Mémoire de Master II / Septembre 2018

Diplôme national de master

Domaine - sciences humaines et sociales

Mention – sciences de l'information et des bibliothèques

Parcours – archives numériques

L'archivage du code source sur les plateformes de développement

Maxime Vigne

Sous la direction de Clément Oury
Responsable données, réseaux et standards ISSN – Centre International de l'ISSN

Remerciements

Je remercie mon directeur de mémoire, Monsieur Oury, pour m'avoir orienté vers la bonne direction au début du projet.

Je remercie également tous les collègues que avec qui j'ai travaillé durant mon stage à Libriciel Scop. J'ai énormément appris et profité de leur expérience.

Je souhaiterais également remercier tout particulièrement, Laure Fabre, pour ses conseils précieux et son soutien au quotidien.

Résumé : Le logiciel tient désormais une place importante au quotidien. Son utilisation et son exploitation résulte d'un ensemble de pratiques qui ont permis de développer des outils toujours plus complexes. Afin de favoriser l'évolution ces outils et parce qu'il permet de sauvegarder notre patrimoine, il est important que les communautés de développeurs au sein des plateformes de développement prennent conscience de l'intérêt d'archiver le logiciel.

Descripteurs : Libre, Open Source, Plateforme, Archive, Code source, Logiciel

Abstract : Nowadays, softwares are everywhere, they hold a big place in everyday life. Its utilisation and exploitation is streaming from a bunch of practices which permitted complex tools to emerge. In order for theses new tools to evolve, and because it will permit to maintain and preserve our heritage, a lot is at stake in the realization of those developping communities that the archiving of source codes and softwares is not only necessary for their own sake, but also that it will have great benefits.

Keywords :Free, Open Source, Platform, Records, Source Code, Software

Droits d'auteurs

Sommaire

Sigles et abréviations

AFNOR – Agence Française de Normalisation

API – Application Programming Interface

CD-ROM – Compact Disk Read Only Memory

CLAN – Comprehensive Tex Archive Network

CNRS – Centre National de Recherche Spatiale

CNTRL- Centre National de Ressources Textuelles et Lexicales

CPAN – Comprehensive Perl Archive Network

CRAN – Comprehensive R Archive Network

CVS – Concurrent Version Systems

CXAN – Comprehensive X Archive Network

ENIAC – Electronic Numerical Integrator And Computer

EVIAC – Electronic Discrete Variable Computer

FLOSS – Free Libre Open Source Software

FTP – File Transfer Protocol

GIMP – GNU Image Manipulation Program

GNU – GNU's Not Unix

GNU AGPL/ GPL Affero – GNU Affero General Public Licence

GNU FDL - GNU Free Documentation Licence

GNU GPL – GNU General Public Licence

GNU LGPL- GNU Lesser General Public Licence

IBM – International Business Machines Corporation

INRIA – Institut National de Recherche en Informatique et en Automatique

NASA – National Aeronautics and Space Administration

OAIS – Open Archival Information System

OTRS – Open Source Ticket Request System

PLUME – Promouvoir les Logiciels Utiles Maîtrisés et Economiques dans l'Enseignement Supérieur et la Recherche

SAE – Système d'Archivage Électronique

SIAF – Service Interministériel des Archives de France

SWH – Software Heritage

URL – Uniform Resource Locator

INTRODUCTION

Avant de devenir la machine stable et très polyvalente que nous connaissons aujourd'hui, l'ordinateur a tout d'abord était une machine à calculer.

Le premier prototype de machine à calculer remonte environ au 14e siècle avant JC[1]. Localisé en Chine, le boulier chinois est le premier modèle de système décimal connu ce jour. C'est-à-dire que la représentation d'un chiffre est basé sur une puissance de dix et ses multiples[2]. Le système décimal servira de bases aux préceptes mathématiques et informatiques, dont nous allons chronologiquement citer les évolutions.

En Europe, il aura fallu attendre le début du XVIIe siècle avant que John Napier (1550-1617) ne mette au point les logarithmes (principes de calcul trigonométrique)[3] qui permit vers 1620 d'exploiter la règle à calcul. Le principe consistait ici, grâce aux longueurs de la règle, de faire différents types de calculs, c'est-à-dire additions et soustractions dans un premier temps, puis la multiplication et la division grâce aux logarithmes de Napier. La règle à calcul est ainsi un calculateur analogique, c'est-à-dire, qu'elle applique le principe de mesure des longueurs avec celles du principe des calculs classiques[4]. Il faut savoir, dans notre contexte, que les calculateurs analogiques ont fait partie des premiers systèmes de calculs dédiés aux ordinateurs, avant d'être supplantés par les calculateurs numériques, ces derniers ne se basant que sur le calcul avec des nombres et qui se trouvent être bien plus rapides[5].

---

[1] POISARD C., *Fiche 3 : L'étude du boulier chinois*, Site culturemath.ens.fr, 2006. [en ligne] Disponible sur : http://culturemath.ens.fr/nodeimages/images/fiche3.pdf [consulté le 30/08/2018]

[2] Vikidia, *Système décimal*, Site fr.vikidia.org [en ligne] Disponible sur: https://fr.vikidia.org/wiki/Syst%C3%A8me_d%C3%A9cimal [consulté le 30/08/2018]

[3] MEYER Jacques, *NEPER ou NAPIER JOHN – (1550-1617)*, Site Universalis.fr, 2018. [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/neper-napier/ [consulté le 30/08/2018]

[4] POISARD C., *Fiche 5 : L'étude de la règle de calcul*, Site culturemath.ens.fr, 2006. [en ligne] Disponible sur : http://culturemath.ens.fr/materiaux/poissard/fiche5.pdf [consulté le 30/08/2018]

[5] Wikipédia, *Calculateur analogique,* Site wikipedia.org [en ligne] Disponible sur : https://fr.wikipedia.org/wiki/Calculateur_analogique [consulté le 30/08/2018]

Pour en revenir à l'histoire des machines à calculer, c'est le prototype de William Schickard (1592-1635) élaboré vers 1623[6] et la Pascaline de Blaise Pascal (1623-1662) qui selon le site homocalculus[7], est la première machine à mécaniser une "démarche de l'esprit". Une expression qu'il nous faudra retenir tout au long de ce mémoire et dont le sens s'applique tout à fait aux raisonnements développés dans notre étude. Dans le courant du même siècle, c'est Gottfried Leibniz (1646-1716), qui, s'inspirant des préceptes de la Pascaline, conçoit la multiplicatrice[8], dont les mécanismes et le concept ont été repris jusqu'au milieu du XXe siècle. Leibniz était aussi un des premiers promoteurs du système binaire, qui est aujourd'hui à la base du fonctionnement de nos machines actuelles. Cependant au XVIIe siècle, ce concept n'était encore que théorique.

Ce n'est qu'un peu plus tard que la machine, dont le fonctionnement se rapproche le plus des ordinateurs que nous connaissons aujourd'hui, fait une timide apparition. Initiée par Charles Babbage (1792-1871), il conçoit un prototype dont le principe était basé sur des cartes perforées qui transmettaient les instructions[9]. Ce système perdurera jusqu'aux années 70 pour laisser la place aux éditeurs de textes. Ce projet a notamment été soutenu par Ada Lovelace (1815-1852), ayant collaboré avec Babbage et qui est l'éditrice des touts premiers programmes pour ordinateur[10] et, par la même occasion, la première femme "développeuse".

Le prototype de Babbage fut la dernière machine à calcul à être basée sur un système décimal. En effet, en 1936, Konrad Suze (1910-1995) met un point un ordinateur basé sur la logique binaire[11]. Sans entrer dans les détails, il s'est basé sur les travaux de trois de ces prédécesseurs :

---

[6] MOUYSSINAT Michel, *La machine de Schickard*, Site irem.univ-reunion.fr [en ligne] Disponible sur : http://irem.univ-reunion.fr/homocalculus/Data/menu/visite/visite/theme2/r_machine_shickard.htm [consulté le 31/08/2018]

[7] MOUYSSINAT Michel, *La machine de Pascal*, Site irem.univ-reunion.fr [en ligne] Disponible sur : http://irem.univ-reunion.fr/homocalculus/Data/menu/visite/visite/theme2/r_machine_pascal.htm [consulté le 31/08/2018]

[8] MOUYSSINAT Michel, *Leibniz*, Site irem.univ-reunion.fr [en ligne] Disponible sur : http://irem.univ-reunion.fr/homocalculus/Data/menu/visite/visite/theme2/r_leibnitz.htm [consulté le 31/08/2018]

[9] PIRE Bernard, *BABBAGE Charles (1792-1871)*, Site universalis.fr [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/charles-babbage/ [consulté le 31/08/2018]

[10] Futura, *Augusta Ada Lovelace*, Site futura-sciences.com [en ligne] Disponible sur : https://www.futura-sciences.com/sciences/personnalites/mathematiques-augusta-ada-lovelace-869/ [consulté le 31/08/2018]

[11] FALQUE Jean-Claude, *ZUSE Conrad (1910-1995)*, Site universalis.fr [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/konrad-zuse/ [consulté le 31/08/2018]

- le concept binaire de Leibniz
- le fonctionnement du prototype de Babbage
- et la représentation des nombres dites en virgules flottantes par l'Espagnol Leonardo Torres Quevedo (grand ingénieur et mathématicien du début du XXe siècle)

Il a combiné chacun des trois principes pour en sortir une invention unique et performante, en 1938, le Z1[12].

De l'autre côté de l'Atlantique, aux Etats-Unis, Howard Aiken (1900-1973) conçoit une machine à calcul automatique immense d'une longueur de 15,3 mètres et d'une hauteur de 2,4 mètres[13]. C'est l'entreprise IBM qui accepte de financer son projet, qui verra le jour au milieu du XXe siècle. Les ordinateurs sont alors composés de pièces mécaniques lourdes et imposantes, pas forcément adaptées à un usage au quotidien. En effet, l'Electronic Numerical Integrator And Computer (ENIAC) et sa version évoluée l'Electronic Discrete Variable Computer (EVIAC) occupent une surface de près de 45m², et nécessitent l'intervention de plusieurs techniciens se relayant pour le faire fonctionner[14].

Ce n'est qu'à partir des années 50 que le transistor permettra de réduire drastiquement la taille, la consommation et le coût des ordinateurs, les rendant ainsi plus accessibles. Puis, à partir de cette innovation, les progrès ne font que croître plus rapidement : en l'espace de 20 ans, différents modèles provenant de constructeurs aujourd'hui renommés (IBM, Intel, Apple) commercialisent leurs produits et proposent des machines toujours plus performantes[15]. C'est la loi de Moore qui s'applique ici : Gordon Moore (co-fondateur d'Intel), avait calculé que le nombre de transistors occupant un même espace sur un ordinateur doublerait tous les dix huit mois, accélérant de fait l'obsolescence des machines. Mais cela permet effectivement d'atteindre des performances toujours plus intéressantes en un laps de temps très court,[16] par rapport aux écarts qui pouvaient exister entre les différentes évolutions auparavant.

---

[12] Centraphone, *L'ordinateur,* Site centraphone.fr [en ligne] Disponible sur : http://www.centraphone.fr/ordinateur.htm [consulté le 30/08/2018]

[13] PIRE Bernard, *AIKEN HOWARD HATHAWAY (1900-1973)*, Site universalis.fr [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/howard-hathaway-aiken/ [consulté le 31/08/2018]

[14] Wikipédia, *Electronic Discrete Variable Automatic Computer*, Site wikipedia.org [en ligne] Disponible sur: https://fr.wikipedia.org/wiki/Electronic_Discrete_Variable_Automatic_Computer [consulté le 31/08/2018]

[15] Centraphone, *Op Cit.*

[16] Ibid

Nous profitons donc aujourd'hui des progrès de nos prédécesseurs. Les principes de calcul exploités par nos ordinateurs, toujours au goût du jour aujourd'hui, sont le fruit de travaux et de recherches qui ont duré sur près de 400 ans (si l'on part des logarithmes de John Napier). D'un chercheur à l'autre, le transfert de connaissances a permis de faire progresser la science, et notamment les mathématiques et l'informatique. Les moyens de communication disponibles jusqu'à la moitié du XIXe siècle, voir le début du XXe siècle, ne facilitaient certes pas les échanges entre les personnalités intellectuelles que nous avons citées, et la révolution de l'imprimerie démarrait à peine. Cependant la curiosité et la soif de savoirs des uns et des autres semblait donner suffisamment de motivations pour étudier les travaux de leurs prédécesseurs et contemporains.

Les publications et le développement des moyens de transports aura permis, au fur et à mesure des siècles, de favoriser la diffusion du savoir, réduisant les temps de trajet et facilitant l'accès aux librairies et aux centres d'études. Cet enrichissement est d'autant plus intéressant qu'il est promulgué par une très grande variété de profils, apportant chacun leurs idées tirées de leurs propres expériences et de leur quotidien. Leibniz, par exemple, en plus d'être un éminent mathématicien, était aussi un philosophe, ce qui lui a permis de réfléchir et de poser les premiers concepts du système binaire, tout en imaginant les possibilités que pourraient proposer un tel système. Ces réflexions ont été reprises par Babbage mais ont également permis à Konrad Suze et Howard Aiken de concevoir les premiers ordinateurs contemporains. Malgré les années, voir les siècles pour certains, c'est la même passion qui aura mené chacune de ces personnalités à construire, morceaux par morceaux, tout en puisant dans les ressources du passé, les concepts fonctionnels des ordinateurs que nous exploitons toujours au XXIe siècle.

Le logiciel lui, s'est développé vers le milieu du XXe siècle, d'abord au sein des institutions de recherche (avec des ordinateurs comme l'ENIAC et l'EVIAC). Les performances du logiciel sont étroitement liées aux performances de la machine. L'évolution s'est donc faite parallèlement. A ses débuts, le logiciel était ainsi au centre des études et son exploitation était libre, c'est-à-dire que les développeurs travaillaient régulièrement en collaboration et pouvaient être amenés à échanger en toute liberté, peu importe leur appartenance à des entités privées ou publiques. Le logiciel répond ainsi à des critères de conception et d'exploitation

collaboratives et permet, par le transfert des connaissances et le partage, de concevoir un produit toujours plus performant.

L'avènement d'internet et les possibilités de communications désormais suffisamment développées et accessibles, les sources informationnelles se sont multipliées, les communautés du monde entier peuvent à présent échanger. La confirmation de la loi de Moore laisse entrevoir des possibilités d'évolution presque illimitées. Mais cette vitesse d'évolution bouleverse cependant les usages et les pratiques de consommation, car, comme mentionné plus haut, les machines et les logiciels sont rapidement délaissés pour laisser place aux nouvelles générations. L'écart est par ailleurs de plus en plus étroit d'un modèle à l'autre (l'exemple des smartphones est flagrant, notamment avec Apple et Samsung). La dématérialisation massive des informations est aujourd'hui une problématique au cœur de tous les domaines, l'archivage étant l'un des principaux concernés. Et bien évidemment, le code source et le logiciel ne sont pas épargnés. Eux qui, auparavant, étaient faits de papiers et d'encres [17], ne constituent plus qu'un assemblage de symboles alphanumériques.

De nouvelles pratiques apparaissent mais d'autres se perdent. Le code source et le logiciel sont fragilisés par leur apparence inaltérable alors qu'ils conviendraient de les considérer comme l'héritage qu'il nous reste de nos prédécesseurs. Ils disposent tout de même d'espaces spécifiques facilitant leur gestion et leur distribution : les plateformes de développement. Ces dernières sont désormais des outils indissociables dans le travail des développeurs et se présentent également comme les très rares garantes de la pérennité du code qui est exploité en leur sein. La conservation sur le long terme du code source et du logiciel est de plus en plus évoquée, elle permet notamment d'optimiser le développement de ceux-ci et d'assurer un suivi optimal tout au long du cycle de vie de l'objet.

L'utilisation du logiciel au quotidien pose la question de son archivage, au sens métier du terme. Au vu de sa caractéristique évolutive, il ne peut être sauvegardé qu'au seul titre de patrimoine.

Quelle place tiennent les plateformes de développement dans l'archivage du code source et des logiciels, notamment vis à vis de l'influence de ces derniers sur notre quotidien ?

Dans un premier temps, nous nous appliquerons à définir le contexte de création et d'exploitation du code source, au sein des communautés de développeurs, ainsi que les

[17] PIRE Bernard, *Op Cit.*

notions clés. Nous verrons qu'aujourd'hui, avec les Free Libre Open Source Software (FLOSS), il existe un ensemble d'acteurs travaillant en faveur des libertés sur internet et du libre-arbitre : les activistes du mouvement Libre souhaitent que les utilisateurs puissent avoir le choix d'utiliser les outils qu'ils désirent sur leurs ordinateurs. Ces communautés ont par ailleurs mis en place tout un ensemble de pratiques permettant de s'adapter aux nouveaux espaces virtuels de travail qui ont suivi la délocalisation des échanges et des communications sur des plateformes de développement.

Dans un deuxième temps, nous étudierons plus en profondeur, les plateformes en question. Ce sera ainsi l'occasion de déceler d'une part les avantages, et d'autre part les failles qui sont susceptibles de nous intéresser pour la conservation à long terme du code source sur des espaces dédiés. D'autres cas spécifiques, faisant office de bons, voir très bons élèves, seront également étudiés, et ils nous permettront d'appréhender les premières solutions et infrastructures mises en place pour préserver les savoirs conceptuels véhiculés par le code source et le logiciel.

Nous terminerons par l'identification des périmètres de conservation du code source et nous essaierons de proposer un ensemble de préconisations basées sur l'étude des diverses méthodes de production et de gestion du code source au sein des communautés. Tout ceci devant être adapté à l'ensemble des étapes du cycle de vie d'un logiciel.

1 DEFINITIONS ET CONTEXTE DE PRODUCTION DU LOGICIEL AUJOURD'HUI

1.1 LE CODE SOURCE ET LE LOGICIEL

1.1.1 Le code source

Le code source est selon sa définition, un ensemble d'instructions écrit dans un langage de programmation lisible par un humain et qui, une fois interprété, compilé ou assemblé, devient un code objet qui peut être exécuté par l'ordinateur[18]. Ceci est bien entendu la définition technique. Dans cette réflexion, nous verrons que le code source est également interprété comme un raisonnement intellectuel, car les différents langages de programmation permettent de communiquer avec une machine via une très grande diversité de dialogues. Au-delà donc du code source en tant qu'objet, c'est aussi toute une logique humaine et personnelle qui est perceptible à travers le code. C'est d'ailleurs cette logique, qui, nous allons le voir, n'est pas toujours facilement perceptible, et que les acteurs de l'archivage du code tentent par différents moyens de conserver.

Dans sa complexité, le code source nécessite donc une gestion toute particulière du fait de différents facteurs que nous nous appliquerons à décliner. Cette gestion tend aujourd'hui encore à évoluer au sein des communautés de développeurs, dont celles du Libre et de l'Open Source, qui sont des fervents acteurs de sa conservation étant donné que le code source est leur outil essentiel de travail mais aussi leur mode d'expression.

1.1.2 Le logiciel

Bien évidemment la principale matérialisation du code source est le logiciel. Le Centre National de Ressources Textuelles et Lexicales[19] définit celui-ci comme étant un « *ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitement de données* ». Cette définition du code source est très synthétique et correspond tout à fait à l'utilisation du terme logiciel que nous ferons tout au long de ce mémoire.

[18] PLUME, *Promouvoir les Logiciels Utiles Maîtrisés et Économiques dans l'enseignement supérieur et la recherche*, Site projet-plume.org [en ligne] Disponible sur: https://projet-plume.org/ [consulté le 31/08/2018]

[19] Cnrtl.fr, *Définition, Logiciel* [en ligne] Disponible sur http://www.cnrtl.fr/definition/logiciel [consulté le 30/08/2018]

Tout comme le code source, nous nous attacherons autant à l'objet que peut représenter le logiciel, qu'à tous les facteurs gravitant autour de celui-ci. De sa fonctionnalité principale à son intégration au sein de systèmes complexes, en passant par les documents descriptifs techniques et théoriques, le logiciel est aujourd'hui un élément essentiel, en évolution permanente. Il est employé dans tous les secteurs industriels, et est devenu générateur de revenus gargantuesques pour une très grande majorité d'entreprises. Certaines ce sont d'ailleurs exclusivement basées sur la production et la commercialisation de logiciels. Il est ainsi aisé de citer Microsoft, Apple ou encore Google comme étant des leaders incontestés sur le marché et dont la notoriété des produits n'est plus à démontrer. D'un point de vue métier et bureautique, nous pouvons citer des acteurs comme Adobe (suite de logiciels de traitement de textes, d'images, de vidéos) ou bien le renommé IBM (solutions métiers variés et multiples). Et pour sortir des classiques, nous pouvons également citer des éditeurs de logiciels vidéoludiques comme Blizzard (Warcraft, Starcraft et le nouveau Overwatch), Ubisoft (éditeur français, dont la licence Assassin's Creed a fait l'objet d'un film[20] ou encore Nintendo (acteur nippon, dont les licences comme Pokemon ou Mario sont des incontournables).

1.1.3 Le modèle économique des logiciels propriétaires

En ce qui concerne la commercialisation des solutions, c'est non seulement l'objet (le logiciel) mais aussi une licence individuelle qui est vendue par les acteurs de l'industrie propriétaire. En effet, dans le cas d'un logiciel propriétaire, la législation[21] oblige l'utilisateur à disposer d'une licence pour exploiter le logiciel.

Cette licence se modélise sous différentes formes. Pour reprendre notre exemple cité plus haut, l'achat d'un jeu vidéo physique sur ordinateur nécessite systématiquement une installation sur un poste. Cependant, pour utiliser son bien, l'acheteur se doit de conserver et de toujours disposer du CD-ROM d'installation à chaque utilisation. En effet, le CD-ROM contient une image ISO unique qui ne

[20] Wikipédia, Assassin Creed (film) , Site wikipedia.org [en ligne] Disponible sur : https://fr.wikipedia.org/wiki/Assassin%27s_Creed_(film) [consulté le 31/08/2018]

[21] Agence Pour la Protection des Programmes, *Contrat de licence d'utilisation : logiciel propriétaire*. [en ligne] Disponible sur: **https://www.app.asso.fr/centre-information/base-de-connaissances/code-logiciels/les-contrats/contrat-de-licence-dutilisation-logiciel-proprietaire** [consulté le 31/08/2018]

s'installe pas sur l'ordinateur, mais qui est obligatoire pour pouvoir le lancer. Sans le CD-ROM et cette image ISO, il est impossible d'accéder à l'information.

Cette licence peut néanmoins très bien être totalement dématérialisée. C'est-à-dire qu'un logiciel peut être téléchargeable sur internet, mais une clé unique générée par l'éditeur est nécessaire à son installation et à son utilisation illimitée. Un seul code source et un seul logiciel sont ainsi décuplés en fonction de la demande et vendus unitairement aux utilisateurs.

