M1 Informatique AIGLE

HMIN204

Conduite de Projet

Méta-Rapport du TER

Groupe Bajonim

Bachar Rima,
bachar.rima@etu.umontpellier.fr
Joseph Saba,
joseph.saba@etu.umontpellier.fr
Tasnim Shaqura,
tasnim.shaqura@etu.umontpellier.fr

Responsable de l'UE :
Eric Bourreau

22 mai 2019

Table des matières

Chapitre 1

Sujet

Les logiciels sont actuellement omniprésents dans tous les aspects de notre vie quotidienne ; archiver leurs codes source paraît ainsi une tâche primordiale. En effet, le code source d'un logiciel constitue un artefact logiciel fondamental dans le domaine des connaisances scientifiques, culturelles, et techniques. D'autre part, le code source est facilement lisible et compréhensible par les humains, et peut être transformé en fichiers exécutables. À ce titre là, des plateformes ont déjà été proposées, telles que `The Internet Archive` et `UNESCO Persist`. Toutefois, ces plateformes se concentraient plutôt sur la préservation des fichiers exécutables.

`Software Heritage` est un projet visant la préservation du code source des logiciels *open-source*, publiquement accessibles par quiconque. L'architecture de `Software Heritage` consiste en un *framework* permettant de retrouver les codes source publiquement accessibles depuis n'importe quel plateforme d'hébergement et de les ingérer dans l'archive de `Software Heritage`. En particulier, les **Listers** en constituent une partie centrale ; il s'agit de crawlers configurés pour parcourir des dépôts de code source et reporter leur ingestion à d'autres composants de l'architecture. En outre, les plateformes d'hébergement embarquent le code source dans des dépôts à structures différentes, ce qui nécessite la création d'un **Lister** dédié pour chaque plateforme. Actuellement, tous les **Listers** développés ont été créés uniquement par l'équipe de `Software Heritage`. Les **Listers** ont été développés pour les plateformes d'hébergement les plus populaires (`Github`, `Bitbucket`, ...).

Dans le cadre de ce projet, encadré par Jessie Carbonnel, nous avons cibler la plateforme d'hébergement `Launchpad` afin de lui développer son **Lister** correspondant. Ainsi, les objectifs de ce TER peuvent être énumérés de la manière suivante :

