ENSICAEN
École Nationale Supérieure d'Ingénieurs de Caen

Projet AutoMix Software

Rapport de mi-parcours

Automix Software

Auteurs : Maxime S., Guillaume H., Jordan E., Louis C., Pierre G.

Tuteurs : Baptiste Hemery, Estelle Cherrier

ENSICAEN
6, boulevard Maréchal Juin – CS 45 053 – F- 14050 Caen Cedex 4
Tél. +33 (0)2 31 45 27 50
Fax +33 (0)2 31 45 27 60

Table des matières

Introduction

Dans le cadre du projet de deuxième année, nous avons choisi de proposer un projet innovant à destination du grand public.

Ce projet a pour but de produire un logiciel qui permettra à des utilisateurs de pouvoir générer un mix audio de qualité à partir d'une liste de musiques donnée (Figure 1), sans que ces derniers n'aient besoin de connaissances dans le domaine musical. De plus, le logiciel sera facile à prendre en main, l'interface devra donc être intuitive.

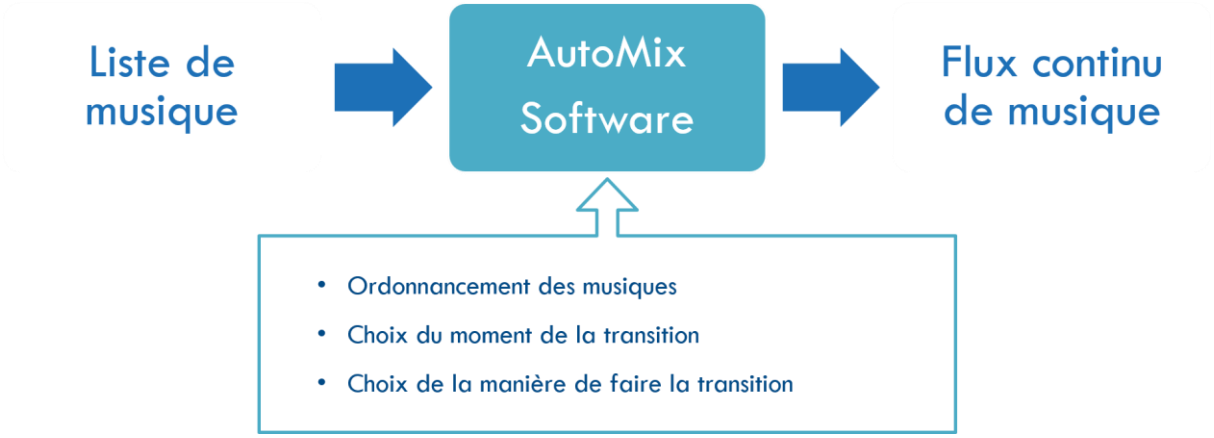

Figure 1 Processus de génération d'un mix audio

L'accomplissement de ce projet nous permet de mettre en pratique et de perfectionner des connaissances que nous avons apprises lors de notre cursus, comme le C++ ou l'intelligence artificielle. Mais cela nous donne aussi l'opportunité de découvrir l'environnement Microsoft .NET ainsi que le C++/CLR. De plus, c'est pour nous l'occasion d'appréhender les outils de travail collaboratif de manière plus poussée ou encore certaines méthodes de gestion de projet.

1 Organisation

1.1 Méthodes agiles

Nous suivons une méthodologie de développement agile reposant sur des sprints de deux semaines avec présentation d'un prototype à chaque fin de sprint.

Nous avons subdivisé le projet en un ensemble (non exhaustif) de *user stories*, c'est-à-dire tous les scénarios d'utilisation possibles. Les *user stories* sont par exemple : « L'utilisateur peut redimensionner la fenêtre du logiciel comme il le souhaite » ou encore « L'utilisateur peut choisir un dossier contenant ses musiques ».

A partir de la liste des *user stories*, nous pouvons définir les produits suivants :

- Le produit minimal présente les fonctionnalités basiques indispensables exigées par le client.
- Le produit intermédiaire présente, en plus du produit minimal, la plupart des fonctionnalités jugées importantes.
- Le produit abouti présente, en plus des produits précédents, des fonctionnalités augmentant le confort d'utilisation et la qualité du mp3 produit.