Une restriction supplémentaire s'applique aux logiciels propriétaires : il n'est pas autorisé d'exploiter le code source pour y apporter la moindre modification ou évolution, tout comme il est formellement interdit d'en faire la commercialisation à des fins personnelles. Toujours dans le domaine du jeu vidéo, la vente d'un jeu ou d'une console entre utilisateurs est tolérée, mais génère encore des tensions entre les éditeurs et les consommateurs[22].

Si le modèle économique du logiciel propriétaire a pendant longtemps concerné la majorité des entreprises, elle est dorénavant contestée par les communautés du logiciel Libre et Open Source.

1.2 LES MOUVEMENTS LIBRE ET OPEN-SOURCE

1.2.1 Notion de liberté fondamentale

Le modèle économique des logiciels libres est basé sur une philosophie autour des communautés d'utilisateurs de l'internet et prônent les valeurs de « liberté fondamentale »[23] sur internet. En effet, selon les membres de la communauté, l'exploitation du code source ainsi que l'utilisation du logiciel devrait être entièrement libre et gratuite. Les sources de revenus des développeurs des communautés du Libre et Open Source sont principalement basées sur le service de maintenance du logiciel mais aussi de prestations techniques auprès des utilisateurs. L'étude de ces évolutions permet notamment de comprendre la naissance d'un besoin, besoin correspondant à des outils spécifiques pour la production collaborative de code source qui, en d'autres termes, se traduit aujourd'hui par les plateformes de développement.

[22] "Bethesda, *Interdir la revente d'un jeu d'occasion et menace de poursuites*", Site journaldugeek.com [en ligne]. Disponible sur **https://www.journaldugeek.com/2018/08/13/bethesda-interdit-revente-dun-jeu-doccasion-menace-de-poursuites/** [consulté le 31/08/2018]

[23] Free Software Foundation, *Philosophie du projet GNU*, Site gnu.org [en ligne] Disponible sur: http://www.gnu.org/philosophy/philosophy.html [consulté le 31/08/2018]

Nous définirons ainsi des notions techniques et informatiques qui font partie intégrante des problématiques de gestion et de conservation du code source. L'idée ici est de bien saisir ce qu'implique un potentiel archivage du code source logiciel, c'est-à-dire que tout comme l'archivage électronique pour les documents à valeur probante, il faut identifier la totalité du cycle de vie d'une donnée pour définir les paramètres optimaux d'un système d'archivage spécifique au code source d'un logiciel.

Néanmoins, avant cela, un bref historique des mouvements Libre et Open Source s'impose, pour comprendre la philosophie de ces deux mouvements ainsi que les différences existantes. En effet, la vulgarisation des termes "libre" et "open source" peut avoir tendance à créer une certaine uniformisation autour de ces deux mouvements, qui du reste ne défendent pas exactement les mêmes valeurs, bien qu'elles soient proches.

1.2.2 Au départ était le libre...

Le mouvement Libre a été initié au tout début par Richard Matthew Stallman (né le 16 mars 1953)[24], dans le début des années 80. Développeur dans une entreprise californienne, il appréciait tout particulièrement l'état d'esprit qui y régnait. Les échanges entre les développeurs étaient très fréquents et chacun partageait ses astuces, ses solutions pour résoudre une diversité de problèmes. Plus important encore, la résolution de bugs et de correction de bugs se faisaient également de manière collaborative. Chacun se penchait sur le problème d'un ou plusieurs utilisateurs et une solution commune émergeait, permettant une résolution efficace des différents points de blocage, ainsi qu'une correction plus rapide du code. L'apport de chacun permettait en outre, selon Richard Stallman, d'optimiser au mieux le code source d'un logiciel.

L'entreprise pour laquelle travaillait Richard Stallman, fut rachetée au profit d'une société beaucoup plus rigide dans la réutilisation du code et dans le partage d'informations. Des accords de confidentialité devaient être signés par les employés, empêchant ainsi légalement les employés de diffuser des informations dites "confidentielles" à un tiers extérieur. C'est à ce moment que Richard Stallman décide de partir et qu'il lance le mouvement Libre, pour continuer à travailler selon les modalités qu'il avait connues. Il développa ainsi toute une

[24] *Ibid.*

philosophie construite autour de la liberté d'utilisation[25] et l'accès pour tous à Internet et aux informations qui y circulent. Le code source des logiciels en fait bien évidemment partie. Son projet se nomme GNU (se prononce "gnou" comme l'animal), pour GNU's Not Unix (GNU n'est pas Unix)[26], Unix étant l'un des tout premiers systèmes d'exploitation. En ce sens, GNU avait pour vocation de devenir le système d'exploitation de référence pour le mouvement Libre, ce que Linux a réussi à accomplir au début des années 90, nous en reparlerons plus loin.

La réflexion de Stallman sur les logiciels et le code source se synthétise ainsi : une production de l'esprit ne peut être commercialisée de la façon dont procèdent les entreprises productrices de logiciels propriétaires. Le code source d'un logiciel (et le logiciel qu'il régit par définition) ne peuvent être vendus dans une forme fixe et à l'unité [27].

Richard Stallman est donc en totale opposition avec le modèle économique précédemment décrit (1.1.3) et se place en tant que fervent défenseur des libertés des utilisateurs, depuis maintenant de nombreuses années. Ses convictions se sont par ailleurs retrouvées au sein d'autres communautés de développeurs qui sont venues gonfler le mouvement. L'association GNU a notamment permis depuis la mise en ligne d'un certain nombre de licences régissant l'utilisation des logiciels Libre et de leurs codes sources[28].

1.2.3 Réglementations en vigueur dans le Libre

Il existe aujourd'hui un très grand nombre de licences créés par la GNU et approfondies par la communauté. Chacune permet plus ou moins de souplesse dans l'utilisation du code source, en fonction également du type de bien qui est produit. Les licences sont rédigées de façon à ce que les citations du texte soient en opposition presque totale avec ce que nous pouvons retrouver dans l'industrie du logiciel propriétaire.

1.2.3.1 La GNU General Public Licence (GNU GPL)

La licence publique générale GNU est souvent désignée par l'abréviation « GNU GPL ». Elle sert de base à toutes les autres et de socle à cette législation globale autour de l'utilisation du logiciel Libre. C'est la licence de la plupart des programmes GNU et

[25] Ibid.
[26] Ibid.
[27] Ibid.
[28] Free Software Foundation, *Licences*, Site gnu.org [en ligne] Disponible sur: http://www.gnu.org/licenses/licenses.html [consulté le 31/08/2018]

de plus de la moitié des logiciels libres actuellement distribués. La version 3 est la plus récente.

Elle identifie notamment les quatre grandes notions fondamentales du Libre :

- la liberté d'utiliser le logiciel à n'importe quelle fin,
- la liberté de modifier le programme pour répondre à ses besoins,
- la liberté de redistribuer des copies à ses amis et voisins,
- la liberté de partager avec d'autres les modifications qui ont été faites.

Si ces quatre notions sont respectées, le logiciel est donc considéré comme libre.

1.2.3.2 La GNU Lesser General Public License (GNU LGPL)

La licence publique générale amoindrie est adoptée par certaines bibliothèques de composants. La version 3 est la plus récente. Elle régit ainsi l'utilisation des composants additionnels ou complémentaires à imbriquer à des solutions complètes.

1.2.3.3 La GNU Affero General Public License (GNU AGPL ou GPL Affero)

La licence publique générale GNU Affero est basée sur la GNU GPL, mais ajoute une clause additionnelle qui autorise les utilisateurs interagissant avec le logiciel sous licence via un réseau à recevoir les sources de ce programme. La GNU recommande l'utilisation de la GNU AGPL pour tout logiciel exécuté sur un réseau. La version 3 est la plus récente.

Par exemple, les logiciels de gestion de tickets dans le cadre d'un service de support pour un ou plusieurs produits peuvent être sous licence GNU AGPL (le logiciel OTRS en fait partie). Un logiciel proposant ainsi d'échanger des informations sur un serveur doit également permettre à d'autres utilisateurs d'accéder et d'exploiter le code source. Chose que ne proposait pas la Licence générale. En effet, un utilisateur qui récupère le code source d'un logiciel exécuté sur un réseau ne peut pas y apporter de modifications sans être obligé de divulguer à son tour son code source et la documentation y faisant référence.

1.2.3.4 La GNU Free Documentation License (GNU FDL)

La licence GNU de documentation libre est une forme de *copyleft*[29] destinée aux manuels, aux livres scolaires et autres documents. Son objectif est de garantir à tous la possibilité de copier et de redistribuer librement le document avec ou sans modifications, dans un but commercial ou non. La version 1.3 est la plus récente. A titre d'exemple, certains éléments de la bibliographie de ce mémoire sont sous licence GNU FDL.

1.2.4 Cohérence et crédibilité

Cette structuration juridique de « degrés » de licence permet ainsi d'opposer un système de fonctionnement viable à celui des licences propriétaires. Lors d'une lecture plus précise des licences, le terme récurrent de *copyleft* permet justement de signifier qu'il est possible de copier un logiciel et que les versions modifiées de ce dernier doivent être sous licence libre. Cela permet notamment d'assurer et de perpétuer le partage de connaissances et respecter les libertés fondamentales que prône le mouvement Libre.

Le fait qu'il existe une licence pour la documentation incite également les utilisateurs à modifier, voir publier leurs propres licences. Ces dernières doivent respecter les quatre notions fondamentales (cf GNU GPL) et doivent être soumises à la Free Software Foundation qui est le support principal de la GNU. Cela permet donc d'adapter une licence aux besoins spécifiques de chaque utilisateur tout en assurant une harmonisation législative, les licences précédemment citées servant de piliers.

1.2.5 Le mouvement Open Source

Au sein du mouvement Libre est né vers la fin des années 90[30] un mouvement très similaire, aux finalités semblables mais dont la dénomination n'évoque pas la même philosophie proposée par Richard Stallman. L'Open Source porte ainsi des valeurs d'ouverture et de diffusion du code source mais soutient également que cette ouverture, encourageant la collaboration entre développeurs, permet d'obtenir des logiciels toujours plus puissants et fiables, susceptibles en premier lieu d'intéresser des utilisateurs plus larges, et qui seraient surtout plus à même de se faire une place sur le marché économique. Pour éviter toute confusion et permettre d'identifier comme il se doit les

[29] *A retrouver dans le Glossaire* - En opposition avec le Copyright, « gauche d'auteur » désigne, pour une œuvre ou création soumise à la propriété intellectuelle, une licence impliquant la conservation du droit d'auteur, inaliénable, tout en permettant une réutilisation, modification et diffusion de l'objet soumis à celui-ci.

[30] STALLMAN Richard M., *En quoi l'open source perd de vue l'éthique du logiciel libre ?*, 2016. [en ligne] Disponible sur : https://www.gnu.org/philosophy/open-source-misses-the-point.fr.html [consulté le 30/08/2018]

deux mouvements, le terme de Free Libre Open Source Software (FLOSS) permet d'englober l'ensemble des logiciels faisant partie du périmètre de production des mouvements Libre et Open Source.

1.2.6 Une implantation difficile dans les mentalités

L'adhésion à la philosophie du mouvement Libre et Open Source aura mis quelques années pour se diffuser, et aujourd'hui la présence de logiciels dits "libres", ou "open source" sur internet et au sein d'une grande diversité de structures, témoigne de la popularité du mouvement[31]. Il n'en a pas toujours été ainsi, notamment à cause du monopole évident de quelques fabricants d'ordinateurs et de systèmes d'exploitations propriétaires. Microsoft pour ne citer que lui, dispose aujourd'hui d'une large palette d'outils proposés via son système d'exploitation Windows.

Dans leurs slogans, les entreprises propriétaires n'hésitent pas à mettre en avant, la simplicité, la sécurité et la fiabilité de leurs produits, en assurant qu'un suivi et des mises à jour récurrentes viennent garantir ces trois aspects. Ce qui n'est pas faux : les développeurs, quelles que soient les fonctionnalités et l'utilité de ces logiciels, ont trouvé en Microsoft ou Apple un socle solide sur lequel écrire du code source. Le temps passant, l'informatique se démocratise et les entreprises évoluent, d'autres apparaissent. La grande majorité d'entre elles continuent à développer des logiciels sous licence propriétaire, en adéquation avec les systèmes d'exploitation les plus populaires pour toucher un plus large public : le marché se structure et le modèle économique de ces sociétés s'impose.

Prenons de nouveau pour exemple le marché du jeu vidéo sur ordinateur: il n'existe à ce jour que très peu de moyens pour profiter d'un jeu vidéo sur un système d'exploitation labellisé FLOSS. Le marché s'est structuré autour de systèmes d'exploitation propriétaires car ils sont leaders dans la distribution de leurs produits, notamment Microsoft. Il paraît logique que les acteurs du monde vidéoludique préfèrent davantage construire leurs produits sur des systèmes exploités par le plus grand nombre d'individus, afin de toucher le plus large public, d'assurer une plus grande souplesse de vente et une rentrée stable des bénéfices.

[31] calimaq, *Les logiciels produits par les administrations sont passés en Open Source par défaut (et voici pourquoi),* Site scinfolex.com, S.I.Lex, 2018. [en ligne] Disponible sur : https://scinfolex.com/2017/12/08/les-logiciels-produits-par-les-administrations-sont-passes-en-open-source-par-defaut-et-voici-pourquoi/amp/?__twitter_impression=true [consulté le 30/08/2018]

Par exemple, le titre phare de l'éditeur Blizzard, World Of Warcraft, n'est disponible que dans une version Windows et une version Mac OSX. Nous parlons ici d'un jeu vidéo qui existe depuis plus de 10 ans et dont le nombre d'abonnés se chiffre à plusieurs millions[32]. Autre exemple : l'éditeur Riot Games, créateur de League Of Legends (jeu vidéo le plus joué au monde[33]) a rapidement développé une version Mac OSX suite au succès de son jeu mais ne semble à ce jour prévoir aucune version pour un système d'exploitation Libre ou Open Source. Et ce, malgré le succès unanime de la licence auprès des joueurs. Il n'est pas intéressant pour ces éditeurs de se tourner vers les FLOSS actuellement car ils savent pertinemment que la population de joueur se trouve aujourd'hui sur des systèmes d'exploitation propriétaires, et c'est le cas pour de nombreux autres éditeurs de jeux vidéos à succès. La situation ne stagne pas cependant, loin de là. Des projets continuent de voir le jour sur les forums, les communautés du Libre et de l'Open Source s'activent et trouvent des solutions de plus en plus efficaces pour permettre aux joueurs d'avoir le choix, et dans le jeu vidéo qu'ils souhaiteraient jouer, et dans le système d'exploitation qu'ils souhaiteraient utiliser pour cela.

En bref, les logiciels libres ont peiné à trouver leur public, face à des éditeurs mieux armés, dont les projets trouvaient plus facilement des sources de financement.

Cependant, même sans financements, les membres des communautés Libre et Open Source travaillent, eux, bénévolement sur une grande variété de projets. Etant donné le caractère "gratuit" de ce travail, les projets qui sont développés ont plus tendance à correspondre à un besoin métier, ou du moins utile quotidiennement. Le logiciel développé doit impérativement apporter une solution à un problème majeur ou mineur qui nécessite l'intervention des développeurs[34]. Toujours dans notre exemple du jeu vidéo : les nouvelles tendances de cette industrie (compétition e-sportive, démonstration en streaming, banalisation du multijoueur en réseau, etc.), sont l'une des raisons pour lesquelles les communautés du Libre et de l'Open Source se penchent aujourd'hui sur la problématique du jeu vidéo. La popularité du domaine détermine en quelques sortes les tendances de production des FLOSS. Auparavant, il était surtout essentiel de proposer des outils de bureaux et de travail pour les utilisateurs et de disposer du même confort que sur un système d'exploitation propriétaire.

[32] Nombre d'abonnés de World of Warcraft, Site millenium.org [en ligne] Disponible sur : https://www.millenium.org/news/176155.html [consulté le 31/08/2018]

[33] L'express, *League of Legends : le jeu multijoueur le plus joué au monde* [en ligne] Disponible sur : https://www.lexpress.fr/culture/jeux-video/league-of-legends-le-jeu-multijoueur-le-plus-joue-au-monde_1721040.html [consulté le 31/08/2018]

[34] BAR Moshe, FOGEL Karl, *Open Source Development with CVS*, 3rd Edition. Arizona : Paraglyph Press, 2003. 368 p.

Aujourd'hui la fiabilité et la puissance de beaucoup de logiciels libres n'est plus à prouver. Des systèmes d'exploitation comme Ubuntu, très populaire, ou encore les suites Libre Office ou Open Office sont des alternatives tout à fait intéressantes par rapport à ce qui se fait côté Microsoft (Suite Microsoft Office). L'argument avancé par les partisans de l'Open Source se justifie en effet, du fait des ressources infinies proposées mises à dispositions par les communautés : les développeurs et les utilisateurs.

Nous nous intéresserons donc au fonctionnement de ces communautés, ainsi qu'aux méthodes mises en place dans la gestion du code source. L'analyse de ces méthodes permettra notamment dans la suite de ce mémoire de mettre en avant le potentiel des pratiques de travail collaboratives pour l'archivage du code source et du logiciel.

1.2.7 Le fonctionnement des communautés Libre et Open Source

Les développeurs de logiciel libre fonctionnent ainsi grâce à la participation bénévole de chacun des membres d'un projet, en fonction de l'intérêt de l'individu pour celui-ci et de ce qu'il a à gagner dans son aboutissement. Pour exemple, nous pouvons partir du titanesque projet de Linus Torvalds : Linux.

1.2.7.1 Linux

Ce dernier a été développé via la collaboration massive d'utilisateurs soucieux d'exploiter un système non propriétaire, fiable et sécurisé à partir d'un noyau solide. Il serait complexe d'expliquer ici en quoi le travail de Linus Torvalds est exceptionnel. Cependant il est néanmoins intéressant d'étudier le travail des collaborateurs sur un projet d'une telle ampleur.

Le système Linux est mondialement connu et utilisé par des millions d'utilisateurs, et a permis de développer un très grand nombre d'autres systèmes. Même si au départ son utilisation restait réservée à des utilisateurs aguerris, il s'est peu à peu ouvert à un plus large public, notamment grâce à un important travail provenant de différents utilisateurs dans le monde.

1.2.7.2 Le Bazar

La communauté du logiciel libre est une communauté soudée, basée sur l'entraide et sur les compétences presque inépuisables d'utilisateurs, issus de tous horizons et de tous profils, et ce à travers le monde grâce à internet. L'un des initiateurs du mouvement Open Source, Eric Raymond, parle, dans son essai *Cathedral and The Bazaar*[35], d'une construction à la manière d'un bazar, en opposition à celle de la cathédrale. C'est ainsi qu'il décrit la méthode collaborative mise en place pour le développement de Linux : un noyau créé par Linus Torvalds, disposant d'une base solide et fiable, sur laquelle d'autres développeurs peuvent venir s'appuyer pour développer: soit, le noyau lui même, soit développer d'autres systèmes et modules à partir de ce même noyau.

Au fil des discussions et des échanges, ce sont donc des sortes "d'échoppes" (comme les échoppes d'un bazar) qui se sont constituées autour de Linux, proposant ainsi un produit différent et évolué, adapté de la version de Linus Torvalds. Ces différentes échoppes représentent ainsi des points de fixations stables reliés, d'une part, au Linux originel et d'autre part, aux créations originales dont Linux a permis le développement.

D'embranchements en embranchements, il faut donc imaginer Linux comme le point central d'un bazar qui permet encore aujourd'hui à celui-ci de s'étendre à travers le monde. Ce modèle de création est, comme précisé plus haut, opposé à celui de la cathédrale: cette dernière répond en effet à une pratique de construction très rigoureuse, ne laissant que très de peu de flexibilité au moment des travaux. Dans la cathédrale, chaque « ouvrier » doit respecter les plans de développement pour aboutir au résultat escompté. L'ouvrage sera certes fabuleux et pérenne mais les possibilités d'exploitation et d'évolutions sont néanmoins beaucoup plus restreintes que via la méthode du bazar.

1.2.7.3 Un modèle récurrent

Le modèle de production de Linux reste une référence pour les communautés d'internet, ainsi que la preuve que les mouvements Libre et Open Source peuvent parvenir à créer un système complexe et fiable, avec pour seule ressource les membres de leur communauté. D'une certaine façon Linux, et tout ce qui gravite autour, est l'aboutissement idéal de la philosophie pour laquelle Richard Stallman continue de prêcher partout sur la surface du globe. D'autres systèmes d'exploitation, comme le

[35] RAYMOND Eric S., *The Cathedral and the Bazar*. États-Unis : édition « Libre », 1997. 241p. [en ligne] Disponible sur : http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ [consulté le 30/08/2018]

projet Debian[36] par exemple, se sont basés sur cette méthode favorisée par l'extension d'internet et de l'amélioration des technologies de communication et de partage.

1.2.8 Les membres des communautés

Cependant tout ceci n'aurait pas été possible sans la participation de très nombreux développeurs et utilisateurs qui contribuent ensemble à produire des logiciels. Ce qui est d'autant plus impressionnant est que Linux a été produit à une époque où internet et les technologies informatiques ne facilitaient pas toujours le travail collaboratif entre les différents membres de la communauté. Bien sûr, certaines règles doivent être respectées, des leaders doivent pouvoir orienter les uns et les autres vers la bonne marche à suivre, etc.

1.2.8.1 Le maintener

Cela nous permet ainsi d'introduire la notion de *maintener* (littéralement, "agent de maintenance" en français). Elle est directement lié à la hiérarchie qui peut exister sur le développement d'un projet. Ici, il n'est pas question pour le *maintener* de régner en dictateur sur la communauté et sur les membres de son projet. Cela n'attirerait d'ailleurs que très peu de disciples à travailler sur celui-ci. Le *maintener* agit au contraire comme un leader et un coordonnateur au sein de son équipe[37]. Cette fonction est d'ailleurs souvent affiliée à plusieurs personnes. Elle va permettre, entre autres, de gérer les habilitations, c'est-à-dire gérer les autorisations de modification du code source.

Le *maintener* est aussi en charge du bon état de santé du code source et doit donc scrupuleusement étudier toutes les propositions d'évolution qui sont soumises à son équipe et qui nécessitent la moindre modification du code. De par ses différentes fonctions, il répartit le travail équitablement entre les membres (*maintener* lui-même inclus) en fonction des compétences de chacun sur ce projet.

Cependant, il reste difficile d'organiser une gestion du travail en parallèle. La centralisation des échanges vers un espace virtuel n'a pas que des avantages : les décalages horaires varient en fonction des positions géographiques des uns et des

[36] Software in the Public Interest et autres, *debian, The universal operating system*, Site debian.org, 2018. [en ligne] Disponible sur: https://www.debian.org/index.fr.html [consulté le 31/08/2018]

[37] BAR Moshe, FOGEL Karl, *Open Source Development with CVS*, 3rd Edition. Arizona : Paraglyph Press, 2003. 368 p.

autres à un moment donné. Chacun gère également soi-même son temps de travail. C'est pour cela que le développement d'un logiciel s'effectue sous forme "blocs". Nous reparlerons de la répartition des tâches et du développement orienté objet plus loin.

