Contribution à Software Heritage

Projet 5éme année

2019 -2020

DALAINE Nathan

FONTFREYDE Joachim

GAUFFIER Léni

GAUTIER-PIGNONBLANC Yann

Introduction

Présentation de Software Heritage

Software Heritage est un projet d'archive logicielle qui a pour but de collecter et préserver le code source des logiciels. Le projet est soutenu par l'UNESCO depuis début 2017. Il a été initié en 2014 par Roberto Di Cosmo et Stefano Zacchiroli. L'objectif étant d'archiver tous les codes sources disponible sur les différentes plateformes collaboratives comme Internet Archive souhaite archiver toutes les pages internet. L'archive à ouvert en 2016 et est soutenu par l'INRIA.

L'archive à ouvert officiellement le 7 juin 2018. À cette date, près de 7 milliards de fichiers sources pour 100 millions de projets sont archivés. On y retrouve tous les codes sources disponibles sur GitHub, l'histoire du développement Unix ainsi que le code source de la mission Apollo 11.

Le 25 septembre 2018, les dépôts de logiciels sur HAL ont également été connecté à Software Heritage.

Le projet

L'objectif du projet était de concevoir et d'implémenter des extensions au projet Software Heritage soutenue et d'autres acteurs du logiciels libres. Plusieurs exemple nous été proposé comme par exemple :

- développer des importateurs et analyseurs depuis d'autres forges : Dockerhub DF, Chef recipes, Ansible, Pupperlabs, ...
- Dataviz sur des métriques logicielles à la Openhub;

et également plusieurs propositions correspondant à des offres de stages proposé par software Héritage : https://wiki.softwareheritage.org/index.php?title=Category:Available_internship

Cependant nous nous somme rendu compte qu'il allait être difficile de commencer par des extensions aussi importante, nous avons donc commencé par choisir des taches "easy hack" proposé par la communauté sur leur Forge afin de prendre connaissance de l'architecture.

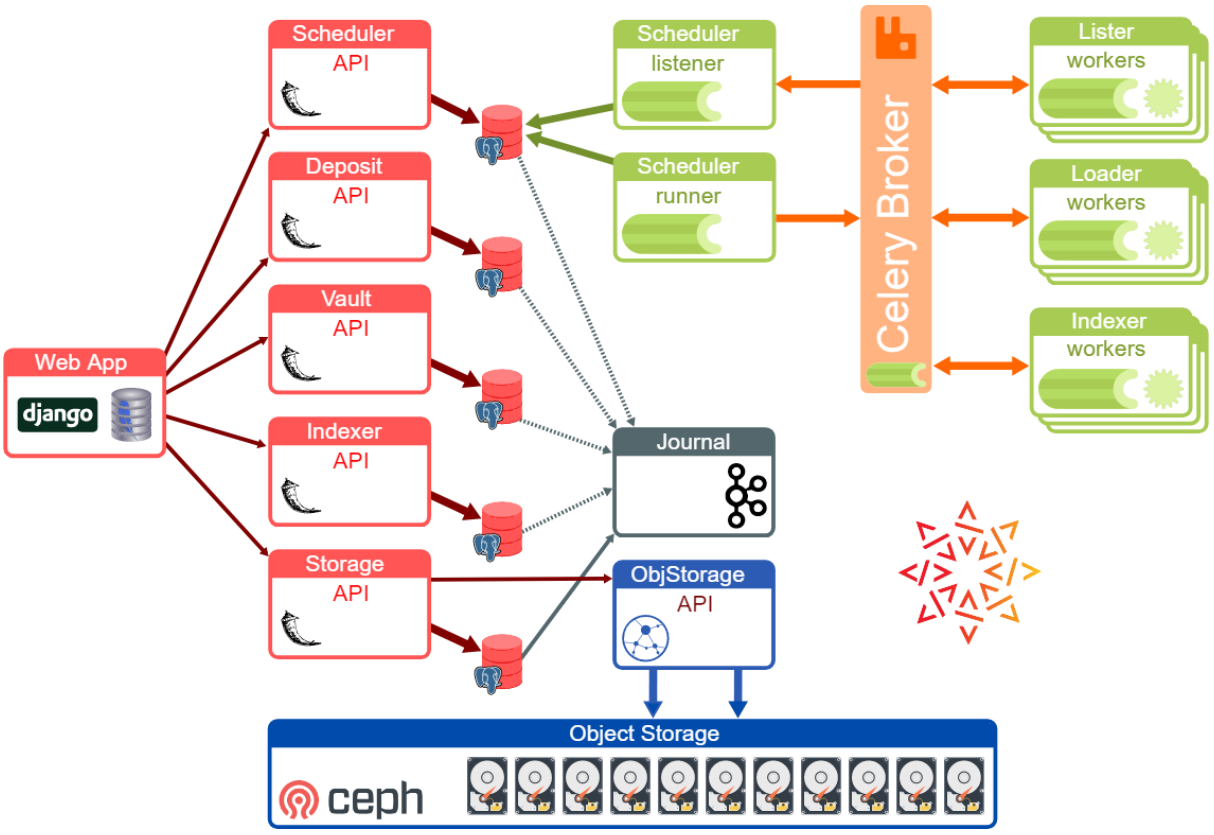
Nous n'avions donc pas de cahier des charges à proprement parlé, ou des tâches précises à réaliser, c'était à nous de choisir quelles extensions nous souhaitions développer.