A ces user stories nous associons des fonctionnalités et des tâches, qui nous permettent de répartir le travail. Pour contrôler l'avancement du projet une traçons une courbe Burn-Up[1] dont le principe est expliqué dans la partie 1.2. Cette méthodologie nous permet d'avancer avec le plus de constance et de rigueur possible et d'identifier rapidement et clairement les problèmes que nous devons résoudre.

1.2 Présentation des outils

Nous développons notre logiciel en C++ avec l'environnement de développement Microsoft Visual Studio à destination de la plateforme Microsoft .NET. Le .NET Framework est un ensemble de composants logiciels structurels, permettant le développement d'applications. Il est distribué par Microsoft, à destination, principalement, des systèmes d'exploitation Windows. Il inclut une machine virtuelle gérant l'exécution d'applications .NET, d'outils de développement comme une bibliothèque de classes (Framework Class Library) et offre une inter opérabilité des langages.

Pour organiser nos *user stories* et nos tâches, nous utilisons Trello (Figure 2), un outil en ligne, que l'on peut décrire comme étant un tableau contenant des cartes. Ces cartes décrivent des tâches ou des *user stories* et peuvent être affectées à des membres, ou être ornées d'une description ou de commentaires.

[1] Voir Annexe1 : Courbe Burn-Up

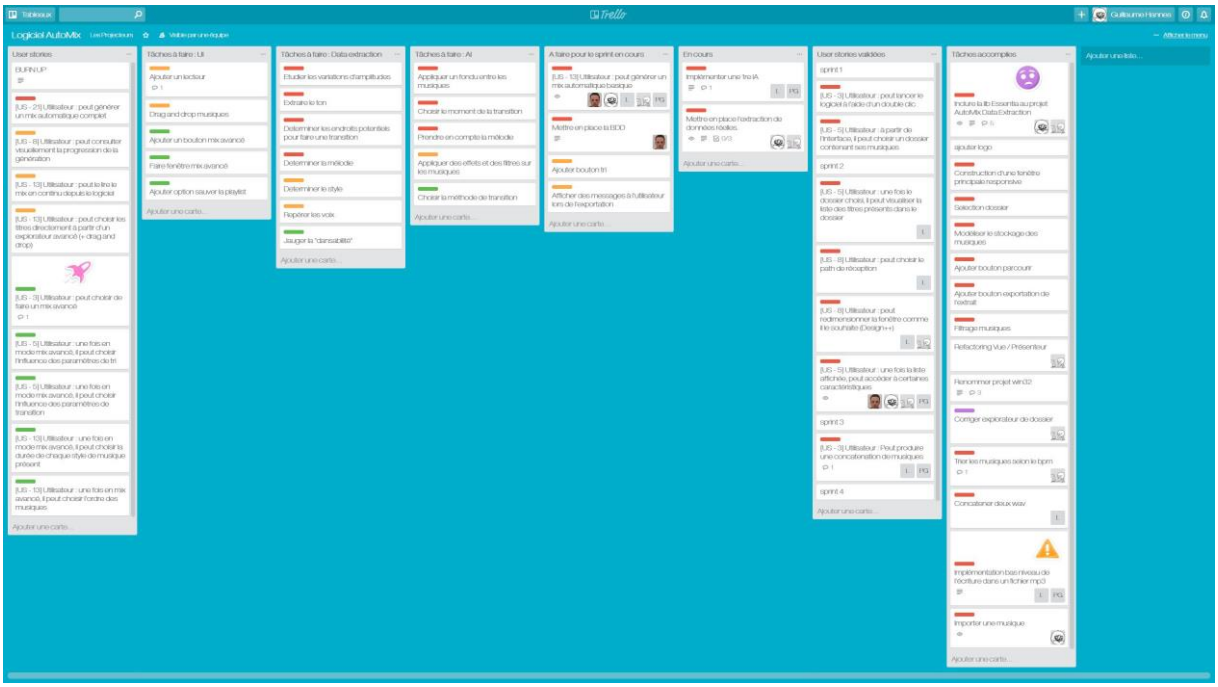

Figure 2 Capture d'écran de notre Trello

Etroitement lié au Trello, nous mettons à jour à chaque sprint le Burn-Up. C'est un graphique qui nous aide à évaluer notre progression par rapport aux objectifs fixés et dont le fonctionnement est le suivant :

- On affecte à chaque *user story* un certain nombre de points, attribués en fonction de l'importance vis-à-vis du client.
- Ensuite, on trace 3 courbes théoriques entre le début du projet, i.e. aucune *user story* de faite, jusqu'au produit minimal, au produit intermédiaire et au très bon produit.
- A chaque réunion de fin de *sprint*, on calcule la somme des user stories effectuées puis on place un point sur le graphique.

