesurant au fur et à mesure. » (William Pietri)
Petite scalabilité	Grande scalabilité	« Un système qui fonctionne pour quelques personnes sur un volume raisonnable de données suffit. Ensuite, chaque étape est une nouvelle story. N'oubliez pas les tests de

		montée en charge ! » (William Pietri)
Moins de "-ités"	Plus de "-ités"	Il est plus facile de traiter les exigences non fonctionnelles plus tard. Une stratégie fréquente est de planifier des spikes dans des projets parallèles pour éprouver la stratégie sur l'architecture.

Fonctionnalités		
Plus facile	*Plus compliqué*	*Pourquoi*
Peu de fonctionnalités	Beaucoup de fonctionnalités	Il est plus facile de réaliser peu de fonctionnalités.
Flux principal	Flux alternatifs	Terminologie empruntée aux cas d'utilisation. Le chemin nominal, donc le chemin optimiste de référence, est généralement celui qui présente le plus de valeur. Si vous ne pouvez pas terminer la transaction la plus simple, alors qui va s'inquiéter de savoir que vous avez mis en place un super traitement de reprise lorsque l'étape 3 échoue ?
0	1	Les architectes hardware ont une règle "0, 1, infini", ce sont les trois valeurs les plus faciles à utiliser. Les cas spécifiques génèrent des problèmes de gestion de ressources.
1	Beaucoup	Il est généralement plus facile d'en avoir un de bon et de travailler ensuite sur la collection.
Condition pour découper	Condition totale	Considérez les "et", "ou", "alors" et autres prépositions comme des opportunités de découper. Simplifiez une condition, ou faites seulement une partie d'un enchaînement d'étapes.
Un niveau	Tous les niveaux	Un niveau est le cas de référence d'un problème à plusieurs niveaux
Cas de référence	Cas général	En général, le cas de référence doit être réalisé en premier (pour s'assurer qu'une solution récursive va bien se terminer).

Résumé

Ces "stratégies de découpage" vous donneront des idées pour trouver un moyen d'avancer par petites étapes. Même s'il est important d'être capable de découper ces stories, n'oubliez pas que vous aurez à les rassembler pour obtenir la fonctionnalité totale. Généralement, vous constaterez qu'il existe un chemin étroit, mais de très grande valeur, passant à travers votre système.

Rédigé pour la conférence XP Day, Sept. 2005.
6 Janv. 2006 : je remercie William Pietri pour ses suggestions.