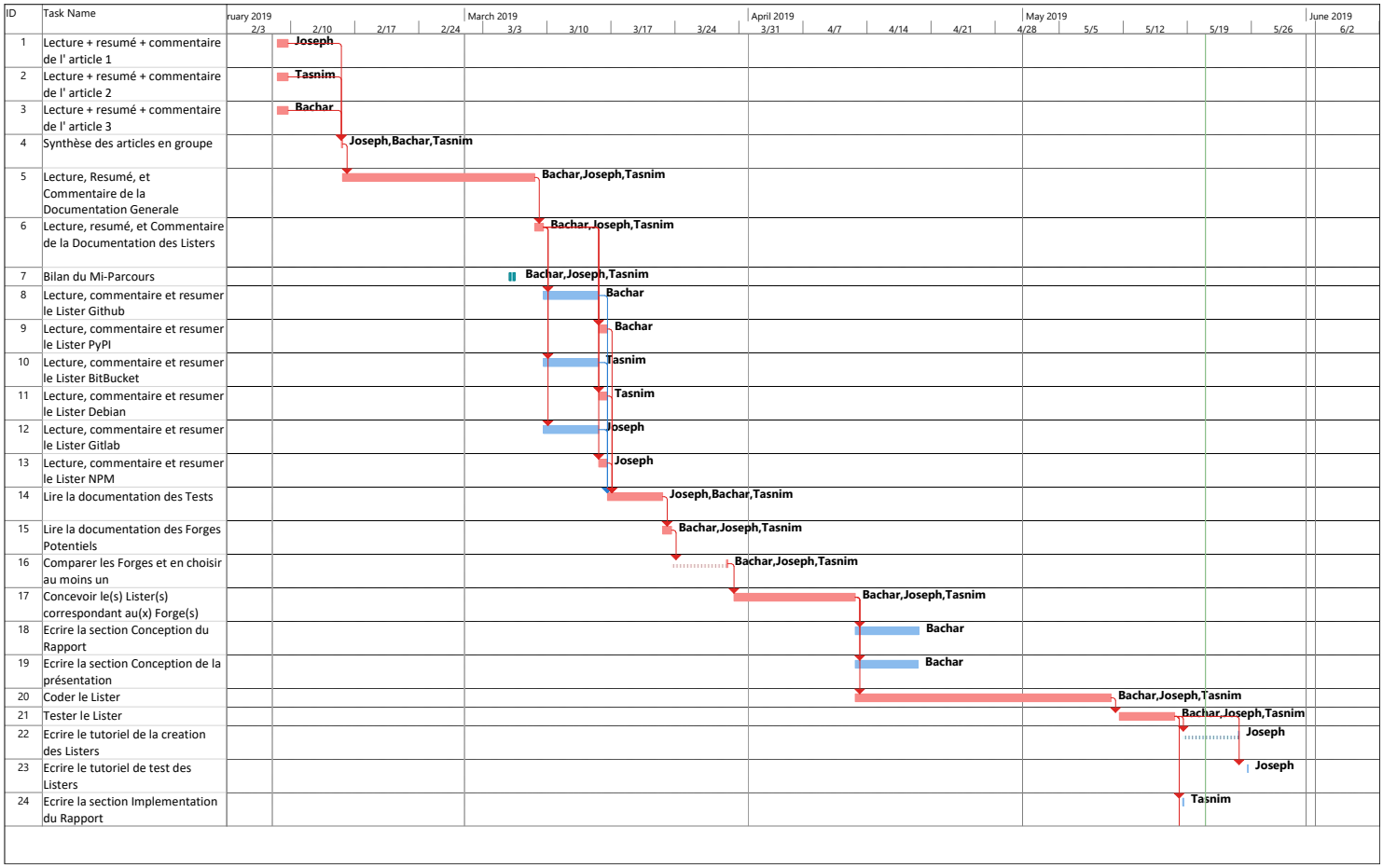
- Lire/comprendre les articles et tutoriels écrits par l'équipe de `Software Heritage` ;
- Analyser différentes plateformes d'hébergement afin d'en cibler une ;
- Concevoir et développer un **Lister** pour la plateforme choisie ;
- Répliquer localement l'environnement de `Software Heritage` afin de tester le **Lister** développé ;
- Faire une *Pull Request* afin d'intégrer le **Lister** testé au dépôt de développement de `Software Heritage` sur `GitHub`.

Chapitre 2

Planning

Dans ce chapitre, nous présentons les versions prévisionnelle et finale du **diagramme de `Gannt`**, ensuite nous en élaborerons un comparatif.