1.2.8.2 Les utilisateurs

Les utilisateurs sont, selon Eric Raymond[38], des atouts essentiels au bon développement du projet. Avant toute chose, ils sont la raison principale pour laquelle le logiciel est développé. Ils sont les vecteurs indirects des créations de projets, de par leurs besoins exprimés au quotidien. La diversité des profils utilisateurs permet d'avoir une grande variété de point de vue et de sortir du cadre technique d'un développeur pour réellement répondre à un besoin utilisateur. Et puis, un développeur est aussi, avant tout, un utilisateur.

As@lae, par exemple, est un Système d'Archivage Electronique (SAE) produit par l'éditeur de logiciels Libre, Libriciel SCOP. L'entreprise a étroitement travaillé, via un groupe de travail, avec des archivistes des collectivités, des informaticiens, et le service interministériel des archives de France (SIAF). Cette solution de SAE a été conçue en fonction des besoins métiers (présence des archivistes) et des contraintes technologiques (présence des informaticiens). Les normes d'archivage NF Z 42-013 et ISO 14721 (Open Archival Information System, OAIS) ont servi de socle pour déterminer les modalités de gestion de l'information au sein d'as@lae, et également de définir ses pré-requis d'exploitation. Parmi d'autres success story, nous pouvons également citer les suites Libre et Open office (pour les raisons citées plus haut), ou bien l'application Wine permettant la compatibilité de la plupart des grosses productions vidéoludiques sur un système d'exploitation libre. GIMP (GNU Image Manipulation Program) et Blender sont des équivalents respectifs de Adobe Photoshop et de Autodesk Maya. Ces différentes réalisations répondent à des besoins utilisateurs ou bien proposent une alternative fiable et surtout performante, aux solutions logicielles propriétaires. Pour citer un autre exemple concret de besoins métiers, il y a aussi les plateformes de développement et les logiciels de versionning. Nous en reparlerons à la fin de cette première partie.

Les utilisateurs participent ainsi, indirectement mais étroitement, à la réalisation d'un logiciel, parce qu'ils sont les premiers à formuler un besoin. Mais ils ne maîtrisent pas toujours les ressources qui permettent de développer un logiciel, d'où la complémentarité avec les développeurs. Cette notion de compréhension des concepts

[38] RAYMOND Eric S., *Op Cit.*

d'un logiciel nous sera essentielle pour le développement de la deuxième partie du mémoire.

1.2.8.3 Les développeurs

Le développeur est donc une sorte de traducteur/médiateur entre la machine et l'utilisateur. Il maîtrise les différents langages de programmation et sait dialoguer avec une machine. Il transcrit ses instructions via le code source, pour obtenir de l'ordinateur l'accomplissement d'une ou plusieurs tâches. Sur les exemples que nous avons cités jusqu'à maintenant, le développeur, qu'il soit instigateur du projet ou bien exécutant, ne travaille presque jamais seul (le SAE as@lae étant l'exception parmi ces exemples).

C'est aussi l'argument non négligeable des FLOSS : si le développeur est l'instigateur d'un projet, et bien qu'il souhaite répondre à un besoin utilisateur, il se doit d'abord d'investiguer sur de potentiels projets ou réalisations qui auraient déjà pu être diffusés et qui sont susceptibles d'aller dans la même orientation que l'idée première[39]. Si c'est le cas, il sera bien plus intéressant de proposer sa participation au projet et d'enrichir de ses compétences l'équipe qui aura potentiellement été formée pour le développement du produit. Si le code source d'un logiciel profite à tous, de quelque manière que ce soit, la philosophie du logiciel libre permet également à un ou plusieurs membres d'un projet de profiter des compétences extérieures et dont la source serait presque inépuisable.

Nous allons à présent nous intéresser aux méthodes d'écriture collaborative, qui, avec le temps, ont dû s'adapter au nombre croissant de collaborateurs amenés par l'expansion d'internet.

1.3 LES MÉTHODES DE TRAVAIL COLLABORATIVES

En effet, comme précédemment évoqué, les communautés du Libre et Open Source mettent à disposition le code source des logiciels produits par leurs membres pour permettre à chacun de venir les éditer à sa guise. Cela sert aussi à venir en aide aux créateurs, notamment par la résolution de bugs[40], l'écriture de codes spécifiques ou même pour diffuser le code source sur un site web ou une plateforme.

[39] RAYMOND Eric S., *Op Cit.*
[40] Ibid.

1.3.1 Définition d'une communauté

« *A virtual community exists when it is possible for a group of individuals to meet and interact with each other in cyberspace and these individuals voluntarily choose to participate in these meetings and interactions* »[41].

C'est ainsi que la communauté était définie par Steinmüller. En français cela signifie qu'une communauté n'existe que lorsqu'il est possible pour un groupe d'individus de se rencontrer et d'interagir entre eux virtuellement, volontairement. En effet, la philosophie des FLOSS supposent que les développeurs discutent de leurs travaux et échangent des informations permettant à chacun d'acquérir de nouvelles compétences et de contribuer à favoriser les libertés sur internet. A cette définition, nous pourrions ajouter également qu'une communauté se caractérise aussi par le partage d'une pratique et d'une activité commune. En ce qui nous concerne ici, il s'agit de code source et de logiciels.

Avant les années 90, les développeurs ne travaillaient que très peu à distance, les technologies ne permettant pas de travailler de façon optimale avec des individus situés dans des lieux différents. Mais avec la démocratisation d'internet, les pratiques sédentaires se sont mises en place sur des espaces virtuels de travail, où s'échangent des informations et des données : nous retrouvons ici l'image du bazar d'Eric Raymond[42]. C'est ainsi qu'il a été possible pour les développeurs de travailler avec de plus larges profils d'individus, indépendamment de la situation géographique de l'un ou de l'autre. De plus, la langue employée dans les langages informatiques dans la majorité des cas est l'anglais, réduisant ainsi les problèmes rencontrés à cause de la barrière de la langue.

La compréhension du code source d'un développeur n'est cependant pas toujours évidente à comprendre malgré cela car il existe, au sein des différents langages de programmation, différentes manières d'arriver à l'objectif escompté. Nous y reviendrons plus précisément en deuxième partie.

Dans tous les cas, quelle que soit la distance, les développeurs ont mis au point quelques règles et méthodes pour gérer et développer le code source qui est produit. Celles que nous allons décliner concernent principalement les pratiques qui permettent entre autres d'assurer un suivi et une conservation fiable du code source. Ces méthodes varient en fonction de plusieurs paramètres : le nombre d'individus participant au projet,

[41] CORIS Marie, LUNG Yannick, « Les communautés virtuelles : la coordination sans proximité ? Les fondements de la coopération au sein des communautés du logiciel libre », *Revue d'Économie Régionale & Urbaine*, 2005/3 (juillet), p. 397-420. DOI : 10.3917/reru.053.0397. [en ligne] Disponible sur https://www.cairn.info/revue-d-economie-regionale-et-urbaine-2005-3-page-397.htm [consulté le 30/08/2018]

[42] RAYMOND Eric S., *Op Cit.*

les compétences dans un ou plusieurs langages de programmation, l'expérience de chacun des membres du projet, etc. Différentes questions sont alors soulevées : comment sont gérées les habilitations mises en place au sein des communautés ? Comment garantir un développement du projet harmonieux ? Comment faciliter la compréhension du code et l'enrichir ? Autant de questions auxquels les équipes ont su répondre au fil des années.

Les pratiques que nous avons sélectionnées tiennent compte de la suite du développement du mémoire pour être mis en relation avec l'archivage du code source et du logiciel. Il est néanmoins important de saisir leur fonctionnement théorique et technique, car il permet de soutenir un certain nombre de solutions pour l'archivage, dont nous rediscuterons en troisième partie.

1.3.2 La notion de projet

Débutons par un petit point de précision sur ce qu'est un projet chez les développeurs. Bien évidemment, cela concerne la production d'un code source régissant un logiciel. Mais c'est aussi réfléchir en amont, et une fois la mise en production de ce dernier effectué, aux possibilités d'évolution et de résolution de problèmes que les utilisateurs auraient rencontrés dans leur quotidien. En d'autres termes: la maintenance.

En effet, le projet ne définit pas simplement l'étape de réflexion et de création du logiciel mais il est développé en permanence pour, d'une part, assurer le suivi des bugs rencontrés, et d'autre part, le rendre toujours plus performant. Cette recherche de performance est d'ailleurs renforcée par l'accessibilité totale du code source à tous les individus. Le projet, c'est une dimension qui va au-delà de la production exclusive d'un code source.

C'est aussi, et sur un troisième point, créer de la documentation, entretenir le site web et/ou le serveur sur lequel le code est conservé pour assurer une diffusion optimale aux utilisateurs, communiquer avec la communauté sur les sorties des nouvelles versions, sur la planification et la tenue de potentiels groupes de travails, sur le début ou la fin d'un projet …

En résumé, un projet, c'est avant tout être capable de tenir plusieurs rôles à la fois pour les développeurs, et d'être capable de s'organiser et de pratiquer de la gestion de projet dans tous ses aspects.

1.3.3 La programmation orientée objet

Il serait complexe et inapproprié de discuter ici des spécificités techniques affiliées à ce qu'on appelle dans le jargon informatique le "développement orienté objet"[43]. En bref, ce concept permet de développer une application via des objets indépendants (du code source fonctionnel) capable de communiquer avec d'autres objets et donc de s'adapter au mieux aux différentes variations pouvant survenir durant la phase de production d'un logiciel. Chaque utilisateur apporte ainsi sa pierre à l'édifice, le *maintener* décidant ainsi de consolider celle-ci avec la proposition de l'utilisateur.

Cependant, cette méthode ne résout pas entièrement les actions effectuées en parallèle sur un fichier, et ne permet pas de rendre plus compréhensible la modification apportée par un utilisateur, même si ce dernier a reçu l'approbation du *maintener*.

1.3.4 Le commit

Parmi les outils méthodiques, il y a ce que les développeurs appellent le *commit*. Le *commit* est un acte informatique qui consiste à valider formellement une modification du code source d'un logiciel[44]. Il permet en quelque sorte d'effectuer un "étiquetage" du code qui a été ajouté à la structure. Cette action est par ailleurs bien documentée : le *commit* s'accompagne ainsi d'un numéro de révision, du nom de l'utilisateur qui a effectué la modification et permet aussi d'ajouter un message optionnel (plus connu sous le terme de "log" chez les développeurs). Le *commit* indique également l'heure (« timestamp ») à laquelle a été réalisé la modification. Chacun des fichiers modifiés est ainsi décrit et documenté.

D'une certaine manière, les informations relatives au code source sont tracées, permettant alors de constituer une sorte d'historique de conception et d'évolution du logiciel. Bien entendu, pour effectuer une telle modification, il faut avoir eu les droits d'écriture par le *maintener,* qui aura préalablement identifié le collaborateur. Cette pratique s'associe ainsi très bien avec la suivante et dernière fonctionnalité que nous allons voir, et qui constitue non seulement une fonctionnalité fondamentale de travail pour les développeurs, mais aussi l'axe principal structurant les plateformes de développement: le *versionning*.

[43] Voir glossaire
[44] Voir glossaire

1.4 LE VERSIONNING

Le *versionning* consiste à sauvegarder chacune des versions d'un fichier[45]. Il consiste à sauvegarder plusieurs versions d'un même fichier sans pour autant écraser les versions antérieures. Il est indispensable pour la production de fichiers à moyen et long terme. La méthode du *versionning* n'est d'ailleurs pas utilisée qu'en programmation, elle est également employée dans la production documentaire. A titre d'exemple, prenons la rédaction de ce mémoire : j'ai été amené à produire plusieurs versions écrites de mon document de travail. Étant donné que durant toute la phase d'élaboration du mémoire le plan de celui-ci est amené à changer régulièrement, du fait des réflexions effectuées sur le sujet mais aussi parce que la rédaction impose souvent une réorganisation des idées, j'ai été ainsi amené à revenir à des versions précédentes pour repartir d'une structure que j'ai faite évoluer dans une version plus récente, et dont je n'étais finalement pas satisfait. J'ai pu ainsi expérimenter plusieurs types de plans pour chacune des trois parties de ma réflexion.

Ce processus peut très bien s'appliquer à la production musicale, au montage vidéo et bien sûr à la rédaction du code source d'un logiciel, ainsi qu'à la production de sa documentation. En effet, celle-ci nécessite d'être corrigée et relue par différents individus, pouvant apporter chacun leurs critiques sur les diverses informations contenues dans le fichier. Par ailleurs, étant donné que les propositions sont faites sous forme de "blocs"de données, il est préférable d'employer un format d'édition simple pour pouvoir prendre en compte les éventuelles recommandations du *maintener*, mais aussi pour tester individuellement les blocs de données avant de les intégrer au code source principal. Le *versionning* peut alors s'avérer d'autant plus utile dans ces cas là.

Grâce au *versionning,* il est également possible de comparer des changements apportés au code source principal du logiciel, et dans le cas d'une défaillance, d'avoir la possibilité de revenir en arrière, à un état fonctionnel et stable du code source. Il peut ainsi s'effectuer manuellement ou alors être totalement automatisé, grâce à un système de gestion de version. Ce système contrôlera et générera chacune des versions d'un fichier[46], il sera ainsi possible de revenir à une version antérieure spécifique. Les systèmes de gestion de *versionning* se déclinent ainsi

[45] CHACON Scott, STRAUB Ben, *Pro Git : Everything you need to know about Git*, Apress, 2018, 517 p.
[46] CHACON Scott, STRAUB Ben, *Pro Git : Everything you need to know about Git*, Apress, 2018, 517 p.

sous trois formes: gestion de version en local, gestion de version centralisée, et gestion de version distribuée, qui correspondent plus au moins à une évolution chronologique, en fonction des besoins mais aussi des spécificités techniques des infrastructures des utilisateurs à un moment donné.

Les logiciels de gestion de *versionning* ont dû évoluer progressivement au cours des utilisations et des pratiques afin de palier à une multitude de problèmes que rencontraient couramment les développeurs. Identifier ces problèmes et les solutions qui sont apportées nous permettront d'avoir une première visualisation sur la conservation du code source.

1.4.1 La gestion de version en local

La première méthode, simple et basique, est la gestion de versions en local. C'est-à-dire que les versions d'un fichier sont sauvegardées sur une base de données unique directement sur la machine de l'utilisateur[47]**.** L'un des pionniers de la gestion de version en local se trouve être une production de l'association GNU, appelé GNU Revision Control System. Nous verrons en effet, qu'en plus d'être extrêmement présents sur les plateformes de développement, les contributeurs du logiciel libre sont les premiers acteurs de la conservation sur le moyen et long terme[48].

Les failles de la gestion de version en local sont multiples: les fichiers étant sauvegardés en local sur une seule et unique machine ne facilitent en rien l'accès à cette ressource à l'ensemble des collaborateurs du projet. De plus, si une défaillance devait survenir sur la machine sur laquelle sont conservés les fichiers, cela augmenterait drastiquement le risque de corruption des données, pouvant rendre alors les éléments versionnés illisibles. Dans le cas de projets collaboratifs, il a fallu donc mettre en place une base de donnée accessible par différents utilisateurs qui pourraient venir versionner leurs fichiers modifiés, et qui pourraient également consulter les fichiers d'autres développeurs.

1.4.2 La gestion de version centralisée

Afin de combler ce besoin, les développeurs mettent au point le système de gestion de versions centralisées[49]**.** Toutes les versions sont sauvegardées sur un serveur dont l'accès peut être contrôlé par le *maintener*. Certains logiciels s'appuyant sur cette méthode sont aujourd'hui encore assez populaires, et toujours exploités pour le

[47] Ibid.
[48] Nous prendrons le temps d'étudier un peu plus en détail ces dispositions mises en place dans la deuxième du mémoire.
[49] CHACON Scott, STRAUB Ben, *Op Cit.*

développement de nouveaux projets. C'est le cas notamment de Subversion, ainsi que Concurrent Version Systems (CVS), le premier cité étant d'ailleurs toujours employé dans le cadre de projet d'archivage de code.

Ici aussi, l'inconvénient est le même que pour la gestion de version locale : il n'existe qu'un seul et unique serveur disposant de toutes les données d'un projet. Tout comme il est fortement recommandé dans les normes d'archivage électronique, il est préférable de disposer de plusieurs espaces de stockage à des lieux relativement éloignés. Cette recommandation s'applique aussi à la conservation du code source, d'autant plus si l'objectif est d'archiver à long terme les informations. Ce système a tout de même permis de faciliter l'accès aux informations et d'ouvrir des possibilités de réplication. Il reste néanmoins perfectible, et c'est en cela que les logiciels de gestion de version distribuée ont été développés.

1.4.3 La gestion de version distribuée

L'évolution majeure et récente est le système de gestion de version distribuée, qui semble être la solution la plus efficace à ce jour. Nous pourrons évoquer ici des logiciels comme Git, Mercurial, Bazaar, pour ne citer qu'eux. Mais au-delà de leurs spécificités à chacun, c'est leur fonctionnement commun qu'il est intéressant ici d'étudier[50].

Ce système permet en effet de recopier entièrement le répertoire où sont sauvegardés les fichiers ainsi que tout l'historique. Les clones sont ici des backup des données, c'est-à-dire des représentations quasi-identiques de l'état d'un fichier à un moment donné. Le système est également accompagné de différents dépôts à distance optimisant ainsi le travail collaboratif grâce à l'exploitation de plusieurs workflows parallèles. Les données versionnées ne viennent pas concurrencer celles d'autres développeurs, chacun disposant de sa version de travail. Une copie du répertoire est également disponible en local sur les machines des développeurs, permettant à chacun de travailler offline, le répertoire se synchronisant à chaque connexion, prenant ainsi en compte les modifications apportées par l'utilisateur durant la période où il n'était pas connecté à internet.

Pour en revenir à notre thématique principale, à savoir les formes de conservation du code source, nous pouvons dès à présent noter que la gestion de

[50] Ibid

version fait partie de ces modules de conservation. La sauvegarde ayant lieu à plusieurs niveaux et par différents utilisateurs, et tous les états étant tracés, cela permet une première récupération du code source à différents états, évite que le code source ne soit altéré, et permet à chacune des productions individuelles d'être conservée et consultable. Ce sont les idées de chacun ainsi que les moyens divers et variés d'un développeur pour atteindre son objectif qui sont importantes et riches pour les productions futures[51].

1.5 LES PLATEFORMES DE DÉVELOPPEMENT

1.5.1 Les noyaux des plateformes de développement

Les logiciels de gestion de version les plus récents sont les noyaux fonctionnels des plateformes de développement les plus populaires sur Internet. Cela se voit notamment dans le nom porté par quelques-unes d'entre-elles : GitHub, GitLab, … L'analyse étymologique du nom de ces deux plateformes permet d'ailleurs de clairement identifier l'utilité première des forges de développements, ainsi que les modules qu'il est possible de retrouver sur chacune d'entre elles. Le logiciel de *versionning* est complètement transparent, laissant place à une interface web travaillée et personnalisée en fonction de l'utilisateur qui utilise la plateforme de développement.

Après avoir parcouru des plateformes comme GitLab, GitHub ou encore Bitbucket, il nous est apparu qu'il n'est pas forcément utile, dans le cadre ce mémoire, de spécifier précisément chacune des particularités de chacune des plateformes. Elles font partie des plus populaires et sont toutes trois développées grâce au logiciel de *versionning* Git. Les spécificités variables en fonction de la forge correspondent en grande majorité à des fonctionnalités propres au métier de développeur et qui facilitent leur travail.

Ce qu'il est néanmoins intéressant d'analyser, ce sont les offres que l'on peut appeler "premium" qui sont proposées aux équipes ou aux entreprises souhaitant disposer de privilèges supplémentaires sur la plateforme. Fait d'actualité sur lequel nous reviendrons plus tard dans notre développement, le rachat de GitHub[52] par Microsoft a boosté le nombre d'utilisateurs sur GitLab, ce dernier profitant de cette migration massive pour proposer les fameux services premiums propres à sa plateforme.

[51] DI COSMO Roberto, propos recueillis par David Larousserie, « Offrons aux jeunes des clés du pouvoir et de la liberté », *Sciences & Avenir,* 2009. [en ligne] Disponible sur : http://www.dicosmo.org/Media/2009-09-SciencesEtAvenir.pdf [consulté le 30/08/2018]

[52] "Rachat de GitHub : Pourquoi ce rachat et quels sont les plans de Microsoft ?" [en ligne] Disponible sur : https://www.developpez.com/actu/207642/Rachat-de-GitHub-Pourquoi-ce-rachat-et-quels-sont-les-plans-de-Microsoft/ [consulté le 31/08/2018]

Mais avant de nous plonger dans le modèle économique des plateformes, intéressons-nous d'abord à ce qu'elles sont. En effet, du fait de ce noyau commun, les forges de développement se placent comme l'outil le plus abouti, dont les développeurs ont toujours eu besoin. Chacune des personnes morales ou physiques dispose de son espace réservé, qu'il soit hébergé par la plateforme ou ayant une instance installée en interne. Toutes les fonctions que nous avons citées précédemment sont réalisables sur la forge, il est même désormais possible d'organiser son projet avec l'ensemble des membres de celui-ci, qu'ils soient développeurs ou non.

1.5.2 Un outil puissant et polyvalent

Commençons par nous intéresser aux généralités concernant la production de code source. Précisons à nouveau que les fonctionnalités rencontrées sur les plateformes sont similaires, les différences principales résidant dans la "philosophie" prônée par les créateurs de celles-ci: nous aurons l'occasion d'y revenir. En tout cas, il est possible désormais pour les développeurs d'assurer les différents rôles que nous évoquions précédemment.

1.5.2.1 Contrôles et habilitations

La gestion des droits et des habilitations pour le projet se font logiquement par le biais d'administrateurs qui sont capables de distribuer les accès à une partie spécifique d'un projet, mais également de définir si oui ou non l'utilisateur a le droit d'éditer un fichier. Limiter l'accès au code source reste un moyen utile pour les *maintener* de garantir un code informatique sain.

Il existe un système de « branche », jaillissant du tronc principal[53] (un *trunk* en informatique, la structure de base où est écrit le code source originel), au moment où un développeur se lance dans la création d'un nouveau fichier. Cette branche ne peut être liée au *trunk* qu'avec l'accord d'un administrateur permettant ainsi de rajouter une couche de contrôle sur les modifications proposées. Maîtriser ainsi les blocs de données susceptibles d'être ajoutés au code source permet aussi de limiter le nombre de versions de fichiers en erreur générés par la gestion de version. Étant donné que ces différentes versions sont supposées servir à revenir à

[53] *Voir glossaire*

un exemplaire stable du code source, et pour des raisons d'efficacité, il faut pouvoir garantir qu'une majorité des fichiers sont exploitables dans l'historique.