Technologies employées

Software Héritage est développé en Python 3 et est divisé en un grand nombre de sous services (13 containers docker pour un lancement en production).

- Une architecture en microservice Docker (une dizaine au total), cette architecture trouve tout son intérêt sur un projet de ce type qui est très lourd avec de très nombreuse dépendances, puisque cela permet de lancer une instance de Software Héritage en production sans avoir à s'en soucier.
- L'interface web est développée en Django un framework web à l'architecture MVC gratuit et open-source écrit en Python.
- Afin de distribuer les tâches programmées un gestionnaire de file de tâches asynchrones est utilisé: Celery. Cet outil permet le partage de la charge de travail entre les différents workers. Ce choix a surement était motivé par le fait qu'il s'agit d'un projet mature qui possède déjà une intégration avec Django
- Les données importées sont stockées grâce à la plateforme Ceph, cette plateforme de stockage est totalement distribuée et ne présente pas de point unique de défaillance grâce à une réplication des données (l'un des clusters de réplication est à la bibliothèque d'Alexandrie).

Architecture technique



Source: Software Heritage

L'architecture est très complexe et la documentation n'est pas forcément à jour un véritable challenge de la comprendre. On accède aux différents services depuis une interface web qui communique aux différents services via des API REST.

Il existe trois types différents de tâches:

- Les Listers, qui ont pour but de récupérer tous les codes sources à partir d'un site internet ou d'une forge donnée. La plupart du temps ils fonctionnent en utilisant les API fournies par les hébergeurs de forge, il n'y a pas de "scrapping à la main".
- Les Loaders: seront appelés après les listers et vont eux s'occuper d'importer ou bien de mettre à jour le code source présent dans une forge donnée. Ils sont spécifiques à chaque outil de versioning utilisé, aujourd'hui Git, Mercurial et SVN sont supportés.
- Les Indexers: ce composant va en extraire des informations complémentaires. Pour ce faire différents Indexers sont appelés en parallèle et vont chercher des fichiers

connus comme par exemple package.json pour NPM, PKG-INFO pour Python… Les packages trouvés seront également importés dans Software Heritage.

Ces trois composants sont intimements liés et ne peuvent pas fonctionner l'un sans l'autre. Par exemple dans le cas de l'importation de l'ensemble des projets d'un hébergeur de forge, une première tâche de lister est programmée. Cette tache appellera elle même l'API de gitlab afin de récupérer les adresses des forges de tous les projets puis les ajouteras dans la base de donnée de Software Heritage via l'API de storage.