2.1 Planning Prévisionel



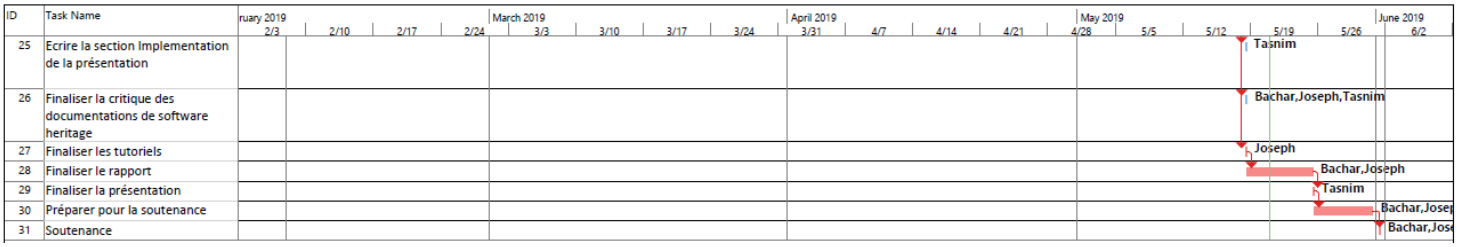
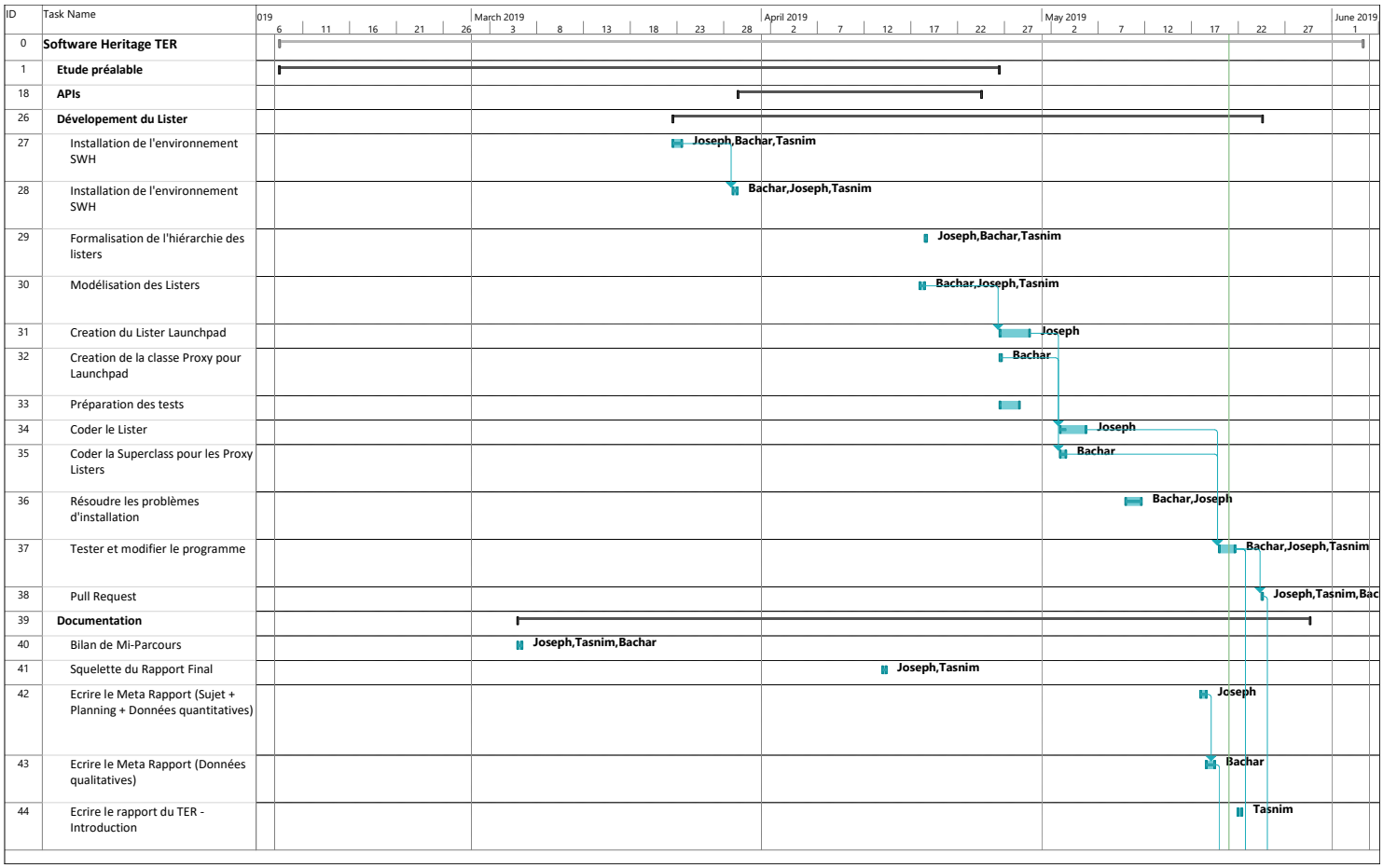


FIGURE 2.1 – Planning prévisionel

2.1.1 Notes sur le planning prévisionnel

Lors de la mise en place du **diagramme de** `Gannt`, la définition des tâches était trop généraliste et leur regroupement explicite était absent.

