Rapport du projet de spécialité
Passerelle Git-MediaWiki, environement de test

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

Le 14 juin 2012

1 Travail réalisé

Le travail réalisé durant ce projet est un environnement de test pour la passerelle *Git-MediaWiki*, un projet initié l'année dernière à l'Ensimag.

1.1 La passerelle *Git-MediaWiki*

Cet outil a pour but de permettre de gérer un MediaWiki a l'aide de Git. Pour bien comprendre de quoi il retourne, nous allons expliquer précisément ce que sont Git et MediaWiki

1.1.1 MediaWiki : des sites communautaires à contribution libre

Le terme wiki désigne un site web dont les pages peuvent être éditées par les internautes qui le fréquentent, et MediaWiki est un type de wiki particulier. Wikipédia, le wiki le plus fréquenté actuellement, fonctionne avec MediaWiki, et on y retrouve toutes ses caractéristiques, notamment :

- chacun peut y contribuer de manière libre ;
- possibilité d'ajouter, supprimer, et surtout éditer des pages ;
- conservation de l'historique complet des modifications.

Le principal défaut de ces sites est que leur édition peut s'avérer fastidieuse, et plusieurs problèmes se présentent régulièrement aux gros utilisateurs :

- conflits entre éditions concurrentes ;
- la moindre petite modification de page, comme une correction orthographique, engendre une entrée dans l'historique ;
- une seule révision de page entraîne une révision complète du wiki, l'historique est donc rapidement surchargé ;

– ainsi pour une mise à jour importante (par exemple sur Wikipédia, un évènement d'actualité marquant), les pages touchées doivent être révisées une par une ;
– de plus, dans le cas d'une modification conséquente sur une page, on est confronté à un dilemme :
  – soit on enregistre régulière et on produit un historique "sale"
  – soit on n'enregistre qu'une fois le travail terminé, et on s'expose alors au risque de perdre son travail sur une coupure de connexion, un crash, une fermeture accidentelle du navigateur...

Le but de la passerelle est alors de proposer une solution à ces problèmes en permettant d'utiliser le gestionnaire de version *Git* pour travailler sur un MediaWiki.

1.1.2 Git : un gestionnaire de version

Un gestionnaire de version est un logiciel qui permet de partager et synchroniser un ensemble de fichiers entre plusieurs utilisateurs. Typiquement, un dépot est créé sur un serveur et toute personne qui y a accès peut le copier sur son ordinateur personnel, pour ensuite partager ses modifications sur les fichiers. *Git*, comme d'autres gestionnaires de version, propose en outre les fonctionnalités suivantes :

– possibilité d'avoir plusieurs versions d'un même fichier ;
– gestion des éditions concurrentes ;
– conservation d'un historique complet et détaillé.

On constate alors quelques avantages par rapport à un wiki :

– on peut éditer un fichier en plusieurs fois en faisant une seule mise en ligne, l'historique résultant est alors plus propre ;
– faire une mise à jour simultanée de plusieurs fichiers est plus simple ;
– aucun risque de perte de donnée en cours d'édition, car la plupart des éditeurs de texte ont un système de sauvegarde automatique.

1.2 L'environnement de tests

L'outil *Git-MediaWiki* est actuellement intégré à *Git* comme une contribution, sans être présent dans son coeur de commandes. Pour pouvoir aller plus loin et intégrer nativement la passerelle dans *Git*, elle doit être munie d'une base de test conséquente pour valider son fonctionnement. Jusqu'à présent, les tests devaient être effectués à la main.

L'objectif de notre groupe est de construire un environnement complet de tests pour la passerelle, avec pour objectif, à court terme de valider les fonctionnalités existantes et d'en développer de nouvelles, à long terme de pousser *Git-MediaWiki* jusque dans coeur de *Git*.

1.2.1 Un wiki sur localhost

Pour effectuer les tests de *Git-MediaWiki*, un premier pré-requis est de posséder un wiki installé à l'adresse (par défaut) http ://localhost/wiki. Pour éviter que les utilisateur aient à effectuer les pénibles étapes de création d'un wiki, l'équipe a décidé de fournir un script d'installation de wiki en local.

L'idée est de disposer ainsi d'un wiki connu avec lequel les tests peuvent interagir aisément. On fournit donc avec ce wiki un package complet de fonction pour manipuler ce wiki : ajout, supression et édition de page ; récupération d'une page et de son contenu ; comparaison d'une page et d'un fichier ; etc...

Le wiki ainsi créé dispose en plus d'un fonction `wiki\_reset` qui permet d'assurer qu'un wiki est vierge au début de chacun des tests de la batterie proposée.

1.2.2 Une Batterie de test

Cette partie de notre travail est pour ainsi dire le *coeur* de notre projet. Nous proposons à l'utilisateur une batterie complète de test pour *Git-MediaWiki* activée via la commande `Make test`. Ces tests sont conçu pour la valider le fonctionnement de *Git-MediaWiki*, avec 2 objectifs principaux.