Une seconde tâche de type Loader est alors ajoutée à la queue, celle-ci va télécharger le contenu (code source, historique des révisions…) et l'ajouter à la base de données. Puis finalement une tâche de type indexer sera programmée afin d'extraire les metadatas du projet nouvellement importé.

Un modèle définit la façon dont les projets importés sont stockés dans la base de données, sont donc conservés: les répertoires, les commits, les tags (releases), les blobs. Des informations sont également ajoutées: l'origine qui représente l'url du repository qui a été cloné, les snapshots qui représentent l'état actuel (commit) des branches au moment de l'importation et un champ visit qui représente le moment auquel l'importation d'un projet à eu lieu, chaque visit est lié à une snapshot.

Réalisations techniques

Tâches "Easy hack"

Nous avons tout d'abord commencé ce projet par une revue des tâches taggées *easy hack* dans la forge Phabricator. Ces tâches ainsi annotées par la communauté indiquent une certaine facilité et un point d'entrée adéquat pour les développeurs nouvellement intéressés par le projet SH. Notre objectif a été d'en choisir quatre pour mettre le pied à l'étrier rapidement.

Parmi la quinzaine de tâches disponibles beaucoup portaient sur l'annotation du code Python. En effet le langage Python est un langage à typage dynamique c'est à dire que le type d'une variable est induit par l'assignation. Ce genre de typage est pratique pour ne pas avoir à se soucier de la déclaration d'un type à l'avance. En revanche lorsqu'un programme se complexifie il devient dur de suivre quel genre d'entrées et de sorties une fonction quelconque va nécessiter. Ces informations pouvaient déjà se trouver dans les commentaires, mais la mise à jour Python 3.5 a formalisé une notation permettant d'annoter les variables et fonctions. Cette version est sortie en 2015 et a dû attendre plusieurs années pour être réellement adoptée, il y a donc beaucoup du code de Software Héritage qui a été développé avant le passage à cette version et aujourd'hui la communauté essaie de pallier les manques d'annotations. Étant un bon moyen d'explorer le code, nous avons choisi deux tâches d'annotations.

Une autre tâche consistait à ajouter des compteurs dans le storage component afin de garder un décompte précis des éléments ajoutés à la base de données. A l'origine seul le nombre de snapshots/répertoires etc étaient pris en compte ces champs sont ajoutés à un tableau et récupérés grâce à leur hash. Cependant trois champs ajoutés à ce même tableau

n'ont pas de hash associé: person, skipped_content, origin_visit ils n'étaient pas donc pas décomptés. Nous les avons ajoutés aux compteurs.

Un dernier easy hack réalisé a été l'ajout d'un endpoint de l'API swh-graph, un outil de visualisation de la représentation en graphe de l'archive Software Heritage. Il s'agissait d'améliorer l'endpoint /randomwalk qui effectue une traversée de graphe d'un noeud source à un noeud destination et renvoie les noeuds traversés. Il existait un endpoint /randomwalk/last qui renvoyait le dernier noeud traversé (avant celui de destination), l'objectif était de généraliser cela pour avoir les N derniers noeuds (ou les N premiers). Nous avons donc remplacer /last par un paramètre de requête ?limit=N avec N $\in \mathbb{Z}$, à la manière de l'indexation Python une valeur négative renverra les N derniers noeuds et une valeur de N positive renverra les N premiers noeuds. Enfin 0 est la valeur par défaut et renvoie tout.

Nouveau lister pour la plateforme Launchpad

Notre principale contribution au projet Software Heritage a été le nouveau lister pour une nouvelle forge logicielle : Launchpad. Cette forge est développée est maintenue par Canonical Ltd. qui est également mainteneur de Ubuntu. Ainsi le code source d'Ubuntu est tout d'abord développé et hébergé en utilisant cette plateforme. Canonical est également à l'origine d'un logiciel de gestion de version appelé Bazaar, Launchpad a donc énormément de projets développé avec Bazaar et non Git. À l'heure actuelle Launchpad héberge publiquement plus d'un million de branches Bazaar et 17000 repos Git.