2.2 Planning Final



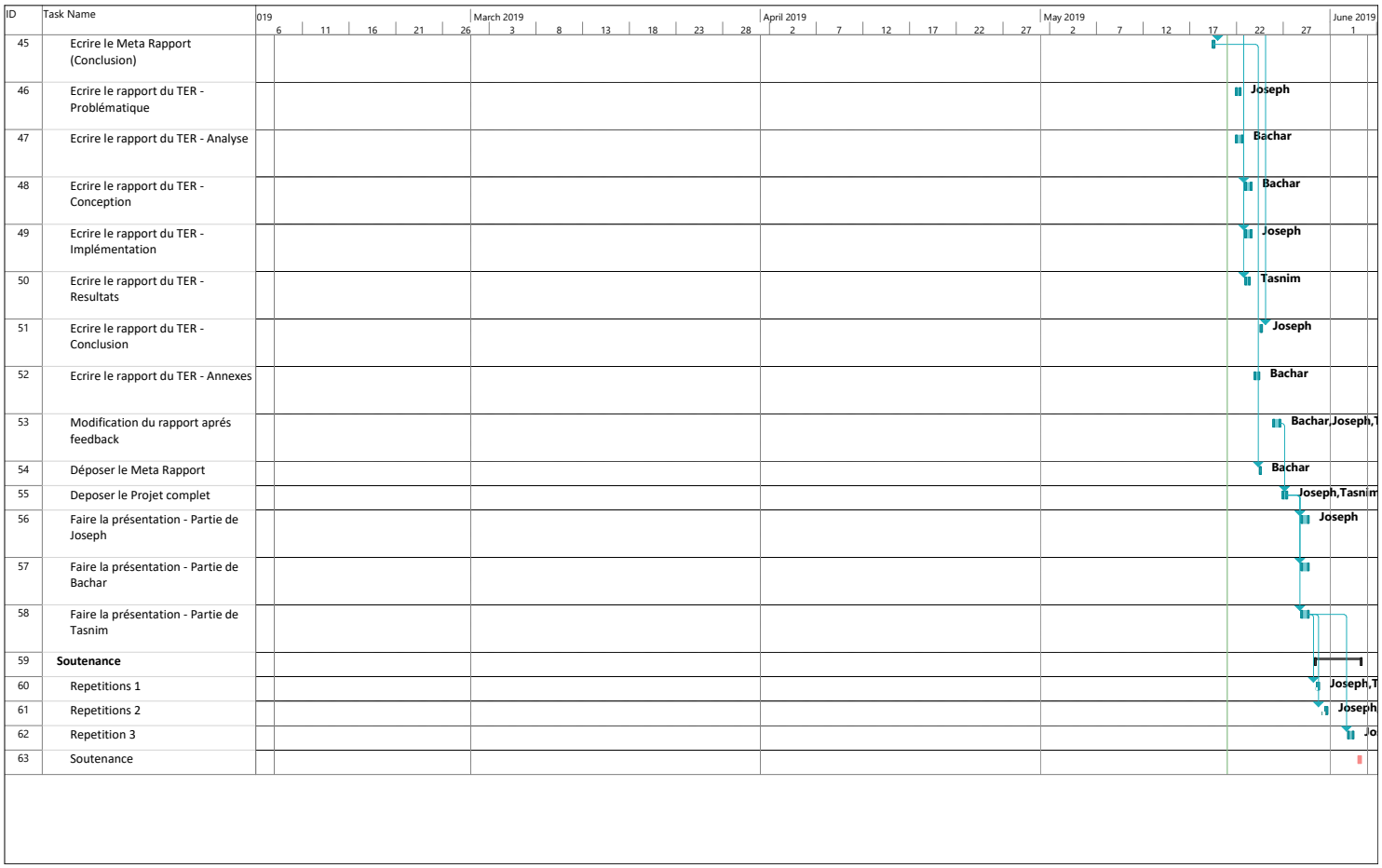


FIGURE 2.2 – Planning final

2.2.1 Notes sur le planning final

Au fur et à mesure de notre avancement, pour mieux refléter la structure du projet et pour être cohérent avec la théorie acquise au sein du module de conduite de projet, nous avons regroupé nos tâches de la manière suivante :