1. Disposer d'une base de test de non régression pour les prochain contributeurs à la passerelle. De plus, disposer d'un environnement de tests complet permet d'utiliser facilement des méthodes agiles tel que le TDD (*Test Driven Developpement*).

2. Détecter et référencer les bugs actuels de la passerelle à l'aide des fonctions de la base de test de qui, et ainsi posséder une base solide du fonctionnement (ou non) de la passerelle selon les cas. De plus, si un développeur indépendant veut débugguer la passerelle, ces tests lui seront très utile pour savoir si oui ou non ses modifications sont efficaces.

1.2.3 Une amélioration de performance

En plus de l'environement de test, notre équipe a également développé une amélioration de performance pour *Git-MediaWiki*. Selon le type de wiki cloné, la passerelle actuelle peut parfois s'avérer très lente pour des opérations simple (*git pull*). L'idée de cette optimisation est de déterminer l'évolution du wiki depuis la dernière mise à jour du dépot Git, et de récupérer les modifications d'une manière différente selon celles ci. Actuellement, ce choix de méthode doit être effectué par l'utilisateur, on ne dispose pas encore de système automatisé de la solution à employer. On distingue 2 cas :

– Dans le cas ou on a plus de révisions (i.e modification) du wiki que de pages, on va procéder en récupérant la dernière mise à jour de chaque page. C'est la méthode présente nativement sur *Git-MediaWiki*. Celle ci peut poser des problèmes dans le cas d'un wiki avec beaucoup de pages et peu actif.

– L'autre possibilité est d'avoir moins de révision que de pages dans le wiki. Dans ce cas, la bonne méthode est non plus de regarder page par page, mais de consulter la liste des révisions et de ne récupérer que la plus récente pour chacune des page. Cette méthode est, au contraire de la précédente, très efficace pour des wiki peu actifs, mais va peiner sur des wiki plus "gros".

2 La communautée *Git*

Git est un logiciel libre et open source, ce qui signifie que tout le monde a accès à son code source et peut y contribuer en proposant des modifications. Il existe alors une communauté de développeurs contribuant à *Git*, qui communiquent ensemble via une mailing-list dédiée et échangent ainsi sur les bugs rencontrés et leur correction, les ajouts de fonctionnalités et les mises à jour du code de manière générale. La version officielle est maintenue par Junio C. Hamano, qui se charge de vérifier que les patches proposés ont un niveau de finition suffisant pour être intégrés, en plus d'être lui-même un des contributeurs principaux au code source de *Git*.

2.1 Organisation

Le code de *Git* est disponible librement sur une archive *Git*. Cette archive comprend trois branches, qui correspondent chacune à différents avancements des patches proposés. La principale, *master*, est celle qui correspond à la version officielle distribuée. La branche *next* contient les patches qui seront ajoutés à la prochaine version officielle. Enfin, la branche *pu*, pour *proposed updates*, comprend les patches qui sont en voie d'intégration à *Git*, mais qui ont encore besoin de peaufinage pour passer dans *next*. Il faut noter qu'avant même qu'un patch puisse passer dans cette branche, il doit déjà être relu, testé et approuvé par la communauté sur la mailing-list.

2.2 Interactions avec la communauté

Nos échanges avec la communauté se font via la mailing-list de *Git*. Nos propositions de patches se font en plusieurs temps : tout d'abord, nous envoyons sur cette liste de diffusion une 'cover letter', soit un mail où nous présentons notre équipe et notre projet. Lorsque nos premiers travaux nous semblent assez mûrs, nous envoyons alors un série de patches sur la mailing list, que la communauté peut alors relire et commenter. Nous tenons alors compte des critiques émises pour corriger notre code, et envoyons une nouvelle série de patches. Nous continuons ainsi jusqu'à obtenir un code qui atteigne les critères d'acceptation par *Git*, et qui soit en mesure d'être ajouté à la branche *pu*. Actuellement, nous n'avons pas atteint cette branche mais avons bon espoir d'y parvenir.

3 Difficultées pendant le projet

3.1 Premier sprint : lundi 21 mai → mardi 29 mai

Le premier sprint a été consacré à la découverte du projet et du contexte de travail. Nous espérions pouvoir rentrer rapidement dans le vif du sujet et envoyer un premier patch sur la mailing, mais les difficultés techniques et organisationnelles rencontrées nous ont forcés à reconsidérer cet objectif.

3.1.1 Difficultés techniques

Il nous a fallu comprendre comment était organisé *Git* et plonger dans le code du script *git-remote-mediawiki* pour comprendre son fonctionnement. Nous avons rencontré nos premières vraies difficultés lorsqu'il a fallu le tester sur nos machines personnelles, avec l'installation de *Perl* et de *MediaWiki* ; c'était toutefois indispensable pour bien en appréhender le fonctionnement et nous a été utile lorsqu'il a fallu écrire nos scripts d'installation.

3.1.2 Problèmes organisationnels

Pour la plupart des membres du groupe, il s'agissait de notre premier vrai projet utilisant une méthode agile, ce qui s'est accompagné de plusieurs soucis au sein de l'équipe :

- manque de communication dans le groupe ;
- mauvaise compréhension de certains points de la charte d'équipe par une partie du groupe ;
- plusieurs membres du groupe ont du s'absenter un après midi ou une journée pour des raisons personnelles ;
- pas d'évènement de cohésion de groupe.