Notre intérêt pour ce lister vient d'une tâche ouverte il y a près d'un an en mai 2019. Sur cette tâche un contributeur Google Summer of Code était assigné et il proposait certaines solutions, vinrent ensuite un groupe d'étudiant de master de l'Université de Montpellier qui avaient également des propositions sur ce lister en particulier. Mais en juillet la discussion prit fin et aucun code n'avait été proposé. Au début nous pensions pouvoir nous appuyer sur les études et propositions des différents contributeurs qui avaient participé à la discussion. A priori le stagiaire et le groupe de master avaient le même constat : Launchpad proposait une API HTTP ainsi qu'une librairie Python launchpadlib et il serait intéressant d'utiliser l'un et l'autre pour arriver à nos fins.

Mais intéressons nous d'abord sur ce que fait réellement un lister, quels sont les prérequis pour qu'il fonctionne et comment utiliser l'existant pour en développer un nouveau. Un lister c'est un package de swh.lister qui contient : une classe lister (ici ce sera LaunchpadLister), un modèle SQLalchemy décrivant les informations qu'un projet listé doit avoir, et une ou plusieurs tasks Celery qui sont celles qui seront effectivement utilisées par SH. Notre LaunchpadLister doit avoir une fonction run(), c'est celle là qui sera exécutée par les tasks Celery et qui contient tout le mécanisme de listing.

Pour prendre un exemple concret attardons nous sur le GithubLister. Au moment de sa conception il est important de se demander quelles informations le fournisseur nous propose et de quelle manière ces données sont structurées. Nous pouvons voir sur https://developer.github.com/v3/repos/#list-public-repositories que Github fournit un endpoint de son API listant tous ses dépôts en les indexant et les paginant selon un ID incrémental, on voit également que le lien vers la prochaine page est donnée dans les headers. Avec ça GithubLister peut utiliser le mixin IndexingHttpLister, cette classe est destinée au genre de lister qui utilisent des requêtes HTTP pour recevoir des listes de dépôts indexés. En

implémentant IndexingHttpLister il suffit de spécifier la transformation de la réponse en données ingestibles selon le modèle défini et indiquer la page suivante à requêter. Il est donc très simple de réaliser un nouveau lister suivant un schéma similaire, c'est d'ailleurs l'exemple donné dans le tutoriel de la documentation Software Heritage. IndexingHttpLister est également implémenté par les listers de Bitbucket et Phabricator. Il existe d'autres mixin prêts à l'emploi tels que PageByPageLister et SimpleLister. Toutes ces classes sont dérivées de ListerBase qui implémente toutes les fonctions liées à la base de données.

C'est en utilisant IndexingHttpLister que les contributeurs à la discussion pensaient développer ce nouveau lister. Or si l'on regarde de plus près ce que propose Launchpad on se rend vite compte que cette manière de faire est bancale et que Launchpad ne détaille à aucun moment l'indexation de ses réponses aux appels API. La proposition de base était de lister tous les projets grâce à un appel à l'API web https://api.launchpad.net/devel/projects puis de trier ceux qui seraient des projets Git et ensuite de lister tous les dépôts Git se trouvant dans ces projets grâce à un appel d'une fonction de leur librairie launchpadlib. Nous n'étions pas convaincus par cette méthode qui impliquait d'utiliser une indexation non fiable, c'est pourquoi nous avons décidé de contacter directement les développeurs de Launchpad pour avoir leur avis sur la question.