Ces outils nous apportent un retour visuel sur nos sprints pour évaluer notre avancement et remédier à d'éventuels problèmes. Nous pouvons également préciser que la plupart des membres de l'équipe utilisent SourceTree, distribué par Atlassian, afin de gérer leur dépôt git local et les échanges avec le dépôt distant situé sur le serveur GitLab de l'ENSICAEN.

1.3 Répartition du travail

Nous avons légèrement sous-estimé la quantité de travail lors du kick-off meeting. Nous estimons à environ 200h le temps de travail cumulé actuellement, ce qui correspond à 55% du temps initialement attribué au projet. En règle générale, nous formons des binômes assignés à une tâche pendant un sprint. Les tâches sont attribuées selon les connaissances et les préférences de chacun, certains développeurs se sont spécialisés dans un domaine, mais les méthodes agiles nous permettent d'être assez souples vis-à-vis de la répartition des tâches. Nous avons passé l'essentiel de notre temps à essayer d'intégrer différentes librairies de gestion audio et à étudier des solutions pour remédier aux difficultés rencontrées, détaillées dans la partie 3.1.

2 Développement logiciel

2.1 Architecture

Comme nous l'avions annoncé lors du kick-off meeting, nous accordons beaucoup d'importance au génie logiciel. Dans cette partie nous n'entrerons pas en détail dans les patrons de conception employés, mais nous allons présenter l'architecture globale de notre projet.

Le découpage en modules est vital pour le développement en équipe, mais aussi pour isoler totalement les fonctions qui proviennent de solutions externes. En effet, l'objectif est de présenter une interface commune, puis au moyen d'un proxy nous pouvons changer l'implémentation réelle de l'extraction de données. Ce mécanisme est important pour pouvoir effectuer la stratégie expliquée dans la partie 3.2.

La Figure 3 illustre notre architecture, au-dessus de la ligne pointillée, les modules sont exécutés par la machine virtuelle .NET, alors qu'en dessous, les modules sont exécutés par le processeur de la machine directement. Les flèches quant à elles, représentent les dépendances entre les différents modules.