N'importe qui ne peut pas apporter de modifications au code. Cela peut paraître logique aujourd'hui avec les outils à disposition, mais auparavant lorsque les codes sources étaient versionnés en local, il était déjà plus difficile de chiffrer les données, c'est à dire de les sécuriser. Désormais, un certain nombre de systèmes d'information viennent restreindre par sécurité les possibilités des utilisateurs, pour qu'ils ne s'égarent pas dans des menus qu'ils ne sont pas habilités à manipuler.

Ce principe revient un peu sur les forges de développement: un administrateur peut décider de ce qu'un utilisateur basique a le droit de voir ou non en fonction du groupe et des projets auxquels il appartient. Dans les communautés Libre et Open Source, le code source est accessible gratuitement mais il n'est pas possible de le modifier depuis sa source. Un utilisateur extérieur au projet pourra cependant récupérer le code source pour le modifier à sa guise sur sa plateforme, au sein de son groupe, comme il est spécifiquement proposé dans les licences libres.

1.5.2.2 Visualisation des tâches et des étapes d'évolution

La plateforme de développement, quelle qu'elle soit, sert donc d'interface de communication entre plusieurs métiers amenés à évoluer dans le même sens pour trouver des solutions à des problèmes communs.

Il est possible par exemple, au sein de projets, de déposer des *Issues* (GitLab), tâches spécifiques assignées à un individu. Les membres du projet peuvent suivre l'avancement du projet grâce à ce système et travailler parallèlement sans risquer d'empiéter sur la tâche d'un autre membre de l'équipe.

Il est bien sûr possible d'aider la personne en difficulté avec ses *issues* pour lui alléger le travail et éviter les situations bloquantes. De plus, différentes granularités permettent d'enrichir la description de l'*issue*, facilitant l'identification et la compréhension de celle-ci, ainsi que son niveau de priorité dans la liste des tâches à accomplir. Chacun a ainsi la possibilité de visionner l'ensemble des tâches du projet, même celles ne concernant pas son secteur, cela facilitant ainsi les échanges entre les différents services.

1.5.2.3 La production documentaire

Nous l'avions vaguement évoqué, mais au sein d'un projet, et pour assurer un suivi et une compréhension optimale de celui-ci, les membres se doivent de travailler sur des fichiers de traitement de texte destinés à créer des manuels d'installation et d'utilisation des logiciels. Rappelons que dans le cadre des FLOSS, il est impératif de mettre à disposition un maximum d'éléments permettant à n'importe qui, en ayant les capacités, d'exploiter le code source et le logiciel sous licence libre. Les manuels doivent ainsi être très détaillés et régulièrement mis à jour pour assurer une cohérence avec les évolutions du logiciel. La plateforme de développement facilite ainsi la gestion des différentes versions d'un manuel et ne nécessite pas toujours l'intervention des développeurs. En effet, ces derniers peuvent déléguer la rédaction des manuels par d'autres membres de l'équipe habilités à le faire, permettant donc de débloquer du temps pour continuer à écrire du code.

Il est ainsi plus aisé pour un développeur de rédiger les manuels d'utilisation et d'administration étant donné qu'il a une parfaite connaissance de son code. Son rôle auprès des rédacteurs du manuel est de leur transmettre des indications et recommandations spécifiques qui faciliteraient le travail de rédaction. L'accès à ce fichier se faisant collaborativement, l'édition du manuel en temps réel effectué parallèlement par les développeurs et d'autres utilisateurs réduit le temps consacré à ce document.

1.5.3 Des environnements modulables et paramétrables

1.5.3.1 Les langages de programmation pour le traitement de texte

1.5.3.1.1 Le Markdown

Les utilisateurs ont la possibilité d'écrire en Markdown sur la forge et de compiler ce code via un module qui générera un document au format pdf ou odt. Les images quant à elles ne sont pas directement copiées dans le fichier en markdown mais sont liées à celui-ci via des liens inter-projets qui appellent des images importés sur la plateforme. Il est important dans ce cas de bien référencer la localisation de l'image. La compilation agit donc sur tous ces éléments pour ne concevoir qu'un seul et unique document.

Ainsi, sur la plateforme, l'information est accessible de manière « éclatée », c'est-à-dire qu'on ne retrouve pas systématiquement tout le contenu d'un projet à un seul et même endroit. En fonction des habilitations de chacun des utilisateurs, il sera possible d'aller modifier le fichier présent dans un dossier spécifique, ou bien simplement d'avoir la possibilité d'ajouter des images à un projet pour qu'ils soient associés par la personne ayant les droits d'édition sur un document. Ce modèle d'agencement de l'information au sein de la plateforme est à prendre en compte dans le cadre d'un archivage pérenne d'un projet. Nous préciserons pourquoi dans la troisième partie de ce mémoire.

1.5.3.1.2 L'hébergement de site web

Parmi les modules intéressants des plateformes, l'on retrouve également la possibilité d'héberger un site web, ou bien produire des documents dans un format spécifique à partir d'un langage informatique déterminé. De manière générale, la plupart des plateformes mettent à disposition des environnements de diffusion qui permettent de tester la fiabilité du code et de vérifier qu'il fonctionne correctement. Il est possible d'écrire son site directement sur la plateforme, qui viendra se compiler lors de cette simulation pour former un site accessible via un URL **(**https://github.com/git/git-scm.com, par exemple**)**. Ainsi, cela rejoint l'objectif de l'écriture collaborative: toutes les modifications à suggérer et que les individus peuvent apporter sont à effectuer directement sur la plateforme, et non pas sur un fichier html présent quelque part en local sur l'ordinateur d'un individu. Cela présente en effet de plus gros risques de dispersion.

1.5.4 La présence d'acteurs multi-secteurs

La documentation ne concerne ainsi pas seulement les manuels mais aussi d'autres typologies de document en fonction des besoins du projet. Un projet peut d'ailleurs ne concerner que la production de différents documents. Par exemple, dans le cadre de la rédaction d'un dossier de candidature à un agrément ou pour obtenir une certification quelconque, différentes compétences sont sollicitées, autant techniques que théoriques. le dialogue peut ainsi se dérouler au sein d'un espace numérique de travail dédié pour garantir l'harmonisation de la production documentaire. Dans la certification NF 461 pour l'archivage électronique, il est par exemple demandé de fournir le détail des modules permettant de vérifier la conformité des fichiers entrant dans le SAE, ainsi que les modules permettant de garantir la pérennité. Des détails techniques comme les procédures de traitement des outils utilisés pour ces modules doivent ainsi faire l'objet de rédaction entre l'équipe technique et l'équipe métier (les archivistes).

En bref, dans des domaines nécessitant l'intervention de deux compétences distinctes, il sera possible pour les entreprises de mettre à disposition des instances sur des plateformes de développement pour permettre la collaboration de leurs équipes. La présence du logiciel étant par ailleurs de plus en plus importante au sein de tous les secteurs, chacun des acteurs du marché tente de se l'approprier. Cela se ressent notamment sur les noms des multinationales que nous pouvons trouver référencées sur certaines plateformes[54].

1.5.4.1 Le tiers-hébergement de code source

Le tiers hébergement de code source est un tout nouveau service mis à disposition des entreprises. Celles ne disposant pas des ressources nécessaires pour proposer à leurs équipes des environnements de travail similaires aux forges en interne trouvent dans ce type de prestation une excellente alternative et profitent pour la plupart des avantages apportés par les communautés du Libre et de l'open Source.

D'autres privilégient justement la confidentialité de leurs informations et se dirigent plutôt vers des plateformes proposant ce genre de services, mais garantissant aux développeurs de logiciels propriétaires le respect de la confidentialité des projets qu'ils mettent en route sur les forges.

1.5.4.2 Une diversité d'utilisateurs pour une diversité d'utilisation

Par ailleurs, les différentes définitions que nous pouvons retrouver des plateformes de développement sur différents sites n'insistent pas exclusivement sur la rédaction du code source, et supportent les idées d'organisation et de communication autour des projets. Ainsi Roberto Di Cosmo dans son article [55] définit les forges comme "*une plateforme d'hébergement de projets logiciels dans un environnement web constitué d'un ensemble d'outils du travail coopératif et du génie logiciel pour le développement collaboratif et distribué de logiciels*".

Sur le site du projet PLUME[56] qui met à disposition des utilisateurs de la documentation spécifique au domaine informatique et de la recherche, l'objectif

[54] GitHub. *The largest open source community in the world* [en ligne]. Disponible sur : https://github.com/open-source [consulté le 31/08/2018]

[55] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

[56] PLUME, *Promouvoir les Logiciels Utiles Maîtrisés et Économiques dans l'enseignement supérieur et la recherche*, Site projet-plume.org [en ligne] Disponible sur: https://projet-plume.org/ [consulté le 31/08/2018]

des plateformes est d'offrir un espace d'échange permanent et de collaboration en ligne aux développeurs de logiciels, et un espace de distribution (versions publiques des logiciels développés) pour les utilisateurs. Elle permet ainsi de rassembler des projets de développeur, mais aussi d'autres personnes travaillant sur ces projets (utilisateurs, traducteurs, …). Cette dernière définition confirme l'argument que nous avons apporté précédemment sur la possibilité désormais de demander à des individus non aguerris au développement informatique de participer à des projets.

1.6 CONCLUSION DE LA PARTIE 1

Nous avons beaucoup insisté sur les communautés des FLOSS car la philosophie qu'elles prônent s'accordent avec le fait que le code source et le logiciel doivent être conservées sous des formes variées, permettant sa réutilisation efficiente par le plus large public. Le partage d'information grâce à internet a ainsi encouragé les développeurs à se munir d'outils permettant, d'une part, d'éditer et de gérer du code source de façon optimale, et d'autre part, d'avoir la possibilité de le diffuser et de le documenter.

Cette panoplie d'outils, auparavant indépendants les uns des autres, s'est au fur et à mesure améliorée pour répondre aux besoins croissants des développeurs, ainsi qu'au développement des espaces d'échanges virtuels. Les plateformes de développement viennent ainsi répondre à un besoin de plus en plus important de production et de gestion de l'information. Elles facilitent ainsi le travail des développeurs.

Cette utilisation s'est étendue également à d'autres possibilités: un projet est désormais non seulement défini par la rédaction d'un code source permettant de générer un logiciel, mais aussi le maintien de son bon état de santé tout au long de sa durée de vie. La communication est facilitée entre les différents corps de métiers, et la diffusion des informations via la retranscription des fichiers dans des formats compréhensibles et lisibles participe à la bonne interprétation du projet.

Il n'existe clairement pas de modules d'archivage spécifiques pour le code source ou les logiciels, mais parmi les moyens mis en œuvre pour gérer le code source et la manière de travailler des communautés Libre et Open Source, nous pouvons constater que le code source peut être conservé sous différents états. Le développement orienté objet ainsi que l'utilisation du commit et du versionning permettent ainsi de garder une trace de l'historique du code, tout en permettant de jalonner le cycle de vie du logiciel à différents stades de son évolution.

Intéressons nous désormais aux cas d'archivage, disons "hybride" du code source et du logiciel.

2 ETUDES DE CAS

Cette deuxième partie a pour vocation d'étudier des cas concrets de conservation de codes source et de logiciels. Nous avons déjà évoqué le fait que, sur les plateformes de développement, il n'y pas réellement de modules d'archivage à proprement parler. Il existe toutefois des méthodes de conservation, nous allons l'aborder dans un second temps, qui sont néanmoins performantes et surtout qui sont profitables aux communautés de développeurs sur internet. Il n'en a cependant pas toujours été ainsi.

En effet, les forges de développement, bien qu'étant des outils relativement récents, n'ont pas toujours proposé des services aussi performants. Elles ont même, dans certains cas, engendré l'isolement de ressources capitales, désormais difficilement exploitables. L'étude de ces plateformes "tombées en disgrâce" nous permettra d'appréhender le fait que malgré l'apparente stabilité des cadors du domaine (GitLab, GitHub, BitBucket), le code source reste une ressource très difficile à conserver et à identifier.

2.1 LA FERMETURE DES SERVICES D'HÉBERGEMENT DU CODE

Le travail collaboratif n'a en effet pas toujours été possible selon les dispositions que proposent les plateformes aujourd'hui: il a existé un certain nombre de plateformes, parfois dédiées exclusivement à un projet ou ne proposant de programmer que dans un seul langage informatique. Nous nous concentrerons ici sur deux cas spécifiques, dont la clôture des services a provoqué une réelle prise de conscience en ce qui concerne la nécessité de garantir les mesures de sauvegarde et d'archivage.

2.1.1 Google Code

En date du 12 mars 2015, la plateforme de développement de Google[57], Google Code, est subitement clôturée. En effet, ayant été lancée en 2006, elle ne dispose plus d'outils assez performants pour concurrencer la nouvelle venue qu'était à cette époque GitHub. Le service proposé par Google ne permettait pas de s'adapter aux besoins immédiats des communautés de développeurs. GitHub est, selon l'article de Google[58] la raison principale de la fin du service proposé par la

[57] Google Open Source Code, *"Biding farewell to Google Code"* [en ligne]. Disponible sur : https://opensource.googleblog.com/2015/03/farewell-to-google-code.html [consulté le 31/08/2018]

[58] Google Open Source Code, *Op Cit.*

firme américaine. Google admet lui même dans son article d'adieu : "*we've seen a wide variety of better project hosting services such as GitHub and Bitbucket*", que les plateformes concurrentes proposent des services plus performants. La fin de Google Code ne résonne donc pas tant comme un échec mais plutôt comme un acte raisonné.

En amont de cette fermeture, un vaste mouvement de migration des projets est néanmoins organisé par Google pour permettre aux utilisateurs de récupérer leurs données avant la clôture, et d'encourager la réintégration de celles-ci sur de nouvelles plateformes de développement. De nombreux projets se retrouvent alors orphelins et ne sont pas réclamés par leurs propriétaires. En se connectant à l'url https://code.google.com/archive/search, il est toujours possible d'accéder à des fragments encore présents sur les serveurs de Google, mais aucune documentation, ni même d'historique ne permet de récupérer ce code dans sa complétude. En effet, aucun historique de conception exploitable n'existait du fait de l'absence d'un système de *versionning* qui aurait permis d'identifier les fragments de projets abandonnés. Cette fin, même anticipée, n'a donc pas permis de réaliser une sauvegarde optimale de la totalité des projets restants sur la plateforme de la firme américaine.

C'est grâce à l'outil de *versionning* principalement que les plateformes de développement sont des espaces efficaces de conservation du code et qu'ils permettent aux développeurs de travailler dans un environnement fiable, en toute sérénité, sans s'inquiéter des conséquences d'un tel événement. Enfin presque. Le plus étonnant reste tout de même que trois des logiciels de *versionning* qui aujourd'hui sont des références (Subversion, Mercurial et le très populaire Git), ont tous été hébergés sur la défunte plateforme. Au final, nous pouvons nous demander si Google n'a pas en quelque sorte "passé" le flambeau à GitHub, compte tenu de ses performances.

Si toutefois il devait advenir qu'une plateforme sous noyau Git connaisse la même conclusion que Google Code, il serait beaucoup plus simple de récupérer les données des serveurs, grâce justement au logiciel de gestion de versions. L'historique des projets seraient conservés sur différents espaces de stockage, permettant la récupération d'informations autant en local qu'à distance. Le versionning dans Git concerne aussi bien le code source écrit par les développeurs que tous les autres types de documents produits sur la plateforme (documentation

d'utilisation, d'administration, d'installation, documentation générale, images, schémas, etc.). Néanmoins, il faut toutefois nuancer: même avec un logiciel de gestion de version puissant, il reste difficile de proposer des services optimaux à des utilisateurs exigeants.

2.1.2 Gitorious

Gitorious est un cas intéressant. Nous affirmions plus tôt que Google Code n'avait pu continuer à proposer ses services notamment parce qu'il ne disposait pas de gestion de version aussi efficace que ses concurrents. En ce qui concerne Gitorious, c'est une plateforme de développement dont le noyau est lui-même fondé sur le logiciel de gestion de version Git. Elle est d'ailleurs le premier service de forge à exploiter le logiciel depuis 2007. Cette plateforme devrait donc disposer des mêmes avantages que les plateformes récentes. Mais Gitorious n'a pas non plus échappé aux atouts des forges modernes.

A l'arrivée de GitLab, de nombreux projets ont migré vers cette plateforme gratuite et disposant de modules plus performants[59]. Le manque d'utilisateurs et notamment d'utilisateurs payant un abonnement pour obtenir des accès privilégiés s'est fait ressentir, or c'est ce qui permettait de faire vivre la plateforme Gitorious. Ainsi, en 2015, les créateurs de Gitorious capitulent[60] et GitLab devient propriétaire de la forge. Tous les projets sont alors voués à migrer sur la plateforme de GitLab. Désormais, il est toujours possible d'accéder à la base de données de Gitorious via l'url https://gitorious.org/, mais ces projets sont non modifiables.

Il faut savoir également que la clôture de ce service a attiré l'attention d'une association : l'Archiveteam. Cette association a pour objectif d'entreprendre de vastes projets de récupération et de conservation de sites webs « mourants » et voués à disparaître sur internet. Elle s'est intéressée à Gitorious et a participé activement à sauvegarder la trace du site, qui autrefois accueillait le service de tiers hébergement[61].

D'une certaine façon, GitLab se place comme une version évoluée de Gitorious. Rolf Bjaanes, directeur des opérations chez Gitorious, a admis lui même que « *GitLab résolvait les mêmes problèmes de Gitorious, mais d'une meilleure manière* »[62]. Symptôme encore, au même titre que la plateforme Google Code, de la fulgurante évolution des services. Cette évolution marque, d'une part, la capacité des fournisseurs

[59] dzecniv, *Gitlab achète Gitorious,* Site linuxfr.org, 2015. [en ligne] Disponible sur: https://linuxfr.org/news/gitlab-achete-gitorious [consulté le 31/08/2018]

[60] Ibid.

[61] ArchiveTeam, *gitorious valhalla*, Site gitorious.org [en ligne] Disponible sur : https://gitorious.org/ [consulté le 31/08/2018]

[62] dzecniv, *Op Cit.*

de service à proposer des outils toujours plus performants, mais d'autre part, elle souligne aussi le caractère "figé" des forges de développement dans leur structure. C'est-à-dire qu'à chaque fois qu'un nouvel hébergeur propose des modules innovants, les forges déjà en place n'ont plus pu concurrencer les nouvelles venues, et capitulent en proposant à leurs milliers d'utilisateurs une migration vers un nouvel environnement.

Ce modèle d'obsolescence se retrouve dans les progrès fulgurants de certains domaines industriels comme les éditeurs de consoles ou les constructeurs d'ordinateurs. Les séries de produits anciens sont délaissées pour la nouvelle génération. Les technologies des anciens modèles sont peu à peu abandonnées, entraînant l'inexploitation des informations auxquelles ils permettaient d'accéder. D'un point de vue scientifique, c'est une perte considérable d'un savoir que nous ne serons peut-être jamais capable de reproduire et d'exploiter, si le code source n'est conservé que sous sa forme brute (c'est-à-dire sa trace écrite).

Les plateformes de développement peuvent être identifiées comme des systèmes d'exploitation géants. Elles exploitent à la fois le *commit*, le *versionning*, la génération de documents, proposent des environnements de tests fonctionnels. La plupart de ces fonctionnalités s'appliquent automatiquement (le *commit* et le *versionning* s'appliquent à la moindre action de modification) mais aussi dans une transparence facilitant le travail des équipes. Les forges présentent ainsi une complexité dans l'imbrication de toutes ces différentes fonctionnalités.

Cette complexité semble cependant nuire à ses possibilités d'évolutions. Pour reprendre l'exemple de la cathédrale et du bazar[63], il est bien plus aisé de proposer une rénovation de telle ou telle échoppe d'un bazar plutôt que de restaurer la nef principale d'une cathédrale monumentale. Il existe néanmoins bien des patchs de correction qui sont régulièrement déployés et qui assurent la bonne stabilité de l'environnement, mais rien à voir avec une mise à jour majeure qui nécessite beaucoup plus d'investissements et de temps.

Pour dire un dernier mot sur le cas Gitorious, il a été proposé aux utilisateurs, avant l'échéance finale, de migrer leurs données vers la plateforme GitLab. La question en suspens désormais est de savoir à quoi pourrait bien servir les dépôts restants de Gitorious, sachant qu'ils ne sont que consultables, et non modifiables. Nous avons déjà légèrement évoqué les potentiels pertes scientifiques engendrées

[63] RAYMOND Eric S., *Op Cit.*

paradoxalement par l'évolution des forges de développement. Mais c'est un concept beaucoup plus précis que nous nous efforcerons d'expliquer **à la partie correspondante** et qui introduit entre autres le logiciel en tant que patrimoine.

2.2 LES PLATEFORMES AUJOURD'HUI

Intéressons nous désormais aux plateformes de développement populaires. Nous partirons principalement des exemples de GitHub, BitBucket et surtout de GitLab. Depuis une dizaine d'années environ (basé sur la fin de Google Code et de Gitorious), elles sont très largement exploitées par les communautés de développeurs déjà présents sur internet mais aussi par une diversité d'entreprises. De manière générale, elles présentent des fonctionnalités similaires. En nous connectant sur les sites internet respectifs de GitLab, GitHub et BitBucket, les fonctionnalités principales sont l'hébergement du code, la gestion collaborative, les environnements de tests ainsi que les possibilités de déploiement du logiciel[64]. Les développeurs choisissent donc de travailler sur l'une ou l'autre des plateformes en fonction de leurs affinités avec la philosophie en vigueur sur une forge de développement. Comparons pour cela GitHub, qui totalise à lui seul près de 85 millions de projets[65], et GitLab, qui affiche une moyenne de 100 000 entreprises collaborant sur sa plateforme.

2.2.1 Un choix éthique

La différence entre ces deux concurrentes réside dans la caractéristique du code source de chacune : GitHub dispose d'un code sous licence propriétaire et GitLab d'un code source libre. Cela n'empêche pas la grande majorité de la communauté de développeurs à collaborer sur Github (les chiffres cités précédemment parlent d'eux-mêmes). Néanmoins l'utilisation d'un service propriétaire pour développer un projet libre peut court-circuiter la "chaîne de fabrication" d'un FLOSS[66]. GitHub propose en effet un environnement de travail agréable mais ne facilite en aucun cas la migration vers une autre plateforme. Son code source n'étant pas libre, il est difficile de savoir comment est imbriqué le code. Les utilisateurs ne peuvent ainsi pas procéder à une migration fiable et totale sans passer par les administrateurs de la plateforme. De cette façon les développeurs présents sur GitHub sont indirectement incités à rester sur la plateforme, une fois qu'ils y sont installés.