Nous sommes donc allés sur l'IRC des développeurs de Launchpad pour leur demander des détails concernant leur API. Ils étaient tout de suite très enthousiastes à l'idée de nous aider à implémenter ce lister dans Software Heritage. Par chance ils se trouvaient pendant cette semaine en congrès et ils purent aussi en discuter physiquement. Pour notre première interrogation sur l'indexation la réponse fut claire : il ne faut pas s'y fier. Ils nous ont par la suite expliquer en détail comment Launchpad marchait et que leur schéma de données était plus complexe que des projets contenant des dépôts, par exemple le code d'Ubuntu était hébergé en dehors des projets et aurait été omis par la solution initiale. Ainsi toute la discussion préalable sur la task par les autres contributeurs était caduque. Notre interlocuteur principal fut Colin Watson, il nous proposa de développer un nouvel endpoint de launchpadlib qui listerait tous les dépôts Git. C'est exactement ce qu'il nous fallait pour mener à bien ce lister.

Le nouveau lister consiste finalement d'appels à cette librairie uniquement et n'utilise donc pas de requêtes HTTP. Il a donc fallu construire par nous même le lister avec comme seul point d'appui ListerBase.

Les dépôts sont récupérés grâce à l'appel launchpad.git_repositories.getRepositories(order_by='most neglected first', modified_since_date=threshold), l'ordre de sortie est en fonction de l'activité du dépôt. Les premiers sortis sont les dépôts ayant été modifiés il y a le plus longtemps. On peut spécifier à partir de quand on recherche ces dépôts, ainsi en prenant la date de modification du dernier dépôt de la collection et en rappelant la fonction on peut itérer sur la collection.

Notre fonction run() comporte un paramètre max_bound qui permet de spécifier à partir de quelle date elle va commencer. La fonction run s'occupe d'ingérer les dépôts git itérativement de max_bound jusqu'à la fin et de mettre à jour la base de donnée en conséquence.

Avec cette configuration nous avons pu concevoir 3 tâches :

- IncrementalLaunchpadLister : tâche de base qui lance simplement run() à un max_bound spécifié.

- FullLaunchpadLister : qui utilise IncrementalLaunchpadLister avec comme max_bound None, ce qui va itérer sur la collection entière.
- NewLaunchpadLister : qui utilise IncrementalLaunchpadLister avec comme max_bound la dernière date modifiée maximale des entrées déjà présentes dans la base de donnée. Ainsi on n'a pas à réitérer sur des dépôts dont on sait qu'ils n'ont pas été modifiés.

À l'heure actuelle le lister est fonctionnel et à l'état de code review sur Phabricator. Les tests sont à modifier pour mocker les appels à launchpadlib, une fois effectué le lister devrait pouvoir être déployé en production.

Gestion de projet

Organisation de l'équipe

Pour ce projet nous étions une équipe de quatres étudiants :

- Joachim Fontfreyde→ Chef de Projet
- Nathan Dalaine → Développeur
- Léni Gauffier → Développeur
- Yann Gautier→ Développeur

Méthodes

Comme expliqué précédemment, nous devions participer à un projet open source, la plus part des méthodes de gestion de projet que nous avions appris pendant notre cursus n'était donc pas adapté.
Nous avons dû découvrir comment une communauté open source fonctionne. Pour cela nous avions accés à la forge de Software Héritage où toutes les tâches proposés par la communauté sont disponible.
Le processus pour participer à une tache était le suivant :

Nous commencions par lire et étudier les tâches proposées afin de déterminer lesquelles étaient les plus adaptés en fonction de notre connaissance du projet et de nos compétences.

Une fois la tâche choisit, nous avions besoins de discuter avec la communauté afin d'avoir plus de précisions sur le code espéré lorsque la description n'était pas assez précise, ce qui

était souvent le cas. Ces discussions durées souvent plusieurs jours car nous devions attendre les réponses de la communauté.

Lorsque nous avions une idée plus précise de ce qui était attendu nous pouvions commencer le développement de la tâche.

Après avoir passé les différents tests d'intégrations, nous avions de nouveau une discussion avec la communauté afin de déterminer les points à améliorer selon eux. Une fois de plus cet echanges pouvait durer plusieurs jours.