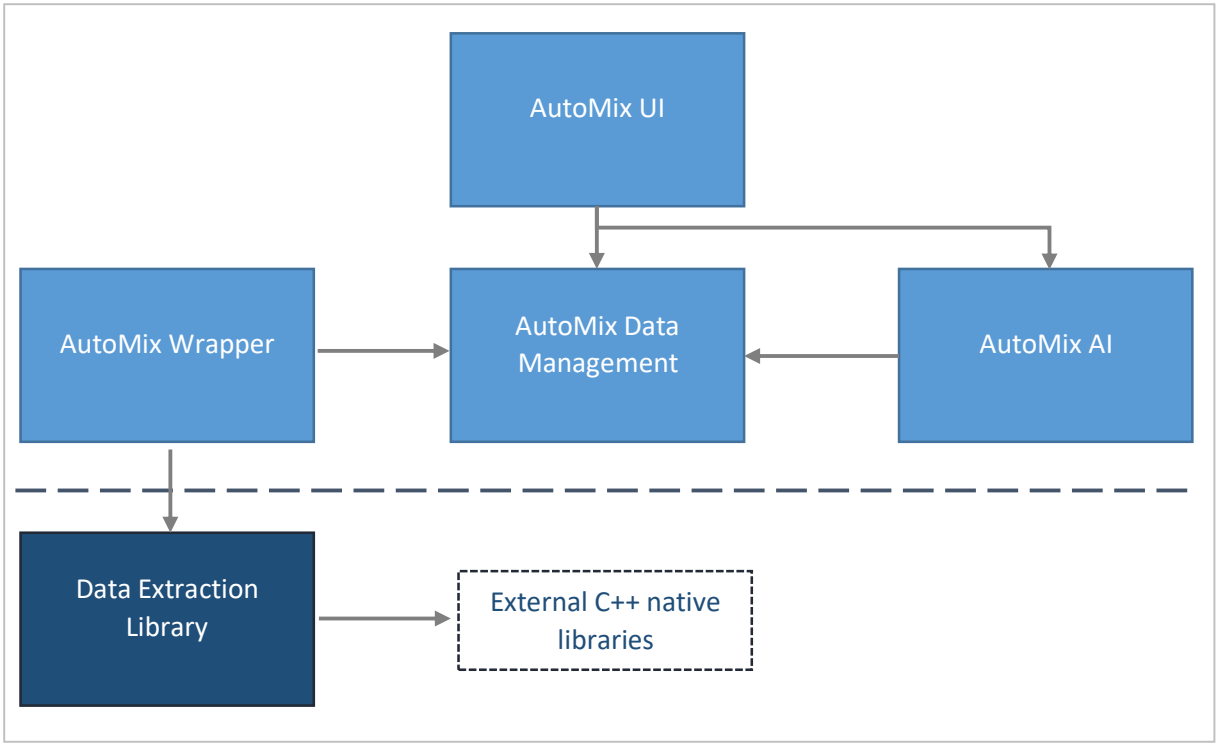

Figure 3 Architecture d'AutoMix Software

L'architecture que nous avons présentée ici sera amenée à évoluer au cours des prochains *sprints*. Notons que même si les modules sont cloisonnés et peuvent fonctionner indépendamment de l'interface graphique, ils ne seront pas distribués indépendamment sous forme d'API comme nous avions pu l'évoquer par le passé.

2.2 Git et intégration continue

Nous utilisons GitLab comme outil gestionnaire de versions couplé à la stratégie du Git Flow (Figure 4). Il s'agit d'un workflow basé sur le développement de fonctionnalités et la structure de ses branches en fait sa particularité.

Git Flow fournit un modèle de ramification défini comme suit :

- La branche *master* contient le code stable délivré.
- La branche *develop* contient le code stable pour l'intégration de fonctionnalités.
- Les branches *feature/** sont utilisées pour coder une fonctionnalité spécifique sur le projet.