Étude préalable : la plus grande section du projet. Elle regroupe toutes les tâches de lecture et de recherche qui ont permis de nous familiariser avec l'architecture et l'environnement de `Software Heritage` en général, et l'implémentation des différents **Listers** prédéfinis en particulier.

APIs : la section regroupant les tâches de recherche d'une plateforme

d'hébergement de code ayant une **API** adaptée à nos besoins et qui sera exploitée par les tâches de développement.

Lister : la section regroupant les tâches de conception, développement et test du **Lister**.

Documentation : la section regroupant les tâches de rédaction des rapports et de la présentation pour la soutenance.

Soutenance : la section regroupant les tâches d'entraînement pour la soutenance.

2.3 Comparaison des plannings

La date de délai du planning prévisionnel occurrant assez tôt par rapport au lancement du module TER, nous n'avions pas eu le temps de récolter les informations nécessaires pour raffiner la répartition des tâches. Par conséquent, n'ayant pas une estimation concrète des différentes phases du développement du livrable, nous nous sommes contentés d'adopter une approche initiale généraliste en vue de la spécialiser au fur et à mesure de notre avancement.

Après, la récolte continuelle d'informations nous a permis de raffiner notre planning prévisionnel. Effectivement, certaines tâches ont été spécialisées et d'autres ont été ajoutées, par exemple :

tâche spécialisée : « *Coder le lister* » spécialisée en « *Création du ProxyLister* `Launchpad` », « *Création de la classe Proxy pour* `Launchpad` », « *Création de la superclasse des ProxyListers* », et « *Création de la superclasse des Proxy* » ;

tâche ajoutée : « *résoudre les problèmes d'installation de l'environnement de* `Software Heritage` ».

Ces adaptations sont majoritairement dues aux raisons suivantes :

— la durée de la recherche d'une plateforme d'hébergement de code ayant une **API** adaptée à nos besoins, imprévisible sans comprendre le fonctionnement de `Software Heritage` permettant de formuler les requis nécessaires ;

— la maintenance et la mise à jour de la plateforme `Software Heritage`, nous empêchant d'estimer correctement la durée des tests ;

— le croisement des dates de délai d'autres UE [1], avec celles définies pour le TER, nous empêchant de suivre minutieusement le planning prévisionnel défini.

1. méritant eux aussi un temps conséquent pour les compléter

2.4 *Features et issues*

Actuellement, le **Lister** a été conçu, implémenté et intégré à l'architecture d'une instance locale de `Software Heritage`. Malheureusement, la plateforme étant actuellement en maintenance et non disponible pour un usage local, nous n'avons pas encore pu exécuter les tests unitaires définis. Toutefois, après avoir communiqué avec l'équipe de développement de `Software Heritage` via un canal IRC dédié, nous nous sommes renseignés sur la date de reprise d'activité de la plateforme. Par conséquent, nous essayerons, d'ici la date de délai de notre travail de TER, de finaliser les tests, une fois la plateforme ait repris son activité.

Enfin, nous ferons un *Pull Request* auprès du dépôt `GitHub` du développement de `Software Heritage`, afin d'intégrer notre **Lister** à leur projet ; nous serons ainsi la première équipe communautaire (*i.e. hors l'équipe de développement de* ***`Software Heritage`***) à avoir effectué une contribution.
Nous estimons environ 9 à 10 heures de travail pour finaliser ces deux étapes.

