Organisation pour le projet de spécialité
Passerelle Git-MediaWiki

Julien Khayat, Guillaume Sasdy, Simon Perrat, Charles Roussel, Simon Cathébras

Le 29 mai 2012

1 Description du projet et contexte

1.1 La bonne idée de *Git-MediaWiki*

Habituellement les pages web d'un wiki se modifient directement sur un navigateur web à l'instar de *Wikipédia*. L'année dernière une équipe d'élèves de l'école a développé *Git-Mediawiki* à l'initiative de Matthieu Moy. Cet outil permet de modifier les pages d'un wiki sous forme de fichiers et de répertoires gérés par le logiciel *open source* de gestion de version *Git*. Ainsi nous profitons des (divines) fonctionnalités d'un tel gestionnaire comme la possibilité d'avoir plusieurs versions d'un wiki à un instant donné.

1.2 La validation d'un projet dans la communauté *Git*

Avant de voir *Git-Mediawiki* ajouté à la version officielle de *Git*, la communauté doit s'assurer de la qualité du code développé et de l'absence de bug. Il y a plusieurs étapes : tout d'abord, un code intéressant sera ajouté à la branche *pu* et de nombreuses critiques sont émises ; une fois les critiques prises en compte et le code corrigé, celui-ci est ajouté à la branche *next* ; à partir de là, c'est en bonne voie pour que le mainteneur décide de l'ajouter à la branche *master* (version officielle de *Git*), souvent plusieurs mois après, le temps que le code "mûrisse". Aujourd'hui, le code de *Git-Mediawiki* est dans le répertoire contrib/ de la branche *master*.

1.3 Notre projet : développer l'environnement de test

Cette année, notre équipe développe un environnement de test pour *Git-Mediawiki*. Il y a deux raisons à cela.

1. Lors de développement informatique sérieux, tester le logiciel est au moins aussi long que le développer. Ça permet de s'assurer qu'on ne remet pas "une bombe" entre les mains de l'utilisateur. Notre base de test permettra à la communauté *Git* de s'assurer que *Git-MediaWiki* est un logiciel fiable ou, cas échéant, savoir que corriger pour qu'il le devienne. À terme, il serait bon que celui-ci soit intégré à la version officielle de *Git*.
2. Les développeurs qui se chargent d'ajouter des fonctionnalités pourront développer en *Test Driven Development*. Une méthode agile qui produit un code plus fiable et plus rapidement qu'avec les méthodes de développements classiques.

Notre travail implique donc des interactions avec la communauté mais aussi avec l'autre équipe qui travaille sur *Git-Mediawiki*, chargée d'ajouter des fonctionnalités à l'existant. Toutes deux, nous utilisons les méthodes agiles. Cela sera présenté en détails dans une autre section de ce document.

2 Objectifs et périmètre

2.1 Objectifs

Nous avons découpé notre projet en trois objectifs réalisables et mesurables grâce à aux critères de performance que nous détaillons également.

1. **Acceptation de notre environnement de tests par la communauté *Git***. Nous souhaitons que notre base de test soit sur la branche *next* (cf. ce qui est expliqué dans le contexte du projet). Un développement accepté sur la branche *pu*, sera considéré comme un objectif atteint à 50%
2. **Couverture complète du code de base de *Git-MediaWiki***. Le critère de performance de ce point est simplement le nombre de fonctionnalités qui sont testées et approuvées de façon satisfaisante.
3. **Conformité de nos tests avec les standards des tests *Git***. En effet, la communauté Git a adopté des standards pour l'écriture des tests du logiciel. Il nous faudra nous y conformer. La mesure de ce critère est moins objective, nous considérerons les retours de la communauté par rapport à la conformité.

Dans tous les cas, la meilleur référence pour juger de l'avancement de nos objectifs seront les avis de la communauté qui développe *Git* car ils sont des développeurs expérimentés.