[64] GitLab. *Documentation* [en ligne]. Disponible sur : https://docs.gitlab.com/ [consulté le 31/08/2018]

[65] GitHub. *The largest open source community in the world* [en ligne]. Disponible sur : https://github.com/open-source [consulté le 31/08/2018]

[66] CHENET Carl, *Le danger Github (revu et augmenté)*, Site carlchenet.com, 2016. [en ligne] Disponible sur : https://carlchenet.com/le-danger-github-revu-et-augmente/ [consulté le 30/08/2018]

Le deuxième inconvénient réside dans les différentes informations relatives à un projet qui transitent sur la plateforme et qui peuvent être captées par GitHub pour des raisons de traçabilité: les fichiers versionnés et les traces de *commit* peuvent par exemple se retrouver stockés sur les espaces de stockage de GitHub via les processus mises en place par GitHub. L'extraction de ces informations est tout à fait possible par les utilisateurs, mais pas automatique. S'il devait advenir que GitHub ferme ses services un jour, l'entreprise, du fait des lois qui régissent son statut de solution propriétaire, peut refuser le rapatriement de ces informations.

Troisième point à prendre en considération, elle est également en droit d'exploiter les données traitées, sachant qu'elles sont pour la plupart produites dans le cadre de projet FLOSS. En ce qui concerne GitLab, le code source étant sous licence libre, il est possible de récupérer le code source de la solution et de l'étudier afin de l'adapter aux besoins de chacun de ses utilisateurs. Si une équipe n'est pas satisfaite d'un ou plusieurs éléments de la plateforme, ils peuvent plus facilement trouver des solutions de substitutions. De cette façon, cela permet à une entité exploitant la plateforme de rester indépendant et de savoir que ses libertés seront respectées, en accord avec la philosophie du Libre.

2.2.2 Les plateformes au service de la recherche

Cette recherche d'indépendance s'exprime notamment par la présence d'institutions de recherche pointues qui tentent d'éviter de voir leurs précieuses données confisquées. Nous pouvons ainsi citer la NASA, autrefois hébergée sur Gitorious, mais aussi l'université de Washington. Nous pouvons en effet déduire que pour la NASA, par exemple, le développement de programmes puissants fait partie intégrante de ses activités, ce qui justifie tout à fait sa présence sur une telle infrastructure.

Cependant n'oublions pas, nous parlons ici d'une entreprise internationale disposant de moyens conséquents. Bref, une entité colossale qui aurait tout à gagner à investir dans sa propre plateforme de développement. Les forges sont désormais de véritables espaces d'échanges pour les équipes, tous les corps de métiers peuvent s'y côtoyer et profiter des compétences de chacun. Pour dérouler notre exemple, au sein de la NASA, nous retrouverons donc des astronomes, des physiciens,des mathématiciens, des informaticiens, des archivistes, … Tous ces

services doivent collaborer ensemble pour fournir des réponses à des questions d'envergure universelle.

La forge représente ainsi un espace de gestion et de stockage de connaissances remarquable, qui est vouée à être développée et exploitée en continu. Les institutions de recherche profitent ainsi des travaux de leurs confrères tout en faisant profiter des leurs à l'ensemble de la communauté. Cette qualité à proposer un catalogue ouvert de projets est d'autant plus marquée par cette possibilité d'explorer les projets sur Gitlab par exemple **(**https://gitlab.com/explore**)**. Il est ainsi possible de se balader entre les différentes productions de différentes équipes et de télécharger le code source des projets FLOSS.

Ces plateformes représentent donc aujourd'hui une colossale bibliothèque où sont gérés les codes sources d'une multitude d'entreprises internationales, entreprises et institutions dont les côtes se chiffrent parfois en million. Si ces entreprises se tournent vers les plateformes de développement pour concrétiser leurs projets, c'est que les forges répondent désormais à un réel besoin et qu'elles permettent de mettre les informations en sécurité.

2.2.3 Le cycle de vie des FLOSS

Dans le cadre du Libre et de l'Open Source, nous avons pu voir que les pratiques d'écriture collaborative doivent permettre à chacun des utilisateurs de manipuler un code source pour avoir la possibilité d'apporter son expertise. Pour ce faire, et en dehors du cadre des plateformes de développement, le code source doit être conservé dans des conditions optimales garantissant son intégrité.

La méthode du "bazar"[67] explicitée plus avant permet en partie de garantir qu'un code source persiste dans le temps. Pour reprendre l'exemple de Linux, il est aujourd'hui le noyau d'un panel extraordinairement large de systèmes et de logiciels qui répondent à des besoins utilisateurs. De cette manière, le code source produit par Linus Torvalds, même s'il est modifié, devient transversal, et les bases de sa réflexion perdurent dans le temps, rebondissant de projets en projets. Mais les différentes entités, si elles ne sont pas conservées dans des conditions optimales (avec historique de développement, *versionning* des fichiers, documentation détaillée), ne permettent pas une exploitation optimale dans le temps, réduisant ainsi les possibilités d'évolution du logiciel tout au long de son utilisation[68].

[67] RAYMOND Eric S., *Op Cit.*
[68] DI COSMO Roberto, propos recueillis par David Larousserie, *Op Cit.*

2.2.4 La connaissance du passé pour mieux comprendre le futur

Étant donné qu'au sein de la communauté du Libre et de l'Open Source le savoir de chacun est mise à disposition librement, il est supposé permettre à chacun de profiter des expériences passées pour éviter de repartir de zéro. Il s'agit d'un principe que nous pouvons retrouver dans tous les domaines, et qui nous est relaté dans l'Histoire, grâce aux vestiges du passé. L'une des raisons de la conservation de ces vestiges, c'est bien de comprendre la manière de penser des communautés présentes avant nous et de profiter du savoir qu'elles ont développé, pour toujours capitaliser les connaissances et s'améliorer.

A défaut de prendre un exemple qui va dans le sens de notre propos, prenons le cas inverse : il nous est très difficile de savoir comment les populations d'Amérique centrale parvenaient à élaborer des calendriers précis, car selon leurs coutumes, la transmission du savoir était très majoritairement orale. L'Histoire a fait que ces civilisations ont aujourd'hui disparu, ne laissant à leurs plus proches descendants et aux chercheurs qu'une infime partie des connaissances qu'elles avaient développé, ne permettent pas aujourd'hui de profiter des trouvailles de ces défunts savants.

Aujourd'hui il n'existe pas encore de modèles d'infrastructure idéale permettant de préserver le code source et de valoriser le développement d'un logiciel, ainsi que toutes ses composantes essentielles à sa bonne exploitation. Disposer ainsi d'infrastructures permettant de collecter et d'enrichir sur le long terme un logiciel permet de garantir la longévité de celui-ci et de l'adapter aux besoins des utilisateurs durant un laps de temps indéterminé.

2.3 DES CENTRES DE DOCUMENTATION VIRTUELLES

Avant l'arrivée des plateformes, les logiciels ne disposaient pas toujours de serveurs dédiés et versionnés. Une fois que le logiciel entre en production, une version de celui-ci est mise à disposition des utilisateurs via le site internet des protagonistes du projet. Les différentes versions du logiciel sont ainsi diffusées via ce site qui devient le point d'échange entre les utilisateurs et les développeurs.

Les communautés ont su s'organiser pour non seulement sauvegarder sur la durée le code source d'un logiciel spécifique, mais aussi parce qu'ils proposent une documentation complète et disponible librement pour les utilisateurs. Les cas que

nous allons étudier sont des modèles dans ce genre. Ils sont de véritables centres de documentation (au sens de bibliothèques du code source). Ces dispositifs présentent la mise en place concrète d'un certain nombre de mesures pour assurer la bonne conservation du code, ainsi qu'une bonne diffusion de celui-ci, sous une forme de donnée adéquate. Ils existent par ailleurs maintenant depuis plus longtemps que les plateformes de développements, restent activement productives et n'ont connu à ce jour aucune fuite de données. Mais avant de nous intéresser de près à ces cas spécifiques, faisons un point rapide sur la notion de *package*.

2.3.1 Le package

Le *package* ou *packaging* est une notion importante dans le cadre de la conservation du code source et d'un logiciel, car elle peut concerner plusieurs choses et est étroitement liée à la gestion du code.

Le *packaging* ou package est en fait l'enveloppe à l'intérieur de laquelle nous pouvons retrouver le code source ainsi que de la documentation permettant d'installer, d'administrer et d'utiliser le logiciel qu'il régit[69]. Il fait ainsi partie intégrante des objets à conserver car il permet la compréhension de l'ensemble de la structure du code. et permettra ainsi de définir précisément quel type de moyens peuvent être mises en place pour garantir son interopérabilité avec une structure ou un système d'exploitation spécifique. C'est sous cette forme qu'il est bien souvent proposé aux utilisateurs de télécharger les fichiers relatives au logiciel. Il peut être mis à disposition sur un site internet ou bien directement sur la plateforme de développement sur laquelle le logiciel a été développé. Le *package* ne peut ainsi être livré simplement avec le code à l'intérieur n'apportant aucune explication sur son fonctionnement et sa mise en place.

Cette notion de *package* est également essentielle pour comprendre l'intérêt d'archiver le code source du logiciel[70]. Étant donné que le *package* contient non seulement le code source mais aussi toute sa documentation, il permet déjà d'appréhender les premiers raisonnements logiques du développeur qui a produit le code. Le *package* contiendra également des détails sur les systèmes d'exploitations compatibles participant dans le même temps à une meilleure compréhension de ces systèmes.

[69] DI COSMO Roberto, ZACCHIROLI Stefano, *Op Cit.*

[70] Ibid.

La principale cause d'obsolescence d'un logiciel ne provient pas toujours des fonctionnalités limitées qu'il propose, car ces dernières peuvent être améliorés au fur et à mesure. Cependant, les systèmes sur lesquels ils sont supposés fonctionner sont amenés à évoluer, nécessitant que le code source soit modulable et adaptable pour qu'il puisse être exploité de façon optimal dans la durée. C'est probablement le problème le plus compliqué à résoudre car il ne dépend pas seulement de contraintes techniques mais aussi de volontés différentes d'évolution des équipes en charge de faire évoluer leur produit. La version X d'un logiciel peut être compatible avec la version Y d'un système d'exploitation, et ne plus être compatible avec la version Z car les développeurs de ce dernier ont changé le raisonnement logique de la structure de leur code, nécessitant ainsi une refonte plus ou moins importante d'une partie du code du logiciel en version X. Bien évidemment, ceci est un exemple volontairement exagéré, mais néanmoins susceptible de se produire. En tout cas, pour exemple concret, nous pouvons citer le cas des disquettes de sauvegarde sur ordinateur, ou encore les cassettes VHS, tous deux des dispositifs autrefois extrêmement utilisés dont la production des systèmes permettant de les lire s'est aujourd'hui arrêtée[71].

2.3.2 Comprehensive Tex Archive Network (CTAN)

La Comprehensive Tex Archive Network (CTAN)[72] a vu le jour au début des années 90 pour promouvoir le langage de programmation Latex. Ce dernier permet de faire du traitement de texte pointu et très spécifique aux besoins de l'utilisateur. Il est notamment utilisé dans les sciences où formules, schémas, algorithmes, calculs et texte se côtoient, et permet ainsi de constituer un document harmonieusement structuré. Bien maîtrisé, il permet à son utilisateur de disposer d'outils de présentation plus flexibles et plus personnels qu'un traitement de texte classique.

Comme tous les autres codes source, il est appelé à évoluer perpétuellement, notamment parce qu'il appartient à la communauté des FLOSS mais aussi parce qu'il est utilisé au sein d'institutions pointues dans leurs domaines, qui y voient un avantage évident pour rendre plus performant Latex.

[71] ROCHEREUIL Chloé, *R.I.P magnétoscope ! La production de lecteurs VHS va être stoppée*, Site mashable.france24.com, 2016. [en ligne] Disponible sur : http://mashable.france24.com/tech-business/20160721-arret-production-magnetoscope-vhs [consulté le 31/08/2018]

[72] CTAN, Comprehensive Tex Archive Network [en ligne]. Disponible sur : https://ctan.org/ [consulté le 31/08/2018]

Les développeurs ont ainsi développé, de part et d'autres du globe, différents *packages* compatibles avec Latex. Ces modules sont non seulement fonctionnels mais aussi, du fait de leur très grande diversité, permettent à chaque utilisateur de trouver la solution qu'il souhaite mettre en place dans le cadre de son quotidien. Le problème, c'est qu'ici les données sont éparpillées, sur internet ou bien au sein d'institutions. L'idée est donc de rassembler en un seul et même lieu la totalité des informations tenues à ce jour sur Latex et de la rendre disponible à tous.

Cette documentation géante a pris place avec le projet CTAN qui, aujourd'hui encore, dispose d'une communauté active[73]. Le modèle proposé par les initiateurs du projet est donc le suivant : à partir d'un seul et même serveur, on peut dupliquer les informations de ce serveur sur d'autres serveurs (les *mirrors*) afin de garantir à la fois une bonne distribution du contenu, mais aussi une duplication de "sécurité" permettant de récupérer les données si un des serveurs venaient à tomber. C'est également un moyen de faire participer la communauté à la bonne gestion d'un projet.

2.3.3 Comprehensive R Archive Network (CRAN)

Initié par la GNU, cette collaboration est constituée de plusieurs serveurs à travers le monde, régis par le protocole FTP (File Transfer Protocol) qui permet de transférer ou de télécharger des fichiers[74]. Les informations stockées sur les serveurs concernent ainsi le langage de programmation R, notamment utilisé pour le calcul de statistiques et de graphiques. L'environnement R permet donc en quelque sorte de manipuler une grande variété de données brutes, et il est principalement employé par les centres de recherches (comme le CNRS ou les universités par exemple). Le site propose donc d'accéder aux archives via ce que les *mirrors* qui sont des copies Les *mirrors* étant ainsi répartis sur différents espaces du globe, ils permettent un accès confortable aux données, peu importe où nous sommes situés sur Terre. En fonction de la distance qui sépare l'utilisateur au *mirror,* le temps de connexion est plus ou moins long. En France, ce sont, entre autres, des instituts de recherches qui se sont proposées pour héberger les données. Le CNRS étant l'un d'eux disposant de sites physiques dans les villes où l'institution est implantée.

Le système de participation pour le développement du langage R est défini par une sorte de politique instaurant un ensemble de spécificités que le package doit respecter avant d'être proposé aux administrateurs. Cela nous permettra notamment de

[73] CTAN, Comprehensive Tex Archive Network [en ligne]. Disponible sur : https://ctan.org/ [consulté le 31/08/2018]

[74] CRAN, Comprehensive R Archive Network [en ligne]. Disponible sur : https://cran.r-project.org/ [consulté le 31/08/2018]

comprendre un peu plus comment les projets qui ne sont pas développés sur des plateformes peuvent continuer d'évoluer, en sachant que ce fonctionnement s'applique aux autres « CXAN ». La politique indique ainsi que pour proposer une modification, il faut transmettre un code source déjà rédigé et à priori fonctionnel, sous la forme d'un *package*. Le fameux *package* qui regroupe ainsi toutes les informations nécessaires pour la bonne compréhension du code source rédigé par un individu. C'est ensuite l'administrateur qui étudiera la proposition dans son ensemble et qui décidera de son intégration ou non dans le code source du logiciel. Ces administrateurs ont la responsabilité de maintenir le bon état de santé du code source. De cette façon, les données relatives au code source et au logiciel sont constamment surveillés et stables et aucun module "parasite" ou instable n'est ajouté au dépôt.

Pour dire quelques mots sur les méthodes de récupération des informations, il a été mentionné plus avant que les informations étaient transmises via un protocole FTP. Les utilisateurs Linux doivent procéder à une manipulation de récupération des fichiers sur les serveurs via le terminal de leur poste. Les utilisateurs Windows et Mac téléchargent le *packaging* directement via un lien sur le site internet. Cela génère une copie des fichiers, que l'utilisateur peut récupérer sur son poste et exploiter individuellement. Ce principe de récupération d'une copie conforme du fichier source se retrouve au sein de divers systèmes d'informations, mais aussi au sein des forges de développement.

Il existe néanmoins, là encore, des problèmes vis à vis de ce dispositif. Le premier concerne le logiciel de gestion de version employé. Pour développer leurs logiciels, l'équipe du projet a choisi d'utiliser Subversion, qui est un logiciel de gestion de version centralisé. Pour rappel, la gestion de version centralisée permet d'enregistrer les différentes versions d'un fichier sur un seul serveur. Ce serveur dispose ainsi de toutes les données correspondantes au programme R. Ainsi, en cas d'incident sur le serveur ou de détérioration de celui-ci, toutes les informations correspondantes au logiciel disparaissent et ne peuvent plus être récupérés. D'une certaine façon, la duplication des données sur les *mirrors* permet de palier à ce problème, mais nous allons voir qu'en réalité ces *mirrors* répondent à une logique de synchronisation régulière avec le serveur source. Ce dispositif s'applique aux trois cas que de Comprehensive "X" Archive Network que nous étudions dans ce mémoire. Nous en détaillerons le principe dans la partie suivante.

Le deuxième problème majeur réside dans le mode d'accès aux données. Le seul point d'entrée pour accéder aux données est le site web de l'association. Si, pour quelque raison que ce soit, le site internet est inaccessible, il faut impérativement penser à maintenir un lien vers les données ou du moins en aménager un pour être capable de récupérer les éléments conservés sur le serveur principal.

2.3.4 Comprehensive Perl Archive Network (CPAN)

Tourné autour du langage de programmation Perl, la Comprehensive Perl Archive Network (CPAN) propose des paquets de modules capable d'exploiter de manière optimale les fonctionnalités du langage Perl. Ici aussi, l'utilisateur dispose de plusieurs serveurs localisés dans des lieux différents sur la surface du globe. Le CPAN propose aux membres de sa communauté de devenir un *mirror*, c'est-à-dire d'héberger des données du site en local chez eux[75]. Le détail de ce dispositif et et les conditions d'accès à ce dernier sont scrupuleusement détaillée sur le site de la CPAN, la procédure est la même sur les cas précédemment étudiés.

Il y a deux conditions capitales avant de devenir un *mirror* : disposer d'un espace disque suffisant, au moins 50 Go, et d'un internet connecté non-stop. Les instructions de mise en production du serveur sont décrites dans une feuille de route.

Trois étapes capitales sont ainsi identifiables :

- tout d'abord la mise à disposition des données via un serveur et un protocole de transfert FTP permettant de télécharger les paquets. Les administrateurs du site préconisent ainsi de passer par des serveurs Apache et donnent ainsi la bonne syntaxe permettant de rediriger les utilisateurs vers les archives du CPAN. Il est également fortement recommandé que le logiciel employé dans le cadre des téléchargements des données soit suffisamment documenté, afin de permettre aux administrateurs d'effectuer des maintenances en cas de besoin mais aussi pour que les utilisateurs comprennent facilement comment accéder aux informations.

- le deuxième point consiste à synchroniser les disques vierges avec ce que les administrateurs appellent le "*cpan primary node*" (en français : le noeud primaire du CPAN). Nous pouvons comprendre par là qu'il s'agit probablement des données originales qui définissent les différents paquets développés à ce jour. Ainsi, chaque instance dispose de sa propre copie authentique des données

[75] CPAN, Comprehensive Perl Archive Network [en ligne]. Disponible sur : https://www.cpan.org/ [consulté le 31/08/2018]

originales, évitant d'avoir une copie d'une copie. Il est ainsi certain que le *mirror* mis en place récupère les versions authentiques et validées par les *mainteners*, créant ainsi une réplique parfaite. Ce processus fonctionne grâce à un logiciel commun (rsync) installé d'une part sur le *mirror* et d'autre part sur le serveur primaire de la CPAN.

- la troisième et dernière étape consiste à mettre en place ce que l'on appelle un "*job*" c'est-à-dire une tâche répétitive qui lancera le logiciel rsync et qui permettra de mettre à jour régulièrement le *mirror*. Cette tâche doit avoir été enclenchée au moins une fois par jour pour assurer la bonne conformité des données présentes sur le *mirror*.

Ces trois actions menées bénévolement par les communautés de développeurs sont à ce jour les seules centres de documentation exclusivement dédiées à un logiciel. Une liste de différents *packages* est accessible, permettant d'aller chercher l'information pour l'ajouter en local. Ces centres de documentations sont de véritables espaces d'enrichissement et de partage profitant à tous. Cela est en effet utile à de grandes institutions, très spécialisées pour la plupart, comme les universités ou les centres de recherches. Ces institutions peuvent ainsi enrichir à leur tour l'archive s'ils souhaitent également développer de nouveaux modules plus spécifiques à leurs besoins, et ce, avec l'aide de la communauté et de toutes les informations archivées sur les serveurs.

2.3.5 Des méthodes différentes en fonction des projets

Ces trois "archives" valident en quelque sorte le bien fondé de la volonté de valorisation du code source. Ce dernier n'est pas exclusivement exploitable comme un pur produit technique. Il est en constante évolution et chaque individu peut correctement se l'approprier s'il a, à sa disposition, les informations nécessaires pour appréhender la logique syntaxique structurelle du code, ainsi que la logique intellectuelle de son auteur originel.

Ces cas que nous venons d'étudier se retrouvent aussi dans d'autres projets plus contemporains. Par exemple, le projet Debian[76], qui comme Linux est devenu le noyau principal de développement d'un grand nombre de logiciels, dispose d'un site internet dont la structure s'apparente grandement aux projets de

[76] Software in the Public Interest et autres, *debian, The universal operating system*, Site debian.org, 2018. [en ligne] Disponible sur: https://www.debian.org/index.fr.html [consulté le 31/08/2018]

Comprehensive X Archive Network. Ici nous retrouvons un grand nombre de paquets permettant d'installer des logiciels adaptés à la solution Debian, et qui permettent d'exploiter toutes les qualités du système.

Autre cas intéressant et d'une toute autre envergure: l'archive Linux Kernel. Cette dernière met à disposition une documentation détaillée d'installation et d'utilisation du système d'exploitation. En plus de cela, elle propose des liens vers la liste des mails archivés dont le sujet principal sont des échanges entre les différents développeurs du projet. Ils discutent ainsi des modifications apportées et à proposer à l'avenir, tout en gardant une trace des patchs appliqués. C'est une manière différente mais intéressante, peut-être plus personnelle, moins dirigée vers les membres externes au cercle des développeurs ayant déjà toutes les habilitations sur le projet.

Ces différents cas nous permettent ainsi de voir, d'une part, que les communautés de développeurs ont su mettre en place différentes méthodes de diffusion et d'évolution de leur projet, et d'autre part qu'ils sont capables de s'organiser de façon pérenne en dehors des plateformes de développement. Différents espaces de stockage sont sollicités et répliqués pour permettre une diffusion optimale, et indirectement une sécurité supplémentaire, réduisant les risques de perte de données. Néanmoins, les pratiques diffèrent en fonction des communautés et l'information peut être difficilement accessible à un large public.

Dans tous les cas, cette information est réemployée dans le cadre de projets de grande ampleur au sein d'organismes spécialisés, démontrant bien que de tels dépôts documentaires, même lorsqu'ils ne concernent qu'un unique logiciel, sont de véritables sources d'informations, vecteur de développement et d'évolutions pour tous les membres faisant partie de la communauté d'utilisateurs.

2.3.6 Archive et conservation sécurisée

Soyons clair : les principes de l'archivage reposent sur un ensemble de normes strictes qui permettent de conserver une donnée dans des conditions assurant sa pérennité, son intégrité et son authenticité. Une fois archivée, l'information ne peut être en aucun cas altérée, ou alors dans des cas exceptionnels certaines métadonnées seulement peuvent être modifiées.