Chapitre 3

Organisation

3.1 Stratégie de développement et répartition des tâches

3.1.1 Analyse du risque

Afin d'élaborer une stratégie de développement performante et pertinente, nous avons commencé notre démarche par l'analyse des risques associés au projet. Par conséquent, nous nous sommes retrouvés avec un profile de risque normal. En effet, notre équipe étant composée de trois personnes a largement minimisé les problèmes de transfert de connaissance. D'autre part, les difficultés liées à l'apprentissage des technologies avaient des conséquences non négligeables sur le temps d'adaptation de l'équipe au fonctionnement de `Software Heritage`. Par ailleurs, le projet n'étant pas créé « *from scratch* » a diminué suffisamment sa taille et son temps de développement.

Toutefois, les risques étaient surtout externes. D'une part, ils étaient liés à l'interruption continuelle de la plateforme de `Software Heritage` pour sa maintenance, étant un projet assez récent et toujours en plein évolution, et dont nous dépendons afin d'effectuer nos tests. D'autre part, nous étions largement contraignés par les **API**s fournies par les plateformes d'hébergement candidates, quelques unes inadaptées à nos requis, d'autres insuffisamment documentées et d'autres instables afin d'être exploitées.

3.1.2 Modèle de développement

Afin de clarifier notre approche de développement, nous nous sommes servis des diagrammes fournis par `Software Heritage` et permettant de comprendre l'architecture et le comportement générique des différents composants. En particulier, nous nous sommes appuyés sur un **diagramme de séquence** décrivant le fonctionnement d'un **Lister**. D'autre part, nous avons développé des **diagrammes de classes** et des **modèles ER** afin de résumer les classes des composants concernés et leurs schémas de base de données correspondents.

Par conséquent, nous nous sommes aperçus que notre projet consiste à développer quelques composants au sein du *framework* défini par `Software`

`Heritage`, et nous nous sommes ainsi contentés d'un **modèle de développement en V**. Ceci est justifié par le fait que le *framework* embarquait déjà la majorité du *Base code* nécessaire ; nous nous sommes ainsi trouvés avec peu de code à rédiger. Cependant, notre charge de travail était principalement concentrée sur l'obtention d'une conception cohérente et extensible du **Lister**, facilitant son intégration à l'architecture de `Software Heritage` et son éventuelle extension par d'autres composants à développer ultérieurement.

3.1.3 Répartition des tâches

Afin d'augementer la productivité et d'accélérer le progrès, la répartition des tâches s'est effectuée par spécialisation en fonction des disponibilités et des compétences des membres du groupe.

3.2 Réunions et Travail

Afin de mettre en évidence notre progrès, discuter des éventuelles difficultés rencontrées, et décider des prochaines étapes, nous organisions des réunions hebdomadaires (ou presque) avec/sans notre encadrante `Mme` Carbonnel. Tout au long du projet, notre planning était en évolution continuelle, même si les modifications y introduites n'étaient pas forcément radicales d'une réunion à l'autre. De plus, à des fins de pilotage, nous avons eu recours à des traçes sous forme de documents *meeting minutes*. Chaque *meeting minute* contenait les informations liées au déroulement et les résultats de la réunion correspondante. Ceci nous a permis de créer une sorte d'un tableau de bord distribué et dont le contenu était chronologiquement ordonné.

Enfin, chaque membre du groupe travaillait principalement seul chez lui. En outre, nous nous sommes embarqués parfois dans la bibliothèque de la faculté, dans une salle de projet, ou même chez l'un des membres afin de travailler ensemble.

3.2.1 Outils

Gantt chart : `MS Project` ;

UML : `draw.io` ;

communication : un groupe dédié sur `WhatsApp` ;

gestionnaire de version : `Git` lié à un dépôt `GitHub` ;

pilotage : réunions et leurs *meeting minutes* correspondants ;

documentation : LaTeX.

Chapitre 4

Données quantitatives

Dans ce chapitre, nous présentons quelques details quantitatives concernant le dépôt `GitHub` associé au projet TER.

4.1 Commits

Depuis la date de création du dépôt `GitHub`, jusqu'au 22 mai 2019, nous avons effectué 105 commits.