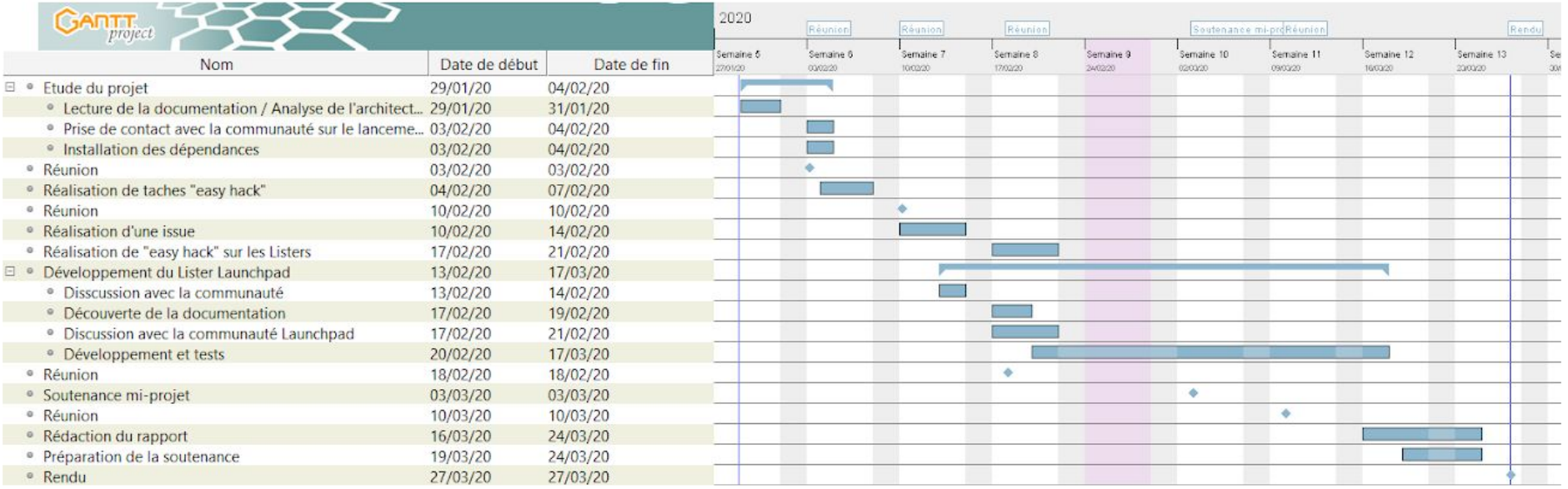
Si le résultat était satisfaisant, la tâches était clôturée, sinon il fallait reprendre le développement, repasser les tests et continuer la discussion jusqu'à arriver à ce que la communauté souhaitait.

Nous avons également mis en place plusieurs méthodes au sein de notre groupe:

Daily stand up : Nous nous sommes réunis régulièrement afin de discuter de l'avancement des tâches de chacun et des difficultés rencontrés, en cas de blocage, le groupe pouvait s'entraider plus facilement.

Pair Programming : Nous avons également décidé dès le début du projet de mettre en place du pair programming afin de pouvoir avancer plus rapidement dans la compréhension du projet en discutant avec son binôme. Cela nous a également permis de passer moins de temps sur du code review avant d'envoyer le code à la communauté qui elle même contrôle ce qui est proposé. Ce qui permettait d'éviter des échanges peu utiles sur des erreurs de codes qui pouvait nous prendre beaucoup de temps.

Planning



Nom	Date de début	Date de fin
Etude du projet	29/01/20	04/02/20
Lecture de la documentation / Analyse de l'architect...	29/01/20	31/01/20
Prise de contact avec la communauté sur le lanceme...	03/02/20	04/02/20
Installation des dépendances	03/02/20	04/02/20
Réunion	03/02/20	03/02/20
Réalisation de taches "easy hack"	04/02/20	07/02/20
Réunion	10/02/20	10/02/20
Réalisation d'une issue	10/02/20	14/02/20
Réalisation de "easy hack" sur les Listers	17/02/20	21/02/20
Développement du Lister Launchpad	13/02/20	17/03/20
Disscussion avec la communauté	13/02/20	14/02/20
Découverte de la documentation	17/02/20	19/02/20
Discussion avec la communauté Launchpad	17/02/20	21/02/20
Développement et tests	20/02/20	17/03/20
Réunion	18/02/20	18/02/20
Soutenance mi-projet	03/03/20	03/03/20
Réunion	10/03/20	10/03/20
Rédaction du rapport	16/03/20	24/03/20
Préparation de la soutenance	19/03/20	24/03/20
Rendu	27/03/20	27/03/20