Nous avons déjà démontré que ni les communautés et ni les plateformes ne garantissent un tel traitement des informations. Le code source peut être ainsi dupliqué ou modifié un nombre incalculable de fois au cours de son cycle de vie au sein d'une

forge. Il aurait été possible d'archiver strictement un document ".txt" à l'intérieur duquel serait rédigé la totalité d'un code source, mais l'intérêt reste relativement faible dans le cadre du logiciel, ou du moins c'est insuffisant. S'il était conservé sur ce modèle, il y a de fortes chances qu'en le ressortant quelques années plus tard, nous n'ayons plus la possibilité de le faire fonctionner du fait de l'évolution des systèmes d'exploitation.

Dans le domaine du jeu vidéo, les principes d'archivage ne se limitent pas uniquement aux jeux sur ordinateur mais aussi aux consoles. Sachant que c'est un marché en perpétuel évolution et dont les systèmes évoluent très vite (les écarts de production des nouveaux modèles se rétrécissent de plus en plus**)**. Il faut ainsi proposer des environnements similaires aux systèmes qui permettaient de faire fonctionner les jeux. Les jeux vidéos étant des logiciels, ce principe s'applique tout à fait à l'objet de ce mémoire. De plus, le code source transporte avec lui un ensemble de connaissances tirés de diverses disciplines, parce qu'être développeur aujourd'hui n'implique plus exclusivement faire des calculs mathématiques pour des programmes de mesures[77]. C'est aussi faire parler sa créativité (web design, jeu vidéo, cinéma) et son ingéniosité (intelligence artificielle, robotique) mais également développer son esprit critique pour la résolution de bugs et l'identification des solutions. Chaque développeur décide de sa propre interprétation des instructions qu'il doit transmettre à la machine pour obtenir un logiciel spécifique. Il existe en effet plusieurs moyens pour arriver à la même conclusion, certains s'avérant plus efficaces que d'autres. En ce sens, l'archivage au sens strict ne peut pas s'appliquer au code source: déjà parce qu'il n'existe pas de moyens qui permettent de capturer le code source dans son enveloppe globale, mais aussi parce qu'archiver le code reviendrait à restreindre ce qui fait sa particularité, c'est-à-dire sa capacité d'évolution et d'adaptation.

Les initiatives "d'archivage" de logiciel et de code source à l'heure actuelle privilégient donc plutôt l'optique de conserver le code source dans des conditions garantissant sa stabilité (éviter que du code ne soit malencontreusement supprimé par exemple), sans pour autant restreindre son extension. Cette extension peut néanmoins être favorisée par l'effort collaboratif, régulier et significatif, des membres d'une communauté qui se consacrent à exploiter le potentiel maximal d'un logiciel et de son code.

[77] Voir introduction

Nous nous concentrerons donc sur trois actions valorisantes d'un programme informatique, qui sont des modèles de partage et de conservation du savoir.

2.4 LE CODE SOURCE COMME OBJET DE SAVOIR

2.4.1 La recherche en informatique

La richesse conceptuelle autour du code source est en fait à l'étude depuis plusieurs années maintenant[78]. Ces études contribuent à mettre au jour tous les impacts que peuvent avoir les logiciels sur notre quotidien, évoquant même une notion patrimoniale du logiciel. Des associations comme l'Institut National de Recherche en Informatique et en Automatique (INRIA) ont permis notamment de définir plus précisément ce à quoi peut se référer la notion de logiciel, et pourquoi il est important de le sauvegarder pour optimiser le développement des nouvelles technologies. Ils soutiennent notamment un projet important en ce sens, dont nous reparlerons en fin de partie.

De nos jours, le logiciel fait partie intégrante de notre quotidien. Nous en utilisons à l'intérieur de nos smartphones, de nos ordinateurs, dans des consoles de jeux vidéos, dans les voitures... Le constructeur automobile Tesla l'emploie notamment à un tout autre niveau dans ses voitures, dans des "super régulateurs", (il prévoit par ailleurs de partager son code source aux communautés des FLOSS[79]).

La connaissance fondamentale des logiciels est ainsi une valeur ajoutée à notre quotidien et nous permettrait de mieux appréhender les appareils qui nous entourent, tout en gardant le contrôle et notre libre arbitre sur les produits qui nous sont proposés.

C'est ainsi qu'est en train de naître un projet titanesque, qui a tout récemment ouvert ses portes[80], pour archiver, diffuser le code source à tous: la Software Heritage.

Ce projet se place comme pionnier dans le domaine de l'archivage du logiciel. Avant d'entrer concrètement sur ce sujet, abordons rapidement sur d'autres problématiques liées.

[78] MATTHEWS Brian, SHAON Arif, BICARREGUI Juan, JONES Catherine, *A framework for Software Preservation,* The International Journal of Digital Curation, e-Science Centre, Science and Thechnology Facilities Council, Rutherford Appleton Laboratory, Oxon, UK, 2010. [en ligne] Disponible sur : http://www.ijdc.net/article/view/148/210 [consulté le 31/08/2018]

[79] CLAUDEL Maxime, *Tesla publie une partie du code source des technologies équipant ses voitures,* Site Numerama.com, 2018. [en ligne] Disponible sur : https://www.numerama.com/tech/376834-tesla-publie-une-partie-du-code-source-des-technologies-equipant-ses-voitures.html [consulté le 31/08/2018]

[80] INRIA, *Robert Di Cosmo, acteur du libre et porteur du projet Software Heritage, Portrait,* Site inria-alumni.fr, 2017. [en ligne] http://www.inria-alumni.fr/roberto-di-cosmo-software-heritage/ [consulté le 31/08/2018]

2.4.2 La fragilité du code

Les chercheurs en informatique se sont en premier lieu penchés sur la fragilité difficilement saisissable du logiciel. En effet, le développement d'un logiciel passe par plusieurs étapes ainsi qu'entre plusieurs mains. Chacun des utilisateurs vient enrichir le code via diverses lignes de caractères dont le raisonnement diffère de celui effectué en amont, au-delà de plusieurs couches d'étapes effectuées. Dans le cadre d'un grand projet, il faut par exemple penser à bien informer un nouvel arrivant sur le projet ou alors mettre à sa disposition les informations suffisantes pour tenter de s'écarter le moins possible de la logique exprimée par ses prédécesseurs. Des lignes de codes rédigées à des intervalles de temps importants peuvent provoquer des dysfonctionnements sans forcément empêcher le logiciel de fonctionner. Ces différents cas participent à la fragilité intrinsèque du code, en faisant une structure composite et complexe parfois fébrile. Il est certain qu'aujourd'hui ce problème est moindre du fait du succès des plateformes et des modules qu'elles proposent pour permettre une gestion optimale du code. Le risque est néanmoins toujours présent au vu de l'importance croissante des projets qui sont développés.

2.4.3 Les forks

Il existe cependant un autre cas qui pourrait rendre le code fragile: dans le cadre d'un développement de logiciel moins centralisé et plus éparpillé, il peut arriver qu'un même logiciel soit développé parallèlement avec des modules supplémentaires propres à chacune des deux productions. Dans ce cas, ce sont deux équipes différentes qui travaillent sur le projet, donc deux manières de penser différentes qui coexistent sur une même base. Les communautés de développeurs appellent cela le *fork*.

Dans sa définition[81] : le *fork* est un nouveau logiciel, un virage, une évolution technologique créée à partir du code source du logiciel existant. Le *fork* est la résultante d'une divergence de points de vue et d'objectifs entre des développeurs liés originellement à un même projet. Ce genre de divergence peut en effet créer des tensions au sein de la communauté. Du fait de ces tensions, les *forks* ne sont pas toujours aussi populaires que leurs homologues originels (le modèle sur lequel s'est construit le *fork*).

[81] Voir glossaire

Le groupe de développeurs travaillant à la base sur le même projet se scinde ainsi en deux, réduisant le potentiel intellectuel capable d'enrichir le code source. La collaboration permettant d'aboutir à des projets comme les Comprehensive X Archive Network nécessite cependant une communauté soudée, capable de travailler ensemble, même dans l'opposition des orientations de chaque individu. Se lancer dans le développement d'un *fork* c'est donc diviser la communauté et devoir en bâtir une nouvelle autour d'un projet qui peut faire beaucoup parler de lui par son statut.

2.4.4 Archive « hybride »

Certaines pratiques de conservation semblent tout à fait fiables, comme les Comprehensive X Archive Network. D'autres sont moins centralisées et prennent ainsi parfois des risques dans la gestion de leurs données.

Proposer une bibliothèque du code, c'est notamment l'objectif que la Software heritage souhaiterait atteindre, c'est-à-dire en proposant un parfait mélange entre pérennisation des logiciels et de leurs données relatives (code source, documentations), et puis une diffusion optimale à partir d'une base de données fiable et accessible à tous. Tout ceci sous licence libre, mettant en avant la collaboration.

2.5 LE LOGICIEL : QUELS FRAGMENTS ARCHIVÉES ?

2.5.1 Les enjeux

Comme précisé plus avant, la moindre fonctionnalité dont nous nous servons sur nos ordinateurs, smartphones, consoles, tablettes etc. sont des logiciels conçus pour nous faciliter la vie au quotidien et nous permettre d'exploiter au mieux les systèmes sur lesquels ils sont installés. Que ce soit pour travailler, se divertir, s'informer, ils font désormais partie intégrante de notre vie. La moindre interaction que nous pouvons avoir avec une machine passe par un logiciel, ou tout du moins par un programme avec lequel l'être humain est capable d'interagir.

Les enjeux de l'archivage du code source sont étroitement liés à la notion du code qui va au-delà de l'objet. Il est perçu par les chercheurs en informatique mais aussi Richard Stallman[82], comme une création de l'esprit humain qui reflète la manière de penser de celui-ci[83], véhiculant ainsi un savoir spécifique. Au-delà du code en lui-même

[82] Voir partie 1

[83] STALLMAN, Richard M., *Free Software, Free Society : Selected Essays of Richard M. Stallman*. Boston : Free Software Foundation, 2015. 305 p. [en ligne] Disponible sur : https://www.gnu.org/doc/fsfs3-hardcover.pdf [consulté le 30/08/2018]

ce sont aussi tous les éléments qui ont permis son développement, ainsi que l'ensemble des informations permettant de comprendre sa structure physique, qu'il est nécessaire de conserver. Une analogie particulièrement parlante pour illustrer est la recette de cuisine[84]: il s'agit d'un assemblage d'ingrédients qui, par un enchaînement d'actions bien spécifiques, permettent de constituer le plat. Il existe une grande variété de possibilités et différentes façons d'arriver au même résultat. Un cuisinier différent est libre d'ajouter le ou les ingrédients de son choix pour apporter sa touche personnelle. C'est la même chose pour le code source : chaque développeur dispose des ses propres méthodes de travail et utilise le langage de programmation qu'il souhaite pour développer son logiciel. Il existe également différentes syntaxes dans le code qui, selon la préférence du programmeur, sont autant de moyens différents pour aboutir à un même résultat. Certaines personnes préfèrent ajouter trois œufs dans leurs crêpes pour ne pas avoir besoin de laisser reposer la pâte, d'autres vont préférer économiser la matière première en réduisant le nombre d'œufs, mais perdront plus de temps en la laissant reposer. Il existe ainsi différents moyens de préparer des crêpes tout comme un développeur a la possibilité d'écrire son code de la manière qu'il souhaite. Autre analogie avec la mécanique vélo : il ne suffit pas d'avoir les pièces les plus sophistiquées sur une monture pour prétendre détenir la plus performante. Un système de freinage hydraulique sera d'une efficacité redoutable pour des adeptes de descentes sportives à vélo, mais il est d'une utilité moindre s'il s'agit de rouler en ville, sur des routes plates, d'autant plus qu'il faudra savoir les entretenir, voir les changer, ce qui n'est pas à la portée de tout le monde. C'est pareil pour le code : rien ne sert de disposer de modules complexes et de l'associer à une structure tout aussi complexe si le développeur ne dispose pas des connaissances nécessaires pour entretenir son code source et proposer des évolutions effectives.

Ces deux analogies nous permettent à nouveau d'appuyer le fait que le code et le logiciel sont des composantes similaires à d'autres secteurs d'activités, et que le savoir obtenu jusqu'à maintenant dans ces autres secteurs s'est fait notamment grâce à l'accumulation d'expériences. En effet, la cuisine et la mécanique sont des activités ancestrales et qui ont, au fil des siècles, connu respectivement des évolutions raffinées (gastronomie et cuisine moléculaire) et fulgurantes (progrès technologique dans les secteurs industriels).

[84] Code Source, Site wikipedia.org [en ligne] Disponible sur : https://fr.wikipedia.org/wiki/Code_source

Comme pour un livre de cuisine, il est intéressant d'avoir à disposition un ensemble de "recettes" permettant de créer des logiciels pour ainsi savoir quel langage, quelle syntaxe utiliser, quelles astuces facilitent et optimisent la réalisation et de connaître la composition de tous les "ingrédients" pour en tirer le meilleur. Ces astuces peuvent se matérialiser comme des commentaires dans le code,ou alors des explications du fonctionnement d'une syntaxe complexe mais aussi des solutions apportées par d'autres développeurs dont la diversité de point de vue ne peut qu'enrichir les deux parties concernées dans l'échange.

Pour poursuivre notre raisonnement, en cuisine, lorsqu'un gourmet goûte un plat, il est ainsi éventuellement capable d'identifier les différents ingrédients qui composent le plat, mais il est beaucoup plus difficile pour lui de deviner l'enchaînement des actions qui ont permis d'assembler les ingrédients. Ainsi, même lorsque le code source est accessible librement, il peut s'avérer compliqué de comprendre les formules employées pour dialoguer avec la machine, car ces dernières sont justement liées à la réflexion personnelle d'un ou plusieurs individus. D'où l'importance d'avoir des données descriptives et informationnelles qui ne font pas réellement partie du corps du code source, mais qui peuvent s'avérer tout aussi enrichissantes que le code lui-même et qui permettent par ailleurs de le mettre en valeur.

L'archivage du code source n'implique donc pas simplement la conservation du code source sous sa forme brute mais aussi des informations descriptives essentielles pour sa bonne compréhension, comme dans tout archivage électronique. Nous insisterons donc bien que le code source, en tant qu'objet, n'est pas suffisant et nécessite d'être archivé avec l'ensemble des éléments constituant ses *packages*.

2.5.2 « L'enveloppe » du code à archiver

Comprenons bien donc, avant d'entrer dans le détail sur ce projet, dans le cas de la Software Heritage, il y a plusieurs paramètres à prendre en compte pour proposer un archivage du code source et des éléments gravitants autour. Le code en lui-même n'est pas le seul élément à archiver. Le *package* en entier doit être pris en compte pour permettre une pertinence optimale de l'archive.

Ainsi, des deux grandes "méthodes" d'archivage électronique, la migration et l'émulation, c'est en effet cette dernière qui doit être utilisée dans le cas des codes sources et des logiciels.

2.6 La Software Heritage et son Archive

2.6.1 La recherche

Les acteurs majeurs en faveur de la conservation et de la diffusion du code source et des logiciels sont principalement des chercheurs et des membres de la communauté des FLOSS. L'INRIA qui est initiateur et principal sponsor de la Software Heritage, travaille ainsi avec un ensemble de collaborateurs mathématiciens et informaticiens. Par le terme générique d'informaticien, nous parlons des chercheurs en informatique, c'est-à-dire qui maîtrisent aussi bien les notions techniques que théoriques du domaine de l'informatique. Différents profils de ces chercheurs peuvent par ailleurs être consultés via l'url : https://www.softwareheritage.org/people/. Ces chercheurs travaillent donc à réfléchir à toutes les notions englobant les systèmes informatiques, certaines ayant été citées précédemment dans ce mémoire (conservation à long terme, fragilité du logiciel, utilisation étendue des plateformes de développement, enjeux de l'archivage électronique…). Ils réfléchissent ainsi à des méthodes de médiation et de sensibilisation auprès des experts de domaines variés, mais aussi auprès des publics. De cette façon, ils espèrent promouvoir les ressources que peuvent apporter l'informatique et internet. Les chercheurs ayant travaillé sur le projet de la Software Heritage sont par ailleurs de fervents adhérents des FLOSS et partagent la même philosophie que les communautés du Libre et de l'Open Source. Le projet d'archive du code source s'inscrit ainsi dans cette continuité, confortée par la notion de science ouverte.

2.6.2 La science ouverte

La science ouverte (ou *Open science* en anglais)[85], est un concept récent qui s'est développé au sein des communautés de chercheurs. Le principe consiste ainsi à rendre disponible les résultats des recherches à l'ensemble de la communauté, et de les valoriser afin qu'elles puissent être analysées et réexploitées. L'intérêt ici étant d'être capable, à partir des résultats d'une expérience, de répondre à d'autres questions toujours plus pointues sur un sujet.

[85] Plan national pour la science ouverte [en ligne], Disponible sur : http://m.enseignementsup-recherche.gouv.fr/cid132529/le-plan-national-pour-la-science-ouverte-les-resultats-de-la-recherche-scientifique-ouverts-a-tous-sans-entrave-sans-delai-sans-paiement.html [consulté le 31/08/2018]

Les publications scientifiques sortent des institutions de recherche pour toucher d'autres communautés de chercheurs, voir un plus large public. La science ouverte s'applique ainsi très bien à la conservation et à l'exploitation des logiciels. En effet, les ressources employées pour développer tel ou tel projet ne sont plus sollicitées si elles ont été correctement documentées et conservées dans des conditions permettant la récupération et l'exploitation du code source. Cela permettrait théoriquement de développer des logiciels plus complexes et plus performants.

C'est ainsi que, dans une optique à la fois patrimoniale mais aussi de performance, le projet Software Heritage voit le jour pour mettre à disposition des utilisateurs d'internet une véritable bibliothèque universelle du code source sous licence libre.

2.6.3 Le patrimoine

Nous l'avons déjà dit, le logiciel fait désormais partie intégrante de nos vies, au point qu'il est de plus en plus reconnu comme un élément patrimonial[86]. Selon le Centre National de Ressources Textuelles et Lexicales (http://www.cnrtl.fr/definition/patrimoine), le patrimoine se définit comme "*Ce qui est transmis à une personne, une collectivité, par les ancêtres, les générations précédentes, et qui est considéré comme un héritage commun*". Le logiciel est ainsi, par comparaison, une réalisation humaine en évolution constante, qui, grâce aux progrès technologiques, a profondément changé le quotidien des individus de la fin du XXe et du début du XXIe siècle. Le numérique faisant également partie intégrante de nos vies, le logiciel est l'outil utilisé pour conserver et continuer à diffuser toutes les productions immatérielles produites ou enregistrées par des périphériques de captures (appareil photo, caméra vidéo, smartphone, tablette,ordinateur, …). La compréhension du logiciel en tant qu'outil a donc un double intérêt, faire avancer la recherche dans tous les domaines, et garantir un accès pérenne au patrimoine numérique (patrimoine vidéoludique, cinéma, musique, etc.).

2.6.4 La bibliothèque d'Alexandrie du code source

C'est ainsi que la Software Heritage s'impose comme la première bibliothèque universelle du code source. Elle est par ailleurs souvent comparée à la bibliothèque d'Alexandrie. Cette dernière était connue pour sa collection de documents incroyables et étant un lieu d'échanges et d'apprentissage pour les érudits de l'Antiquité. Les sources ne permettent toujours pas aujourd'hui de prouver le bien fondé des dires concernant la

[86] Software Heritage [en ligne]. Disponible sur : https://www.softwareheritage.org/?lang=fr [consulté le 31/08/2018]

bibliothèque d'Alexandrie et sa titanesque collection, cependant, les récits qui sont parvenus jusqu'à nous sur la richesse du savoir qui y était entretenue a inspiré les créateurs du projet Software Heritage.

L'infrastructure doit répondre à trois objectifs. Ces principes fondateurs sont par ailleurs originalement représentés par le logo du projet (par ailleurs intelligemment animé sur la page d'accueil du site[87] :

- Les triangles internes pointent vers le centre du logo, représentant l'action de collecte provenant des quatre coins du monde. Les données ainsi récupérées forment une seule et même entité dont les différentes parties gardent leur indépendance.

- Les traits horizontaux à la base des triangles et pouvant s'apparenter au slash entre deux balises (< / >) sont les barrières représentant les moyens mis en place pour assurer la bonne conservation des données au sein de l'infrastructure.

- Enfin, les triangles pointés vers l'extérieur représentent la "redistribution" du savoir. Par ailleurs, la schématisation de l'Archive de la SHW ne dispose d'aucune fermeture, indiquant ainsi bien l'intention de proposer cette titanesque bibliothèque à tout un chacun, du moment que celui-ci souhaite consulter les informations qui y sont archivées.

En effet, l'Archive serait disponible sous licence libre, et même les codes source de logiciels propriétaires seraient acceptées.

Ne perdons pas de vue que la très grande majorité des logiciels sont aujourd'hui développés sur les plateformes de développement. Cela n'a pas échappé à la Software Heritage qui a déjà archivé le contenu des forges disparues dont nous avons parlé en début de cette partie. En effet, l'Archive dispose ici des fragments orphelins restés sur les serveurs de Google Code et Gitorious, permettant ainsi de sauvegarder le code afin qu'il puisse faire partie de la bibliothèque et permettre aux utilisateurs de les consulter dans un ensemble d'éléments correspondants entre eux.

[87] Software Heritage *Op Cit.*

2.6.5 Le principe de fonctionnement

La méthode de capture des codes sources en accès libre se base à peu près sur le même principe que nous avons étudié dans les C"X"AN. En effet, un module de la bibliothèque a été conçue pour aller balayer l'ensemble des répertoires des plateformes de développement, dont le code source est en libre accès. De cette façon cela permet à la SHW de récupérer les nouveaux projets, mais aussi potentiellement de mettre à jour les projets déjà archivés, mais qui auraient évolués entre deux balayages. L'Archive se met ainsi à jour régulièrement et permet d'assurer la "fraîcheur" des données qui sont susceptibles d'être conservées et exploitées sur la plateforme.

Par cette méthode, ce sont actuellement tous les dépôts publics de GitHub qui sont régulièrement balayés et récupérés. Une synchronisation est également effectuée avec les paquets source de la distribution Debian, ainsi que les distributions du projet GNU[88].

Les graphiques des données déjà archivées dans la solution affichent des chiffres rocambolesques : près de 4,7 milliards de fichiers sources, un peu plus d'un milliard de *commits* et 84 millions de projets sont présentes sur l'Archive[89]. Les graphiques affichent des courbes à la hausse, et étant donné que l'initiative s'apprête à élargir son périmètre de balayage à d'autres plateformes de développement, ce sont quelques milliards d'autres données qui viendront s'ajouter.