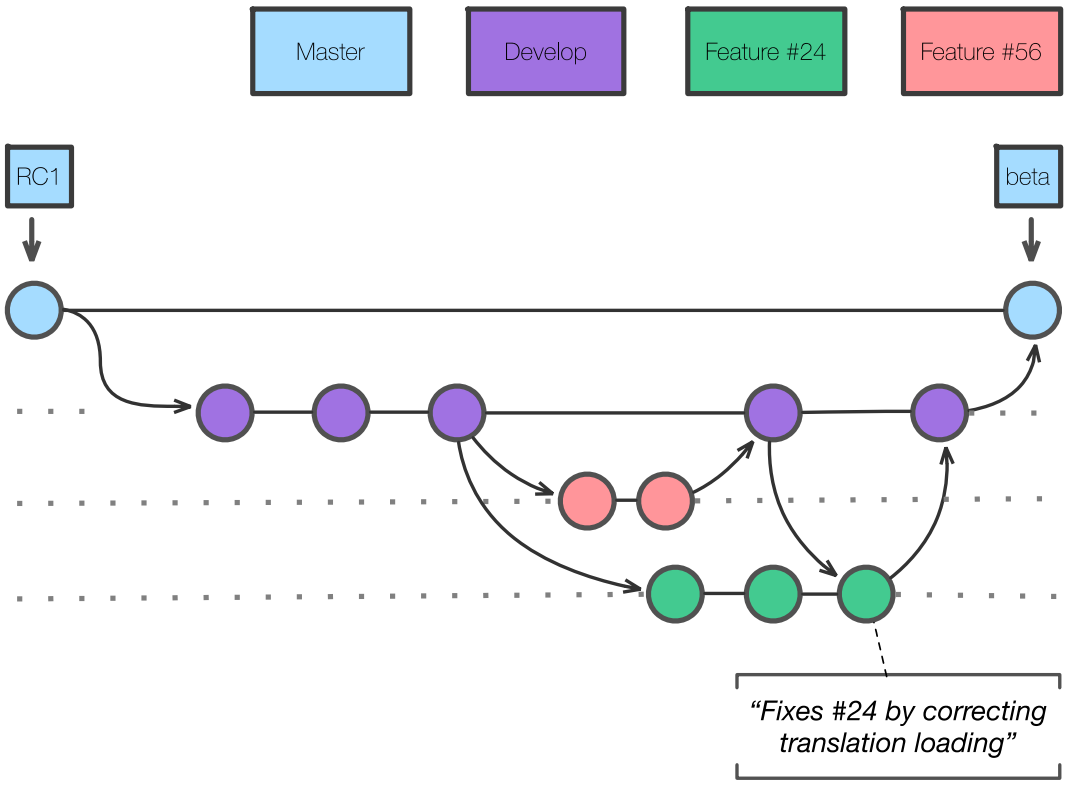

Figure 4 Illustration du Git Flow

Le flux de travail est défini comme suit : un développeur crée une branche de fonctionnalité basée sur le dernier développement, commit son travail, teste son code puis le fusionne dans *develop* s'il s'exécute correctement. Le développeur ne risque pas de faire régresser le développement puisque tout se passe dans sa branche de fonctionnalité aussi longtemps que le développement est dirigé par les tests. Après s'être assuré que sa branche de développement locale est à jour, que tous les conflits sont résolus et que les tests ont bien été exécutés, on peut fusionner dans *develop*.

Pour faciliter le développement de l'application, nous avons choisi de réduire le plus possible les tâches externes au logiciel en les automatisant. En effet, cela réduit les retards notamment dus à l'intégration du code ce qui laisse plus de temps au développement, produit un code plus propre et plus de fonctionnalités.

Nous avons ainsi délégué les tâches répétitives que sont l'intégration, les *builds* et les tests à un serveur qui les exécute dès que du code est envoyé sur le dépôt : c'est l'intégration continue. Cela centralise le contrôle du code, assure une qualité et une stabilité constantes et moins de travail redondant. Les développeurs obtiennent ainsi un retour visuel rapide sur le déroulement des choses et si un problème survient, la version stable du logiciel sur *develop* reste protégée car le contenu ne pourra pas être fusionné.

Cependant, il ne s'agit là que d'un aperçu de la puissance de l'intégration continue car le serveur GitLab fournit par l'école n'implémente pas toutes les fonctionnalités de GitLab-CI, dont la gestion automatique des branches, la livraison et le déploiement de code en continu.

3 Un premier bilan

A l'heure actuelle, notre logiciel présente une interface graphique à la fois minimaliste et intuitive, nous sommes capables de charger une liste de musiques et de produire un fichier audio qui est la concaténation de ces différents morceaux. La Figure 5 montre la fenêtre principale du logiciel, notons que les informations relatives aux pistes ne sont pas extraites automatiquement de ces dernières mais importées par un autre moyen, que nous expliquerons dans la partie 3.2.