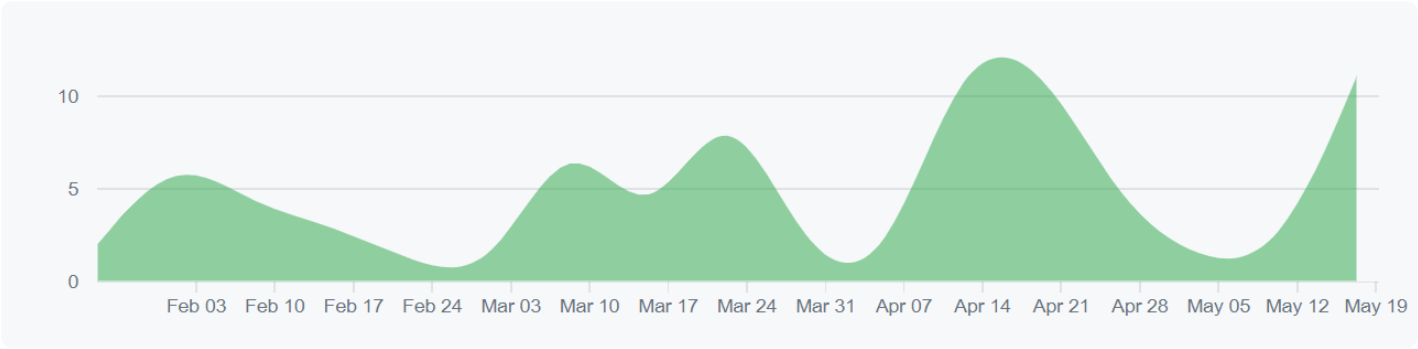


FIGURE 4.1 – Graph des commits

Dans la figure 4.1, nous observons, au délà de la préparation et de l'organisation du dépôt, que le nombre de commits était bas pendant la période de fin de février et début de mars. Ceci est dû au fait qu'il s'agit de l'étude préalable consistant surtout en des tâches de lecture.
Ensuite, nous constatons la présence d'une autre période de baisse de commits (fin mars et début avril), due aux périodes de rattrapage du premier semestre et de recherche d'une plateforme d'hébergement à **API** adaptée.
Après, les commits augmentent tout au long le mois d'avril où nous entamons la conception et l'implémentation du **Lister**, et puis la rédaction des rapports et de leurs documents associés.
Enfin, les commits baissent encore une fois lors de la période de préparation aux examens finals (fin avril et début mai) pour augmenter vers la fin de mai, avant la date de délai du TER et de ses livrables correspondant.