Cette bibliothèque titanesque est ainsi entièrement consultable via une API (cf source) qui permet d'explorer les différents projets susceptibles de contenir les caractères que nous avons entrés dans la barre de recherche. En venant cliquer sur un des liens listés, nous sommes redirigés vers un *snapshot*[90] de l'entièreté du projet, nous permettant d'aller dans chacun des répertoires de celui-ci et d'inspecter le moindre recoin de l'archive. Il est également possible de télécharger l'entièreté du projet, comme sur une forge de développement.

Intéressons-nous désormais aux typologies de données contenues par la SHW, qui permettent justement d'exploiter dans le détail les différents projets.

2.6.6 Le contenu

Le balayage automatique des plateformes récupère concrètement la totalité du code source ainsi que tous les fichiers versionnés du projet depuis sa création jusqu'au moment du balayage. Nous détaillerons nos propos plus précisément en troisième partie.

[88] Ibid.
[89] Ibid.
[90] Voir glossaire

Dans tous les cas, en fonction du logiciel de *versionning* qui est exploité, c'est tout un éventail de type de fichiers qui sont récupérés et qui forment la ressource principale de la SHW. En voici la liste exhaustive (liste consultable ici[91]):

- file contents - le code source, sans les métadonnées
- Directories - les différents répertoires qui permettent de rediriger vers les fichiers relatifs au code source qui peuvent être à l'intérieur de sous-dossiers
- revisions ("commits") - une liste des différentes modifications effectuées durant le développement du logiciel
- releases ("tags") - une version du logiciel exploitable

2.7 CONCLUSION DE LA PARTIE 2

L'archivage, à strictement parler, peut très difficilement être appliqué au logiciel. Cependant, les cas que nous venons d'étudier sont représentatifs de la diversité des méthodes de conservation à long terme du code source.

La clôture des plateformes Google Code et Gitorious a permis aux plateformes modernes de prospérer et de palier aux défauts de leurs prédécesseurs. Cette popularité est notamment identifiable par l'identification d'institutions scientifiques et d'entreprises spécialisées dans des domaines variés au sein des forges.

Un logiciel fédérateur et exploité par une très grande diversité d'utilisateurs (chercheurs, professeurs, auteurs, etc.) permet de valider le caractère collaboratif et étendu très marqué des communautés du Libre et de l'Open Source. Les C"X"AN en sont par ailleurs de parfaits exemples.

Ils proposent entre autres une méthode d'archivage "hybride", à mi-chemin entre l'archivage électronique classique et la sauvegarde simple. La centralisation des informations (issue par ailleurs d'un éclatement des lieux de sauvegarde) en un seul et même point via un site internet permet de proposer une documentation très riche, fournie par les utilisateurs, pour les utilisateurs.

[91] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

Cette idée de bibliothèque universelle est par ailleurs adoptée par le projet de la Software Heritage. Cette initiative a su identifier et capturer les composantes essentielles qui définissent un projet de développement d'un logiciel, pour mettre à disposition de tous, les codes source de millions de logiciels, partiellement ou pleinement exploitables en fonction de la complétude des données récupérées.

La Software Heritage a permis, entre autres, de mettre en application un ensemble d'études sur la conservation, l'identification et la restructuration des données relatives au code source. Nous nous baserons donc sur les composantes techniques de l'initiative de la Software Heritage pour développer la troisième et dernière partie de ce mémoire, tout en y ajoutant un ensemble de recommandations que nous déduirons des pratiques communautaires étudiées, ainsi que sur la nécessité de proposer un environnement performant mais évolutif pour répondre aux attentes de chacun.

3 RECOMMANDATIONS

L'initiative de la Software Heritage a permis d'identifier les différentes contraintes à prendre en compte dans la conservation du code source sur le très long terme. Les caractéristiques techniques de l'Archive permettent en effet la récupération, la conservation et l'identification de chacun des fichiers susceptibles d'être sauvegardé dans la bibliothèque.

3.1 UN VERSIONNING, UNE MÉTHODE DE RÉCOLTE SPÉCIFIQUE

Git, Mercurial, Subversion, Bazar, CVS sont différents exemples de logiciel de gestion de version parmi les plus populaires, mais il en existe bien d'autres. Nous l'avons vu, chacun propose en fonctionnalité principale de générer plusieurs versions des fichiers créés dans le cadre d'un projet, et donc de garantir un historique complet retraçant l'évolution du code source. Ces données peuvent, entre autres, être exportées des espaces de stockage où elles sont conservées. S'il existe bien une grande diversité de logiciels de gestion de version, ce sont principalement les gestion de version centralisées et distribuées qui sont le plus utilisées, donnant ainsi le choix aux équipes de développement, en fonction des besoins du projet.

Un logiciel de gestion de version centralisée est bien plus adapté à l'exploitation de données volumineuses, la concentration de celles-ci sur un seul et même serveur évitant d'avoir à les répliquer plusieurs fois, comme dans le cas de gestion de version distribuée. Cela provoquerait notamment des ralentissements conséquents. Néanmoins, la gestion d'un projet impliquant le traitement de fichiers moins volumineux, mais nécessitant tout de même un développement souple, privilégiera un logiciel comme Git ou Mercurial.

Chacune des solutions de versionning propose également des fonctionnalités spécifiques propres, dont l'intérêt dépend ici de la préférence des développeurs exploitant le logiciel en question. Le choix étant ainsi suffisamment vaste, chacun peut y trouver son compte. Les environnements collaboratifs de développement, dont font partie les forges, sont ainsi construites sur la base des solutions de contrôle de version. GitLab, fonctionnant sous Git, affichera quelques différences avec la plateforme supportée par la GNU Savannah[92] fonctionnant sous Mercurial,

[92] Free Software Foundation, *Savannah*, Site savannah.gnu.org [en ligne] Disponible sur: https://savannah.gnu.org/ [consulté le 31/08/2018]

bien que les deux solutions soient très proches[93]. Ces différences seront ainsi d'autant plus marquées entre GitLab et les initiatives de la C"X"AN, dont le logiciel employé est Subversion. Différents outils existent néanmoins pour permettre la mutation du corps d'un projet entre deux logiciels de gestion de version, git-svn étant l'un d'eux.

3.1.1 Et la Software Heritage dans tout ça ?

La diversité de ces formules impliquent pour l'initiative de l'Inria de prendre en compte chacune des spécificités des projets qu'elle récupère. Chacune des solutions de gestion de version doit pouvoir être totalement gérée par l'Archive. Le grand défi est donc de définir un un modèle de données générique et universel, commun à tous les systèmes de gestion de version[94].

Pour cela une solution a été mise en place par la SWH : le *listing* et le *loading*[95]. Le principe s'articule en deux étapes :

- le *listing* viendra par le biais du balayage, à la base de chacun des répertoires racine des projets, pour lister l'ensemble de l'historique du code source et de ses composants annexes. Cette liste exhaustive référence ainsi la totalité des fichiers et une partie des annotations et en capture une copie temporairement.

- c'est à ce moment qu'intervient le *loading* (ou chargement en français). La fonction extrait ainsi l'ensemble du code source de tous les fichiers précédemment listés pour les ajouter à l'Archive.

Remarquons qu'il existe un module de *loading* spécifique à chacun des systèmes de gestion de version, permettant ainsi de rapatrier le modèle de données dans sa forme la plus fidèle. Le *loading* dispose également d'un système de duplication intelligente, c'est-à-dire qu'il est capable de comparer et éventuellement de faire correspondre une donnée listée avec une déjà stockée sur

[93] PLUME, *Mercurail Hg : Gestionnaire de version décentralisé*, Site projet-plume.org [en ligne] Disponible sur : https://projet-plume.org/fiche/mercurial-hg [consulté le 31/08/2018]

[94] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

[95] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

l'Archive, évitant donc de récupérer à chaque fois la même information et générer des doublons[96].

Ce processus ingénieux permet donc à l'Archive de disposer non seulement de l'ensemble des données d'un répertoire racine, mais aussi de garder la structuration originale, imposée par le logiciel de gestion de version.

L'inconvénient désormais est que, comme nous l'avons vu, les plateformes de développement ainsi que les logiciels de gestion de version évoluent rapidement. Même si actuellement Git semble faire l'unanimité parmi la grande majorité de la communauté, il n'est pas impossible (voire probable), qu'un nouveau système de gestion de version apparaisse un jour et supplante ceux déjà en place. Cela nécessiterait ainsi de développer un nouveau modèle de *loading* pour chacun des systèmes de contrôle de version. Cela engendrerait également une charge de travail importante, d'une part parce que la SWH souhaite s'étendre à un plus grand ensemble des plateformes de développement (la seule étant compatible pour le moment étant GitHub)[97], mais aussi parce que l'évolution fulgurante des logiciels implique de toujours disposer des technologies les plus adaptées à chacune des solutions présentes ou futures.

Les ingénieurs en informatique sont ainsi appelés à collaborer étroitement pour permettre de développer un modèle de contrôle de version universel et propre à la Software Heritage[98]. Le code source de la SWH étant entièrement libre d'accès, chacun peut contribuer à améliorer l'environnement mis en place. Les utilisateurs sont par ailleurs vivement invités à collaborer au projet[99].

3.2 L'ORGANISATION DES DONNÉES

3.2.1 Merkle Direct Acyclic Graph

Le modèle qui a été choisi pour organiser cette bibliothèque géante et universelle est celui du Direct Acyclic Graph (DAG)[100]. Ce dernier permet aux données archivées dans l'environnement de stockage de s'accroître en continu et de disposer d'un

[96] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

[97] Software Heritage [en ligne]. Disponible sur : https://www.softwareheritage.org/?lang=fr

[98] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

[99] Software Heritage [en ligne]. Disponible sur : https://www.softwareheritage.org/?lang=fr

[100] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

identifiant unique pour chacun des noeuds. Un noeud représente les liens et les dépendances qu'il peut exister entre différents fichiers d'un même projet. De cette façon, chacune des branches du projet dispose d'un modèle de référencement unique à l'intérieur duquel différentes informations viennent. Par exemple, il pourrait s'agir de la date de création, la dernière date du *commit* et l'auteur de ce dernier[101], et potentiellement un éventuel commentaire permettant de décrire le fichier ou la modification qui lui a été apporté, ainsi que les relations qu'il peut entretenir avec d'autres fichiers. Ce modèle s'étend ainsi aussi bien aux fichiers simples qu'à l'ensemble des dossiers et sous-dossiers.

Ralph Merkle[102] qui a donné son nom à ce dispositif, l'a également agrémenté d'un système de cryptage permettant de sécuriser les informations échangées entre les plateformes de développement et la SWH. La cryptographie, qui concerne ainsi le domaine de la sécurité, étudie les processus d'échanges entre deux entités pour définir des moyens de sécurité pour permettre le bon acheminement des informations[103]. Cela permet de garantir que les informations, durant leur transition entre la forge et l'Archive, ne sont pas interceptées par des individus dans de mauvaises intentions.

A ceci, nous pouvons ajouter la fonction de hashage qui, en faisant office de signature électronique, complète ainsi le modèle d'organisation des données dans l'Archive.

3.2.2 Le hash de signature

Une fonction de hashage est une fonction qui peut être utilisée pour mapper des données de taille arbitraire à des données de taille fixe[104]. C'est à dire qu'il est possible de calculer l'empreinte d'un fichier provenant d'une plateforme de développement, peu important sa taille et son identification initiale. Cela permet de lui assigner un enchaînement de caractères alphanumériques long et unique, qui constitue sa fiche d'identité.

[101] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

[102] MERKLE, Ralph C., *Ralph C. Merkle*, Site merkle.com [en ligne] Disponible sur : **http://www.merkle.com/** [consulté le 31/08/2018]

[103] MERKLE, Ralph C., *Ralph C. Merkle*, Site merkle.com [en ligne] Disponible sur : **http://www.merkle.com/** [consulté le 31/08/2018]

[104] MERKLE, Ralph C., *Ralph C. Merkle*, Site merkle.com [en ligne] Disponible sur : **http://www.merkle.com/** [consulté le 31/08/2018]

Ce processus permet donc, indépendamment du nom du fichier d'origine, d'éviter d'avoir plusieurs entités ayant le même nom. Au sein des projets, sur les forges de développement, le *versionning* attribue déjà un identifiant unique aux différents fichiers versionnés[105], basés notamment sur la fonction de hashage. Cependant, cet identifiant ne sera unique qu'à l'intérieur d'un seul et même projet, chaque équipe de développeur disposant en quelque sorte de sa propre instance sur la plateforme de développement.

Même s'il est tout de même peu probable qu'un identifiant soit similaire à celui d'un autre fichier dans un autre projet contenu sur une autre instance (la représentation de l'identifiant étant relativement longue), la probabilité de rencontrer un identifiant similaire au sein de l'Archive est revue à la hausse du fait du traitement important de la totalité des données sur les plateformes. A l'avenir, la SWH souhaitant s'étendre à l'ensemble des forges existantes, le nombre de fichiers ne peut être vu qu'à la hausse, comme l'indiquent les graphiques[106]). Recalculer ainsi l'empreinte du fichier entrant, permettra de lui attribuer un identifiant unique et globalisant vis à vis de l'ensemble de l'environnement dans lequel il est nouvellement ajouté.

Cette organisation complexe des données permet ainsi à l'archive de disposer d'un modèle de données extensible en continu, dont les limites semblent être infinies. C'est un modèle qui correspond tout à fait à l'archivage des milliards de fichiers que l'infrastructure est supposée accueillir et référencer et qui pourrait convenir à d'autres dispositifs de centralisation du savoir.

3.3 LES MÉTADONNÉES

3.3.1 Le principe

Les métadonnées sont des données sur la donnée. C'est-à-dire que la métadonnée permet de typer et donner des informations, éventuellement un contexte, sur la valeur d'une donnée. Dans le cas du logiciel et notamment du code source, définir des métadonnées à la fois descriptives et informatives relève d'une certaine complexité.

La Software Heritage réfléchit encore actuellement à comment enrichir son Archive grâce à un modèle de métadonnée qui soit adapté à l'ensemble des fichiers qui doivent être conservés. Une réflexion a par ailleurs été amorcée en ce sens, via un

[105] GitLab. *Documentation* [en ligne]. Disponible sur : https://docs.gitlab.com/ [consulté le 31/08/2018]
[106] Software Heritage [en ligne]. Disponible sur : https://www.softwareheritage.org/?lang=fr

article[107]. L'auteur définit ainsi le processus de référencement via trois étapes principales :

- le créateur d'un document ou d'un fichier soumet sa création à une entité supérieure
- après différentes étapes de validation, le fichier est publié, ou mis en diffusion dans le cadre du logiciel, et l'entité supérieure lui attribue un identifiant
- si un utilisateur souhaite ainsi faire référence à ce fichier, il appelle ainsi ce dernier grâce à la métadonnée qui inclut l'identifiant de l'objet

L'auteur fait également la distinction entre une métadonnée récupérable et non récupérable. Dans le cas d'une métadonnée récupérable, l'identifiant attribué par l'entité supérieure a permis de se référer directement au fichier correspondant. Une métadonnée irrécupérable implique que l'identifiant n'est pas suffisant pour faire référence au fichier, il faudra donc passer par des informations secondaires comme le nom de l'éditeur du fichier, la date de diffusion, etc.).

3.3.2 L'application au logiciel

Dans le cas du logiciel, si la métadonnée principale n'est pas suffisante, il y a un certain nombre d'éléments à prendre en compte pour appeler la référence au sein de la plateforme. Rappelons-nous en effet que le logiciel est contenu dans une enveloppe qui transporte aussi l'ensemble des fichiers annexes qui lui sont liés (documentation, historique, …). De cette façon tout cet ensemble doit être à la fois identifiable dans sa globalité mais doit également permettre l'identification des différents fichiers qui composent l'enveloppe, indépendamment des uns et des autres. Il faudra donc aussi permettre d'apporter des métadonnées aux différents "noeuds" que peuvent engendrer les relations entre tous les fichiers.

Une des solutions proposées par l'auteur[108] serait de créer des métadonnées permettant de faire référence à un fichier en faisant correspondre sa localisation

[107] KATZ Daniel S., *Software Heritage and repository metadata : a software citation solution,* Site danielskatzblog.worpress, 2017. [en ligne] Disponible sur : https://danielskatzblog.wordpress.com/2017/09/25/software-heritage-and-repository-metadata-a-software-citation-solution/ [consulté le 31/08/2018]

[108] KATZ Daniel S., *Software Heritage and repository metadata : a software citation solution,* Site danielskatzblog.worpress, 2017. [en ligne] Disponible sur : https://danielskatzblog.wordpress.com/2017/09/25/software-heritage-and-repository-metadata-a-software-citation-solution/ [consulté le 31/08/2018]

sur la plateforme de développement et celle qu'il occupe dans la Software Heritage, par exemple par le biais d'une URL.

3.3.3 L'attribution de métadonnées en amont du processus de création

Une solution préconisée également par l'auteur de l'article dans sa conclusion[109] serait de créer directement un référentiel à l'intérieur de chaque répertoire, qui fournirait de cette manière l'ensemble des métadonnées auxquelles faire référence pour appeler les données correspondantes.

Cette idée est par ailleurs tout à fait cohérente : définir un certains nombre de règles à respecter dans la conception et la description du code source en amont faciliterait sa conservation en aval, que ce soit sur les plateformes de développement ou bien au sein de la Software Heritage.

3.4 DÉFINIR DE NOUVELLES RÈGLES DE CONCEPTION DU LOGICIEL

3.4.1 Au niveau des communautés

L'idée ici n'est pas de redéfinir l'ensemble de la chaîne de production du logiciel, loin de là. Nous ne disposons pas des compétences pour proposer une telle refonte, en sachant que les pratiques qui ont été développées ont mises du temps à se mettre en place et qu'elles ont fait leurs preuves. Cependant comme nous l'avons fait remarquer tout au long de ce mémoire, le code source et le logiciel sont sujets à divers facteurs menaçant leur exploitation (appropriation des données, fermeture des services d'hébergements, …). Nous avons également beaucoup insisté sur l'intérêt de conserver toutes les composantes liées au logiciel.

Au sein des communautés donc, il serait intéressant de centraliser et valoriser la documentation qui est produite. Cette dernière étant notamment éditée sous la forme d'un code informatique (Markdown, Json), les mêmes précautions devraient pouvoir s'appliquer au code du logiciel et au code des documents.

Parmi cette documentation, une fiche similaire à un carnet de bord permettrait à chacun de consulter l'avancement d'un projet. Même si les plateformes de développement proposent de suivre l'avancée du projet via différentes interfaces, il

[109] KATZ Daniel S., *Software Heritage and repository metadata : a software citation solution,* Site danielskatzblog.worpress, 2017. [en ligne] Disponible sur : https://danielskatzblog.wordpress.com/2017/09/25/software-heritage-and-repository-metadata-a-software-citation-solution/ [consulté le 31/08/2018]

n'existe pas de réels outils reprenant strictement les principes de la gestion de projet, et notamment du suivi d'avancement comme dans un diagramme de Gantt. Cette typologie documentaire apporterait un plus dans le suivi d'évolution d'un logiciel et serait produit pour chaque projet interne (évolutions mineures ou majeures, développement d'un module demandé par les utilisateurs, …). Ces "rapports" permettraient ainsi de granulariser plus précisément les étapes de confections du logiciel et donc, au moment de son archivage, d'enrichir d'autant plus la ressource conservée.

Autre point : la méthode du Bazar évoqué par Eric Raymond[110] s'est certes avérée efficace mais a eu tendance à faire "éclater" les différentes informations relatives au code source. C'est-à-dire que plus les projets grandissent, plus ils sont éparpillés. Ils ne profitent pas tous d'une visibilité légitime, qui elle ne devrait pas dépendre du succès immédiat d'un logiciel. En ce sens, il pourrait être intéressant de centraliser les informations ou de référencer chaque projet entre eux, à la manière que nous avons déjà évoquée pour la mise en place de métadonnées. C'est-à-dire qu'il faudrait créer un référentiel dont les métadonnées appelleraient les projets qui sont liés entre eux et qui disposent d'un noyau commun. Cette information supplémentaire participerait ainsi à une meilleure traçabilité des projets au sein des plateformes de développement, en participant à une communication plus large de toutes les branches du "bazar". De cette façon, les projets n'étant pas développés sur des plateformes et qui seraient susceptibles de ne pas voir leur travail conservé dans l'Archive, pourraient justement être mis en lumière de cette façon. Les bibliothèques étant des centres de recherches et de loisirs, l'exploration des rayonnages peut s'avérer très enrichissant pour déceler des perles.

D'autre part, la notion de besoin métier peut avoir évolué aujourd'hui, c'est-à-dire que le lancement d'un projet peut dépendre désormais d'autres facteurs. Dans le Libre, les développeurs ne sont pas spécifiquement contraints par le revenu apporté par la vente de leur produit, car il est gratuit, mais peuvent tout de même travailler pour des sociétés offrant des services connexes[111]. Rien n'empêche des équipes de développeurs de travailler ensemble sur des projets qui leur tiennent à

[110] RAYMOND Eric S., *The Cathedral and the Bazar*. États-Unis : édition « Libre », 1997. 241p. [en ligne] Disponible sur : http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ [consulté le 30/08/2018]

[111] STALLMAN, Richard M., *Free Software, Free Society : Selected Essays of Richard M. Stallman.* Boston : Free Software Foundation, 2015. 305 p. [en ligne] Disponible sur : https://www.gnu.org/doc/fsfs3-hardcover.pdf [consulté le 30/08/2018]

coeur, développer peut en effet être une passion, mais néanmoins les besoins des utilisateurs passent en priorité, sans quoi cela pourra être très fastidieux d'amener un projet à terme[112]. Il serait dommage de passer à côté de code finement structuré, apportant une véritable valeur ajoutée à d'autres types de projets. La traçabilité décrite plus haut permettrait de mettre un peu plus en lumière les projets "étouffés" par le "bruit" que peut générer la popularité d'autres productions.

3.4.2 Au niveau des plateformes

Les plateformes étant désormais des espaces de production et de consultation gigantesque, elles restent néanmoins fragilisées par l'obsolescence des outils qu'elles proposent. Les actualités de ces dix dernières années[113] ont démontré que les forges, dans leur structuration, ne permettaient pas d'évoluer significativement.

Dans la pratique, disposer d'une plateforme au code libre serait une aubaine pour les utilisateurs, car comme GitLab[114], cela permettrait à chaque équipe de personnaliser son instance comme il le souhaite. Certes, le travail serait conséquent au vu de la complexité de l'environnement[115], mais avoir la possibilité de retirer, modifier et faire évoluer des modules exactement comme un logiciel classique, permettrait d'ouvrir la porte à une meilleure adaptabilité des forges aux besoins évolutifs de ses utilisateurs.