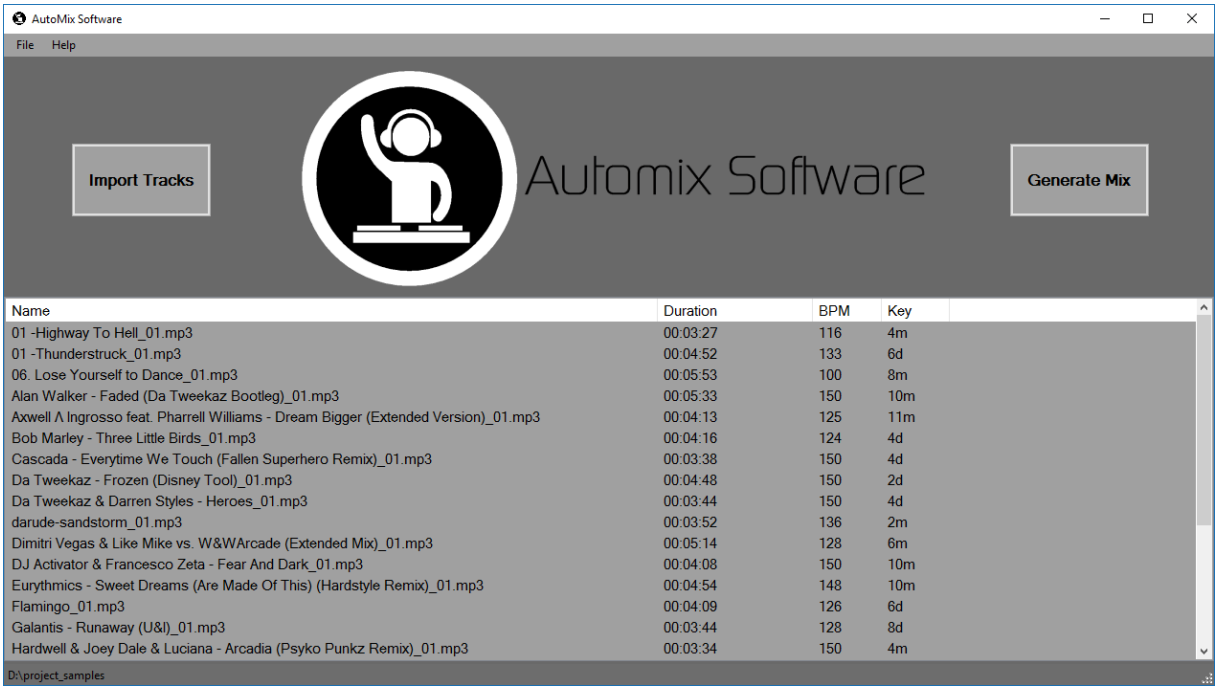

Figure 5 Capture d'écran d'AutoMix Software

Notre projet a quasiment atteint le stade du produit de valeur minimale, nous avons passé beaucoup de temps à surmonter les difficultés exposées ci-après, mais les solutions que nous sommes en train d'apporter devraient nous permettre de rapidement ajouter de nouvelles fonctionnalités.

3.1 Difficultés rencontrées

La prise en main de l'environnement Microsoft .NET (présenté dans la partie 1.2.) n'a pas été évidente, nous n'avions en effet pas les idées très claires sur les différences entre le C++ managé et le C++ natif, et donc par conséquent de l'importation de bibliothèques natives dans notre logiciel qui lui tourne sur la machine virtuelle .NET.

Un des risques que nous avions évalués concernait la prise en main de la bibliothèque audio vitale au bon déroulement du projet. Nous avons malheureusement rencontré d'importantes difficultés sur ce point bien que nous ayons essayé d'intégrer plusieurs bibliothèques audio à notre projet, majoritairement Essentia[2] et Marsyas[3].

[2] Distribuée sous licence GNU Affero General Public License
[3] Distribuée Sous licence GNU General Public License

Dans chaque cas nous avons rencontré des problèmes similaires, comme l'impossibilité totale de recompiler les sources nous même pour pouvoir ajouter des fonctions utiles dans la table des fonctions exportées ou corriger quelques erreurs. Les fonctions les plus intéressantes de ces deux bibliothèques, du point de vue de l'extraction d'informations sur les pistes audio, ont été construites dans des exécutables. Ces programmes compilés sont les seules choses fonctionnelles dont nous disposons.

3.2 Solutions apportées

Pour éviter de rester bloqués par ce problème majeur, nous avons mis en place différents *mocks*[4] simulant une extraction de données audio. Même si cette solution peut encore nous dépanner, nous sommes à la moitié du parcours et nous devons prendre une décision qui soit pérenne pour le logiciel.

Pour cela, nous avions deux options : développer une bibliothèques d'extraction audio, qui respecte les principes du génie logiciel, à partir d'algorithmes existants, ou alors utiliser les exécutables en ligne de commandes distribués par Essentia pour récupérer les informations qui nous intéressent.

Nous avons choisi la deuxième solution, moins chronophage, parce que nous préférons nous concentrer sur l'intelligence du logiciel ainsi que sur les fonctionnalités dont l'utilisateur pourrait profiter plutôt que sur les primitives et traitements liés aux données audio. Cependant, afin d'offrir une expérience utilisateur la plus agréable possible, nous sommes en train d'étudier comment paralléliser et optimiser l'emploi des exécutables et le traitement des informations que ces derniers renvoient. De plus, nous avons également pris la décision de créer une base de données pour conserver en mémoire les données extraites et les charger plus rapidement si l'utilisateur veut créer un nouveau mix incluant des musiques précédemment analysées.

[4] En programmation orientée objet, les mocks sont des objets simulés qui reproduisent le comportement d'objets réels.

Conclusion

En regard du travail accompli, nous avons pour objectif de terminer le produit de valeur minimale durant les deux prochains sprints en appliquant les solutions que nous avons proposé précédemment.

Une fois cela terminé, nous implémenterons de nouvelles fonctionnalités, tout en améliorant l'interface graphique ainsi que la qualité du mix afin d'obtenir le meilleur logiciel possible au terme de ce projet de deuxième année.

Nous allons travailler en gardant la même rigueur du point de vue de l'organisation et du développement, tout en accentuant les efforts sur l'intégration continue et particulièrement la mise en place de tests plus systématique qu'à l'heure actuelle.

Annexes

Annexe1 : Courbe Burn-Up