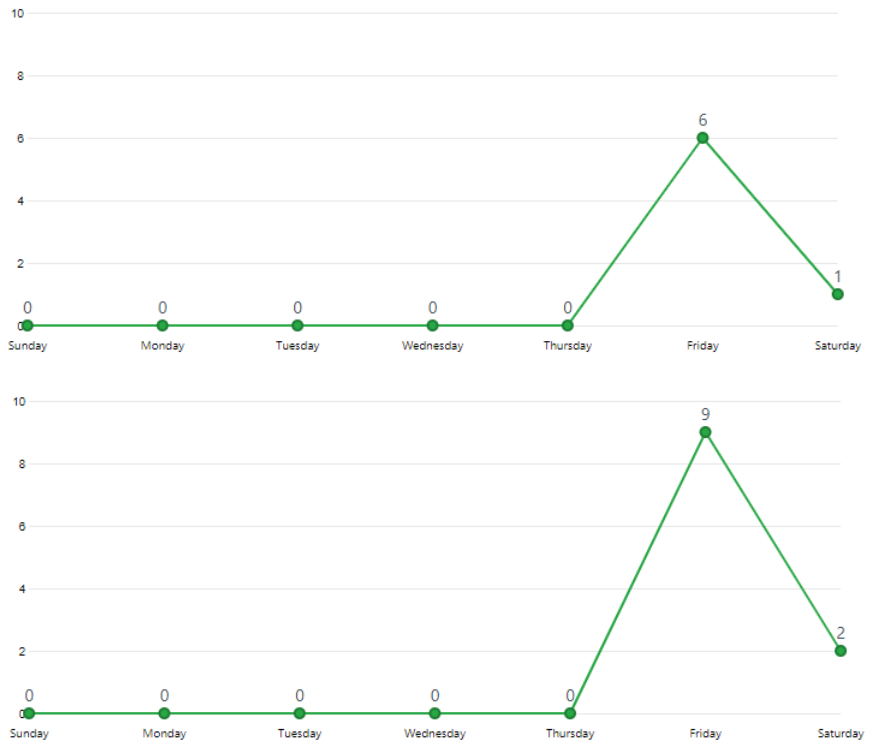


FIGURE 4.2 – Moyenne des commits par jours pour les semaines du 3 février et du 24 mars

Dans la figure 4.2, nous observons que les commits étaient souvent effectués les vendredis. En effet, nous avons decidé de se réunir et de travailler ensemble tous les vendredis, le seul jour de la semaine où nous étions tous les trois disponibles simultanément.

4.2 Classes et lignes de code

Classe	Lignes de code	Description
LaunchpadProxy	178	classe exploitant l'**API** de `Launchpad` via un objet proxy désignant une instance du client officiel de `Launchpad`, afin de récupérer tous les dépôts **Git** publiquement disponibles.
LaunchpadGitModel	10	classe définissant le modèle d'un dépôt **Git** hébergé sur `Launchpad` au sein de `Software Heritage`.
LaunchpadGitLister	50	classe définissant le **Lister**, implémentant toutes les fonctions requises pour énumérer les dépôts **Git** hébergés sur `Launchpad`.
LaunchpadGitTasks	48	classe définissant les différentes manières dont **LaunchpadLister** puisse lister les dépôts **Git** hébergés sur `Launchpad`, et encapsulant les tâches de *scheduling* et de *loading* responsables de l'ingestion de leur contenu dans l'archive de `Software Heritage`.
LaunchpadListerTester	21	classe définissant les tests unitaires à effectuer sur le **Lister**.
LaunchpadTaskTester	89	classe définissant les tests unitaires à effectuer sur les tâches du **Lister**.
ProxiedLister	10	classe abstraite désignant la **superclasse** de n'importe quel **Lister** à intégrer à `Software Heritage`, possédant les mêmes exigences que celui de `Launchpad`, (*i.e. intégrant un proxy qui agira comme intermédiaire entre l'**API** et le **Lister***).
WebApiProxy	49	Une interface contenant les **signatures des méthodes** requises par la classe d'un **proxy** utilisé par un **ProxiedLister**.

4.3 Temps de Travail

Concernant le temps de travail, nous estimons (approximativement) :

- 110 heures sur l'étude préalable du projet ;
- 60 heures pour la recherche de plateformes d'hébergement de code et pour tester leurs **API**s ;
- 110 heures pour le developpement du code ;
- 20 heures pour la documentation et les rapports ;

Nous estimons encore environ 30 heures de travail, portant sur l'écriture du rapport du TER et de la présentation.

Chapitre 5

Conclusion

Dans ce projet, nous avons essayé d'appliquer les techniques proposées par le module de conduite de projets, afin de baliser et maîtriser le projet le plus possible. En particulier, nous avons créé un planning prévisionnel en utilisant un diagramme de Gantt, nous avons évalué le profile de risque de notre projet et nous avons concrétisé une stratégie de développement complète.

En outre, le fait que nous n'avions été introduit à la méthodologie de développement agile qu'après avoir commencé le module TER, nous a empêché de l'adopter dès le début. Par ailleurs, notre émulation d'un tableau de bord via les *meeting minutes* était assez utile, certes, mais nous aurons dû la remplacer par un tableau de bord digital tel que celui offert par l'application `Trello`.

En définitive, le projet nous a rapporté une expérience conséquente en terme de gestion de projets par application directe des notions apprises en cours, mais également en terme de nouvelles technologies assimilées.