Mis bout à bout, ces points ont conduit à un premier sprint assez désorganisé, ce qui nous a amenés à repenser notre fonctionnement, tant en terme de travail que de cohésion :

– stand-up meeting fixé à 9h35 chaque matin ;
– petit-déjeuner d'équipe ;
– utilisation systématique des post-it ;
– programmation par paire et relecture de code obligatoire.

Ces quelques règles nous ont permis de gagner significativement en efficacité lors du deuxième sprint.

3.2 Deuxième sprint : mardi 29 mai → mardi 5 juin

Notre priorité lors de ce sprint était d'envoyer un premier patch sur la mailing-list, afin d'avoir une idée de l'opinion que se fait la communauté de notre projet. Cet objectif clair associé à notre organisation nous a permi de travailler efficacement, et nous avons pu envoyer la première version de notre batterie de tests sur la mailing-list, puis prendre en compte les retours.

3.3 Troisième sprint : mardi 5 juin → mercredi 13 juin

Ce sprint s'est nettement moins bien passé que le précédent : malgré un début de sprint où nous sommes restés sur la lancé du précédent et avons continué à avancer, nous avons commencé à rencontrer des problèmes techniques que nous n'avions pas sérieusement appréhendés au début du projet, à savoir l'application de nos patches à d'autres machines que les nôtres avec les contraintes techniques associées. En plus de cela, nous avons eu des problèmes lorsqu'il a fallu intégrer nos travaux respectifs ensemble, et nous avons perdu énormément de temps à former une série de patch qui puisse être envoyée sur la mailing-list. En fin de sprint, nous nous sommes donc rendus compte d'une part que nous avions perdu certaines de nos bonnes habitudes du sprint précédent, d'autre part qu'il nous a manqué d'avoir certaines règles de codage qui auraient facilité l'intégration de nos travaux.

4 Bilan

Au niveau de l'équipe

Gestion d'équipe et méthodes agiles

Tout au long du projet, la vie de l'équipe a été marqué par l'empreinte des méthodes agiles. Ces techniques ont influencer notre manière d'apréhender le projet, de répartir les charges de travail, et de gérer l'équipe. Notre gestion a souvent un avantage, mais parfois aussi un handicape.

On retiendra que des règles de vie mal comprises par certains au début du projet ont ralentit le travail, alors qu'à la fin c'est la fatigue généralisé qui a empêché un bon fonctionnement de la communication de groupe. Mais ces problèmes ont gardé un impact réduit sur le fonctionnement de l'équipe.

Au final, les bilans de fin de sprint on permit de mettre à plat les problèmes et de leur trouver une solution convenant à l'ensemble du groupe.

On notera également que la pratique des méthodes agiles a été apréciée par l'ensemble du groupe, que ce soit ceux qui en avaient utilisé au projet GL, ou ceux qui pour qui c'était nouveaux. le Scrumboard aura été un élément de référence tout au long du développement, servant de point de repère sur l'avancement des tâches.

Le logiciel libre

Après une première approche du logiciel libre, l'équipe ressort de cette expérience avec un mélange de satisfaction et de frustration.

Satisfaction tout d'abord car ce premier contact nous a montré tous les avantages qu'apporte ce système, notamment grâce à la mailling-list qui permet de se lancer dans du code et garantit de bénéficier d'une revue de qualité. Malgrès tout, on retiendra que l'accès à la mailling-list peut s'avérer compliquée au profane.

Frustration devant un travail laissé partiellement inachevé, mais surtout devant le peut d'intéré porté par la communauté à nos travaux. Le peu de diversité parmis nos relecteurs nous a parfois poussé à nous demandé si notre projet valait vraiment le coup.

Au final cette expérience nous aura permit de découvrir à la fois les bon et les mauvais côté du logiciel libre.

Au niveau des enseignants

L'ensemble de l'équipe s'accorde à dire que l'enseignant de référence (Matthieu Moy) a été un très bon encadrant tout au long du projet : disponible, réactif sur les questions par mail et présent dans la salle. Il aura été d'une grande aide grace à sa connaissance des l'environnement du projet, c'est à dire la communauté Git et ses coutumes, le logiciel libre dans un sens plus général, mais également dans sa capacité à nous orienter dans la bonne direction en ce qui concerne l'architecture des fichiers.
Un autre point très positif du projet aura été l'application des méthodes agiles, qui ont permi à l'équipe de travailler à la fois plus sereinement et plus efficacement. Les passages régulier de Jean-François Jagodzinski nous ont incité à réfléchir à notre organisation, améliorant notre travail d'équipe tout au long du projet.

Améliorations

Après discussion dans l'équipe voici quelques pistes d'amélioration pour le projet dans les années à venir :

– donner des informations "technique" sur le projet sur la page de présentation. Il aurait put être interessant de connaitre les langages utilisés à l'avance.