Outils

Pendant ce projet , nous avons utilisés plusieurs outils à des fins différentes, dans cette sections nous allons expliquer leurs rôles:

Outils de développement

Visual studio code : Nous avons décidé de tous utiliser le même IDE afin de faciliter la collaboration au seins du groupe, VS Code étant celui que nous utilisons régulièrement, nous nous sommes naturellement tournés vers lui.

Outils de collaborations

Forge Software Heritage:

La forge de Software Héritage à été l'outils de collaboration principale au cours de notre projet. Une forge permet de rassembler des projets et des développeurs. Mais la plupart des forges permettent aussi à des personnes ne pratiquant pas la programmation informatique de participer, par exemple les traducteurs ou les graphistes, ou les utilisateurs qui s'entraident dans des forums ou soumettent des rapports de bogues. Une forge permet donc de rassembler tous ces gens autour du développement de projets de logiciel.

Dans notre cas, elle nous permettait d'explorer les tâches proposées par la communauté afin de choisir les plus adaptés à notre projet, c'est également sur cette plateforme, via un chat, que nous pouvions échanger avec les membres de la communauté Software Héritage sur les problèmes rencontrés ou pour demander des précisions sur le résultat à fournir.

Phabricator :

Phabricator a été l'outil utilisé dans notre projet afin de gérer le versionning. C'est une suite d'outil qui gère entre autre git, et contient notamment Differential qui permet de faciliter la review de code. Chaque fois qu'une partie de code a été testé et validé en local, il faut la soumettre à la communauté afin qu'elle soit discuté et validé c'est ce qu'on appel le pré-commit. Pour cela il faut créer un Differential qui va présenter l'ensemble des modifications faites au codes et permettre aux différents membres de mettre des commentaires afin de discuter de la pertinence des modifications. Ensuite si des modifications sont nécessaires elles sont faites en locales puis elles sont push sur le même Differential. Lorsque le code est accepté il faut ordonner ses commits afin que l'historique soit propre et enfin merge son travail avec la branch master pour finalement push.

Outils de tests

Jenkins : il s'agit d'un outil d'intégration continue et open source qui peut être utilisé pour automatiser toutes sortes de tâches liées à la construction, au test et à la livraison ou au déploiement de logiciels.

Métriques

Au cours de ce projet nous avons travaillé à plusieurs sur l'ensemble des tâches, en raison des va-et-vient avec la communauté le nombre de lignes de code final n'est pas très représentatif du travail fourni. Notre intermédiaire privilégié avec la communauté était Leni Gauffier c'est pourquoi la majorité des commits ont été faits en son nom.

- https://forge.softwareheritage.org/D2873
- Lister launchpad
    - https://forge.softwareheritage.org/T1734
    - https://forge.softwareheritage.org/D2799
- https://forge.softwareheritage.org/D2770
- https://forge.softwareheritage.org/D2681
- https://forge.softwareheritage.org/D2669
- https://forge.softwareheritage.org/D2644
- https://forge.softwareheritage.org/D2636

Conclusion

Au cours de ce projet nous avons beaucoup appris sur l'univers open-source, cependant nous ne pensons pas que ce type de projet soit réellement adapté à un projet de fin d'études car une grande partie du projet a été consacré à la discussion avec la communauté. Nous avons également trouvé dommage de ne pas pouvoir avoir de réel encadrement (avec un membre de Software Héritage par exemple) au cours du projet afin de nous aiguiller sur des tâches réalisables.