2.2 Périmètre

2.2.1 L'environnement

L'environnement de test que nous allons mettre en place consiste essentiellement en des fonctions de manipulation de wiki. Nous avons également mis en place des fonctions de tests de contenu et de présence de fichiers. Ces fonctions sont utilisées pour faciliter la vérification de la cohérence entre un wiki de test et une archive *Git*.

1. Ajout/édition d'une page
2. Suppression d'une page
3. Récupération d'une page
4. Comparaison de deux fichiers
5. Recherche d'un fichier par son nom

2.2.2 Les tests

Les tests que nous allons écrire serviront dans un premier temps à valider le code déjà écrit. C'est à dire les fonctionnalités basiques de la passerelle comme le *clone* (transformer un wiki en un dossier *Git*), le *push* (transformer les fichiers du dossier *Git* comme pages du wiki), et le *pull* (transformer les pages du wiki en fichiers *Git*).

Une fois que ces fonctionnalités seront validées, nous planifions d'écrire des tests pour l'autre équipe de développement de la passerelle. Ce point est à la frontière du périmètre, c'est à dire que nous le réaliserons seulement si nos objectifs sont terminés dans les délais. Toutefois, si nous arrivons à ce stade, nous envisageons de corriger les bugs du code existant que nous parviendrons à détecter avec l'environnement de test. Il nous sera également possible de prêter main forte à l'équipe qui développe de nouvelles fonctionnalités pour *Git-MediaWiki*.

3 Organisation de l'équipe

3.1 Dates clef

Notre projet se déroule avec les principes des méthodes agiles, les dates importantes prévues sont les suivantes :

- Réunion de fin de sprint du Vendredi 25 Mai
- Réunion de fin de sprint du Lundi 4 Juin
- Réunion de fin de sprint du Lundi 11 Juin
- Rendu du rapport écrit le Jeudi 14 Juin
- Soutenance le Vendredi 15 Juin

Pour s'assurer que nous respectons ce planning du mieux possible, nous avons mis en œuvre plusieurs moyen agiles.

- Burndown
- Tableau de post-its de la méthode Scrum

3.2 Organisation au quotidien

Stand-up Meeting Chaque matin, l'équipe se réunit à heure fixe pour Stand-up Meeting. Chacun choisit les tâches qu'il remplira dans la journée, et signale à l'équipe si il rencontre des difficultés.

Réunion de fin de Sprint Cette réunion se fait une fois par sprint avec le Product-Owner. Cette réunion a pour but de réaliser une démonstration de ce qui a été réalisé dans le sprint écoulé.

Interactions Notre projet dépend de deux facteurs principaux. Notre product-owner, Mathieu Moy, d'une part. Et d'autre part la communauté de développement de Git.
Un objectif important dans notre projet est de réussir à interagir au mieux avec la communauté, essentiellement par mail. C'est en effet nécessaire si l'on veut voir notre projet retenu pour la prochaine release de Git.

3.3 Risques pour le projet

Voici une liste des principaux risques que nous avons identifiés pour notre projet, ainsi que des exemples de solution.

1. La communauté n'est tout simplement pas intéressée par notre projet.
2. Nos travaux sont envoyés trop tard pour bénéficier de retours.
3. L'équipe fonctionnalité est mise en retard pour réaliser des tests.

Solutions possibles :

1. Envoyer un mail au plus tôt avec des exemples sur nos travaux. Être actifs pour promouvoir notre projet.
2. Faire de l'envoi à la Mailing-List une priorité.
3. Choix d'un représentant auprès de l'équipe fonctionnalité. Ce poste est assuré par Charles Roussel.

4 Conclusion

Pour beaucoup d'entre nous, ce sera notre premier projet avec les méthodes agiles.
De plus, nous allons tous, pour la première fois, développer pour un logiciel *open source*.
C'est-à-dire soumettre notre code à des développeurs expérimentés et utiliser une mailing-list mais aussi se conformer à des standards existants.
L'autre grande valeur ajouté de ce projet de spécialité est que, pour la première fois, nous allons être en plein dans la validation de logiciel existant. Une activité non négligeable lors du développement que, jusque là, nous n'avons que très peu expérimenté lors de projet scolaire à l'ENSIMAG.