C'est aussi reprendre les principes évoqués par la Software Heritage, c'est-à-dire, développer et enrichir le code source et le logiciel qui permet justement de conserver et exploiter l'ensemble des composants numériques que peuvent produire les êtres humains[116]. La plateforme de développement étant environnement de production et environnement d'exploitation en même temps, il convient que le code de celle-ci soit régulièrement amélioré pour maintenir le service sur le long terme et éviter de reproduire les phénomènes Google Code et Gitorious. A ce titre, Gitorious, qui fût la pionnière des forges de développement basé sur un noyau Git, a son code disponible sur la plateforme GitLab. L'étude de ce code et la comparaison de celui-ci avec le code des plateformes actuelles permettrait peut-être de mieux saisir les raisons de la fin du service, de revenir aux bases de la forge et pourquoi pas d'exploiter à nouveau le code source.

[112] BAR Moshe, FOGEL Karl, *Open Source Development with CVS*, 3rd Edition. Arizona : Paraglyph Press, 2003. 368 p.

[113] Google Open Source Code, "*Biding farewell to Google Code*" [en ligne]. Disponible sur : https://opensource.googleblog.com/2015/03/farewell-to-google-code.html

[114] GitLab. *Documentation* [en ligne]. Disponible sur : https://docs.gitlab.com/ [consulté le 31/08/2018]

[115] CHENET Carl, *Le danger Github (revu et augmenté)*, Site carlchenet.com, 2016. [en ligne] Disponible sur : https://carlchenet.com/le-danger-github-revu-et-augmente/ [consulté le 30/08/2018]

[116] Software Heritage [en ligne]. Disponible sur : https://www.softwareheritage.org/?lang=fr

Enfin, au vu de l'ampleur du projet initié par la Software Heritage, il aurait été intéressant que les plateformes de développement participent activement à l'alimentation efficace de l'Archive, en proposant par exemple des réplications de leurs bases de données, qu'il conviendrait de relier à l'Archive, facilitant la récupération des données. Un tel dispositif permettrait aussi de proposer aux utilisateurs d'alimenter l'infrastructure ainsi mise à disposition, en respectant des règles relatives aux métadonnées, à la description du code source, et à la bonne conformité de celui-ci. De cette manière, les dispositions mises en place pour l'archivage des logiciels[117] seraient le fruit d'une discussion entre les plateformes et l'initiative de l'INRIA. Du côté de la Software Heritage, l'adaptation du modèle de donnée serait plus simple et ne nécessiterait pas toujours l'utilisation d'un *loading* différent. Pour les plateformes de développement, ce serait aussi la possibilité de disposer d'une sauvegarde complète, qu ne serait pas localisée en interne. En cas de défaillance des serveurs ou de tout autre inconvénient grave rencontré, cela permettrait de pouvoir repartir d'un état stable par lequel la forge est passée. Une sorte de *versionning* de la plateforme en fait.

3.5 DES ÉVOLUTIONS DE L'ARCHIVE DE LA SWH

Les principes théoriques et techniques mis en place pour la SWH sont déjà très complexes et répondent à un ensemble de problématiques qui permettent d'archiver le code source des logiciels. Diverses évolutions pourront néanmoins enrichir les fonctionnalités de l'Archive, à terme.

3.5.1 La notification de liens morts

Le balayage qui consiste à identifier et récupérer les informations (via le principe du *listing* et *loading*) est capable de signifier à l'Archive qu'un fichier est déjà référencé. En revanche il n'est pas indiqué, dans le fonctionnement de ces modules[118], ce qu'il se passe lorsque la "sonde" balaye un projet ou une donnée déjà archivée dans l'infrastructure dont les données sont manquantes, et si un message vient notifier les administrateurs de l'Archive qu'un lien n'est plus actif.

[117] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

[118] DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

La possibilité de palier à cela serait donc de développer cette fonctionnalité qui alerterait les membres de la Software Heritage de l'absence de données (alors qu'il y aurait une correspondance côté Archive).

Pour que ces recherches soient d'autant plus efficace, il serait aussi intéressant développer une sonde dont la tâche principale serait justement de détecter les sites fermés ou les projets arrêtés. Un tel dispositif irait ainsi dans le même sens que les objectifs de la Software Heritage, c'est-à-dire que l'Archive soit capable d'identifier des liens morts et de rapatrier ainsi les données avant qu'elles ne soient totalement inaccessibles ou inexploitables.

3.5.2 Les évolutions prévues

Le projet de Software Heritage pourrait devenir la bibliothèque universelle de référence d'ici quelques années, ambition initiale des créateurs du projet.

Les objectifs et évolutions prévues pour aller dans ce sens peuvent être regroupés en trois axes principaux[119] :

- Elargissement de la couverture, qui a pour objectif d'atteindre un bien plus grand ensemble de forges et de sites hébergeurs, mais aussi encourager les démarches proactives en proposant aux utilisateurs de venir directement ajouter l'URL d'un code source

- Apporter un plus grand socle contextuel, notamment sur les informations de provenance, c'est-à-dire spécifier plus précisément les origines du code source récupéré et diffuser ces informations aux utilisateurs pour permettre d'exploiter ces informations dans le cadre d'applications

- Permettre la recherche en texte intégral, qui consistera en la constitution d'un index de l'Archive et la consultation en "full" texte du code. Cela permettra alors de faciliter la recherche au sein même des Archives qui pour le moment passe par une API qui n'expose les résultats d'une recherche que par le biais d'une liste, sans indiquer les numéros de page.

Ces améliorations constituent ainsi pour la Software Heritage la possibilité de prétendre légitimement au statut de bibliothèque universelle de codes source.

[119] Software Heritage [en ligne]. Disponible sur : https://www.softwareheritage.org/?lang=fr

3.6 CONCLUSION DE LA PARTIE 3

L'ensemble de ces recommandations met l'accent sur le lien étroit qui peut y avoir entre les plateformes de développement et la Software Heritage. Maintenant que cette dernière a ouvert ses portes et qu'elle espère pouvoir devenir la bibliothèque d'Alexandrie du code source, l'évolution de son environnement et des services qu'elle sera amenée à proposer dans le futur pourrait intéresser bien plus que les utilisateurs.

L'Archive de la Software Heritage a en effet déjà su montrer qu'elle dispose de technologies tout à fait intéressantes permettant de conserver le code source, mais également de s'adapter au besoin croissant en espace que nécessite le stockage d'une telle ressource.

CONCLUSION

Le logiciel, jusqu'au début des années 90, était l'apanage des éditeurs de solutions propriétaires. Les FLOSS ont tout de même su progressivement se faire une place et à instaurer une législation "libre". Celle-ci a fait ses preuves, et les valeurs véhiculées ont légitimé l'utilisation de logiciels Libre et/ou Open Source auprès d'une communauté de développeurs et d'utilisateurs répartie aux quatre coins du globe.

D'abord isolés par les restrictions des technologies et par la place proéminente des logiciels propriétaires au sein des systèmes, ils ont su développer des solutions alternatives, puissantes et fédératrices en s'organisant virtuellement avec des individus partageant les mêmes valeurs et les mêmes ambitions. De ces fructueuses collaborations ont découlé des projets majeurs (Linux, Debian, Git, …) mais surtout tout un ensemble de méthodes permettant la gestion du code source à toutes les étapes de son cycle de vie: de sa création à sa mise en production, en passant par la réalisation détaillée de la documentation. Ainsi, le logiciel libre arrive à proposer une maintenance et une solution évolutive.

L'avènement des plateformes de développement au début des années 2000 est une réponse à un besoin significatif chez les développeurs, de disposer d'un "couteau suisse" du développeur, leur permettant à la fois de travailler sereinement sur leurs projets, tout en maintenant la communication avec l'ensemble des membres du projet et les utilisateurs, qui sont plus que jamais des acteurs indirects du développement d'un logiciel. Les moyens technologiques grandissant, les projets font de même, nécessitant désormais la traçabilité complète du développement d'un logiciel, de la première ligne de code à la dernière éditée. Cela permet d'avoir une marge de manoeuvre dans les erreurs. Indirectement, l'emploi du *commit* et du *versionning* devient des actes significatifs et réguliers participant à la conservation sur le long terme du code source. Celui-ci étant sujet à moultes modifications au cours de son exploitation, il devient essentiel de pouvoir maintenir la stabilité de sa structure. C'est d'autant plus prégnant qu'avec les grands mouvements d'évolutions dont sont sujettes les plateformes de développement, les communautés doivent pouvoir migrer la totalité de leurs projets, en optimisant au maximum les processus "d'empaquetage", pour permettre le rapatriement complet des données relatives au code source traités dans le cadre du projet. Pour cela, le choix dans la plateforme de développement doit être le

résultat d'un processus de réflexion poussé sur les attentes par rapport aux objectifs fixés par les équipes.

Propriétaires ou Libres, les plateformes de développement ne peinent pas à trouver des utilisateurs, qui se pressent pour exploiter les ressources mises à leur dispositions et qui génèrent dans le même temps plusieurs milliards de données. La grande majorité de ces données sont d'ailleurs labellisées FLOSS et donc exploitables par tous (administrateurs des forges inclus). La notion de propriété ne s'appliquant pas dans le cadre du Libre et de l'Open Source, c'est un véritable catalogue du logiciel, ouvert à tous, qui se constitue perpétuellement, et dont la richesse intellectuelle et patrimoniale s'accroît grâce aux actions de valorisation et de diffusion. Un certain nombre d'entités œuvrent désormais activement pour la conservation et la pérennisation du code source et du logiciel, ainsi que de tous les éléments relatifs à sa logique conceptuelle. Que ce soit les Comprehensive "X" Archive Network, ou bien les projets Debian ou Linux, chacun met à disposition des utilisateurs un ensemble de ressources fiables et exploitables permettant ainsi de souligner la valeur ajoutée que peut représenter un potentiel "archivage" du logiciel.

En effet, les dispositions d'archivage "hybrides" proposées par les plateformes et les initiatives précédentes ne correspondent pas tout à fait à un archivage stricte du code source, ce dernier ne représentant que très peu d'intérêt à être archivé seul où sous son unique forme textuelle.

Le savoir invisible véhiculé par le code source (à savoir la réflexion logique ayant permise de mettre en place l'intelligibilité de celui-ci) est matérialisé sous différentes formes, en fonction des étapes par lesquelles le logiciel est amené à être développé. L'historique permettra par exemple de revenir à un état où la complexité structurelle du code n'empêche plus d'apercevoir les bases fondamentales de celui-ci. Les documentations permettant l'exploitation (l'utilisation et le développement) du logiciel permettront de mieux identifier les compatibilités existantes avec des systèmes d'exploitations différents, garantissant ainsi la possibilité de faire fonctionner le logiciel dans le futur.

Tout ceci n'est possible bien sûr que si le logiciel est conservé dans des conditions optimales, et que sa licence permette l'exploitation libre et gratuite de son code. La Software Heritage, projet extrêmement intéressant et ayant été

officiellement lancé pendant la rédaction de ce mémoire, propose en ce sens de créer la plus grande bibliothèque du code source n'ayant jamais existée. Cette initiative ambitieuse a pour objectif principal, sur le long terme, d'être capable de continuer à exploiter et préserver l'ensemble du patrimoine numérique pour lequel le logiciel est indispensable. L'objectif donc est de mettre à la disposition des chercheurs et des utilisateurs de tous les secteurs d'activités, un prototype de la bibliothèque d'Alexandrie au XXIe siècle. Pour se faire, il aura fallu se soustraire à un certains nombre de contraintes théoriques et techniques liées directement à l'archivage du code source. Les différentes études en ce sens ont permis de développer un ensemble de solutions technologiques résolvant les problématiques d'alimentation de l'Archive, de conservation et d'identification des ressources ainsi récupérées, et de proposer un environnement stable de consultation pour les utilisateurs.

Bien évidemment, il reste encore beaucoup à faire : étendre le périmètre de balayage, indexer la totalité des données, et enrichir les informations de localisation d'un projet logiciel. Tout ceci en prenant en compte les structurations différentes des plateformes n'exploitant pas les données avec les mêmes logiciels de *versionning* ou ne référençant pas les projets. A l'avenir, il faudra également que l'ensemble des préconisations à mettre en place soit d'une part, appliqué par la globalité des communautés, mais aussi que ces recommandations fassent l'objet permanent d'études permettant de faire évoluer les solutions, parallèlement au logiciel. Celui-ci lui devrait désormais connaître une évolution croissante supérieure à ce qu'il connaît aujourd'hui, grâce aux infrastructures de conservation que nous lui dédions au siècle d'aujourd'hui.

SOURCES

ArchiveTeam, *gitorious valhalla*, Site gitorious.org [en ligne] Disponible sur : https://gitorious.org/ [consulté le 31/08/2018]

Atlassian. *Git* [en ligne]. Disponible sur : https://www.atlassian.com/git [consulté le 31/08/2018]

dzecniv, *Gitlab achète Gitorious,* Site linuxfr.org, 2015. [en ligne] Disponible sur: https://linuxfr.org/news/gitlab-achete-gitorious [consulté le 31/08/2018]

Free Software Foundation, *Évaluation des hébergeurs de logiciel selon les critères éthiques de GNU*, Site gnu.org [en ligne] Disponible sur: https://www.gnu.org/software/repo-criteria-evaluation.fr.html [consulté le 31/08/2018]

Free Software Foundation, *Philosophie du projet GNU*, Site gnu.org [en ligne] Disponible sur: http://www.gnu.org/philosophy/philosophy.html [consulté le 31/08/2018]

Free Software Foundation, *Licences*, Site gnu.org [en ligne] Disponible sur: http://www.gnu.org/licenses/licenses.html [consulté le 31/08/2018]

Free Software Foundation, *Savannah*, Site savannah.gnu.org [en ligne] Disponible sur: https://savannah.gnu.org/ [consulté le 31/08/2018]

GitHub, *Documentation* [en ligne]. Disponible sur : https://github.com/features#documentation [consulté le 31/08/2018]

GitLab. *Documentation* [en ligne]. Disponible sur : https://docs.gitlab.com/ [consulté le 31/08/2018]

GitHub. *The largest open source community in the world* [en ligne]. Disponible sur : https://github.com/open-source [consulté le 31/08/2018]

GitHub. *Blog* [en ligne]. Disponible sur : https://blog.github.com/ [consulté le 31/08/2018]

GitLab, *GitLab acquires Gitorious to bolster its on premises code collaboration platform*, Site about.gitlab.com, 2015. [en ligne] https://about.gitlab.com/2015/03/03/gitlab-acquires-gitorious/ [consulté le 31/08/2018]

INRIA, *Robert Di Cosmo, acteur du libre et porteur du projet Software Heritage, Portrait,* Site inria-alumni.fr, 2017. [en ligne] http://www.inria-alumni.fr/roberto-di-cosmo-software-heritage/ [consulté le 31/08/2018]

MERKLE Ralph C., *Method of providing digital signatures*, Brevet US4309569A, 1982. [en ligne] Disponible sur : https://patents.google.com/patent/US4309569 [consulté le 31/08/2018]

Ministère de l'économie, de l'industrie et du numérique, *Guide sur le Cloud Computing à l'attention des collectivités locales,* 2015. [en ligne] Disponible sur : https://www.entreprises.gouv.fr/files/files/directions_services/secteurs-professionnels/numerique/guide-cloud-computing-et-datacenters-2015.pdf [consulté le 31/08/2018]

PLUME, *Promouvoir les Logiciels Utiles Maîtrisés et Économiques dans l'enseignement supérieur et la recherche*, Site projet-plume.org [en ligne] Disponible sur: https://projet-plume.org/ [consulté le 31/08/2018]

PLUME, *Mercurail Hg : Gestionnaire de version décentralisé*, Site projet-plume.org [en ligne] Disponible sur : https://projet-plume.org/fiche/mercurial-hg [consulté le 31/08/2018]

Software in the Public Interest et autres, *debian, The universal operating system*, Site debian.org, 2018. [en ligne] Disponible sur: https://www.debian.org/index.fr.html [consulté le 31/08/2018]

BIBLIOGRAPHIE

Ouvrages

BAR Moshe, FOGEL Karl, *Open Source Development with CVS*, 3rd Edition. Arizona : Paraglyph Press, 2003. 368 p.

BLONDEEL Sébastien, CARTRON Daniel, RISI Juliette, THOMAS Jean-Marie, *Débuter sous Linux avec Mandriva,* Editions Eyrolles, 2011. 520p.

CHACON Scott, STRAUB Ben, *Pro Git : Everything you need to know about Git*, Apress, 2018, 517 p.

FutuRIS, *La recherche et l'innovation en France.* Paris : Odile Jacob, 2016. 456 p.

PERRIAULT Jacques, VAGUER Céline, *La norme numérique : savoir en ligne et internet.* Paris : CNRS édition, 2011. 264 p.

RAYMOND Eric S., *The Cathedral and the Bazar*. États-Unis : édition « Libre », 1997. 241p. [en ligne] Disponible sur : http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ [consulté le 30/08/2018]

STALLMAN, Richard M., *Free Software, Free Society : Selected Essays of Richard M. Stallman.* Boston : Free Software Foundation, 2015. 305 p. [en ligne] Disponible sur : https://www.gnu.org/doc/fsfs3-hardcover.pdf [consulté le 30/08/2018]

Articles scientifiques

BECKER Georg, *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis,* Ruhr-Universität Bochum, 2018. [en ligne] Disponible sur : http://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker_1.pdf [consulté le 31/08/2018]

CORIS Marie, LUNG Yannick, « Les communautés virtuelles : la coordination sans proximité ? Les fondements de la coopération au sein des communautés du logiciel libre », *Revue d'Économie Régionale & Urbaine*, 2005/3 (juillet), p. 397-420. DOI : 10.3917/reru.053.0397. [en ligne] Disponible sur https://www.cairn.info/revue-d-economie-regionale-et-urbaine-2005-3-page-397.htm [consulté le 30/08/2018]

COUTURE Stéphane, « L'écriture collective du code source informatique. Le cas du commit comme acte d'écriture », *Revue d'anthropologie des connaissances*, 2012/1 (Vol. 6, n° 1), p. 21-42. DOI : 10.3917/rac.015.0061. [en ligne] Disponible sur: https://www.cairn.info/revue-anthropologie-des-connaissances-2012-1-page-21.htm [consulté le 30/08/2018]

DI COSMO Roberto, ZACCHIROLI Stefano, *Software Heritage : Why and How to Preserve Software Source Code*, iPRES 2017 – 14th International Conference on Digital Preservation, Kyoto, Japan, 2017. 11p. [en ligne] Disponible sur https://hal.archives-ouvertes.fr/hal-01590958 [consulté le 30/08/2018]

DI COSMO Roberto, « Organiser le partage pour préserver les données », *Pour la science,* 2012/000. [en ligne] Disponible sur: http://www.dicosmo.org/Media/2013-11-PourLaScience-partage_donnees.pdf [consulté le 30/08/2018]

DI COSMO Roberto, « La science du logiciel libre », *Les Cahiers de l'INRIA*, La recherche, 2009/436 [en ligne] Disponible sur : http://www.dicosmo.org/Media/LaRecherche-2009-12.pdf [consulté le 30/08/2018]

DI COSMO Roberto, propos recueillis par David Larousserie, « Offrons aux jeunes des clés du pouvoir et de la liberté », *Sciences & Avenir,* 2009. [en ligne] Disponible sur : http://www.dicosmo.org/Media/2009-09-SciencesEtAvenir.pdf [consulté le 30/08/2018]

DI COSMO Roberto, *What would you do with billions of source code files ? Challenges and opportunities in software archival*, EPFL Workshop du 7 février, INRIA, 2017. [en ligne] Disponible sur : https://project.inria.fr/epfl-Inria/files/2017/02/RobertoDiCosmo-talk.pdf [consulté le 30/08/2018]

ILLIEN Gildas, « Une histoire politique de l'archivage du web ». *Bulletin des bibliothèques de France (BBF)*, 2011, n° 2, p. 60-68. [en ligne] Disponible sur: http://bbf.enssib.fr/consulter/bbf-2011-02-0060-012 [consulté le 30/08/2018]

MANCINELLI Fabio, BOENDER Jaap, DI COSMO Roberto, VOUILLON Jérôme, DURAK Berke, LEROY Xavier, TREINEN Ralf, *Managing the Complexity of Large Free and Open Source Package-Based Softxare Distributions*, 21st IEEE/ACM International Conference on Automated Software Engineering, Septembre 2006, Tokyo, Japan, 2006. [en ligne] Disponible sur : https://hal.archives-ouvertes.fr/hal-00149566/file/ase.pdf [consulté le 30/08/2018]

MATTHEWS Brian, SHAON Arif, BICARREGUI Juan, JONES Catherine, *A framework for Software Preservation,* The International Journal of Digital Curation, e-Science Centre, Science and Thechnology Facilities Council, Rutherford Appleton Laboratory, Oxon, UK, 2010. [en ligne] Disponible sur : http://www.ijdc.net/article/view/148/210 [consulté le 31/08/2018]

STALLMAN Richard M., *Le logiciel libre est encore plus essentiel maintenant*, 2017. [en ligne] Disponible sur: https://www.gnu.org/philosophy/free-software-even-more-important.fr.html [consulté le 30/08/2018]

STALLMAN Richard M., *En quoi l'open source perd de vue l'éthique du logiciel libre ?*, 2016. [en ligne] Disponible sur : https://www.gnu.org/philosophy/open-source-misses-the-point.fr.html [consulté le 30/08/2018]

WILLAIME Pierre, HOCQUET Alexandre, « Wikipédia au prisme de l'épistémologie sociale et des études des sciences », *Cahiers philosophiques*, 2015/2 (n° 141), p. 68-86. DOI : 10.3917/caph.141.0068. [en ligne] Disponible sur: https://www.cairn.info/revue-cahiers-philosophiques1-2015-2-page-68.htm [consulté le 30/08/2018]

Ressources web

Agence Pour la Protection des Programmes, *Contrat de licence d'utilisation : logiciel propriétaire.* [en ligne] Disponible sur: **https://www.app.asso.fr/centre-information/base-de-connaissances/code-logiciels/les-contrats/contrat-de-licence-dutilisation-logiciel-proprietaire**

Bastien L, *Snapshot de stockage : qu'est-ce qu'un instantané de stockage ? A quoi ça sert ?,* Site Lebigdata.fr, 2018. [en ligne] Disponible sur : https://www.lebigdata.fr/snapshot-stockage-definition [consulté le 30/08/2018]

"Bethesda interdit la revente d'un jeu d'occasion et menace de poursuites", Site journaldugeek.com [en ligne]. Disponible sur https://www.journaldugeek.com/2018/08/13/bethesda-interdit-revente-dun-jeu-doccasion-menace-de-poursuites/

calimaq, *Les logiciels produits par les administrations sont passés en Open Source par défaut (et voici pourquoi),* Site scinfolex.com, S.I.Lex, 2018. [en ligne] Disponible sur : https://scinfolex.com/2017/12/08/les-logiciels-produits-par-les-administrations-sont-passes-en-open-source-par-defaut-et-voici-pourquoi/amp/?__twitter_impression=true [consulté le 30/08/2018]

Centraphone, *L'ordinateur,* Site centraphone.fr [en ligne] Disponible sur : http://www.centraphone.fr/ordinateur.htm [consulté le 30/08/2018]

CHENET Carl, *Le danger Github (revu et augmenté)*, Site carlchenet.com, 2016. [en ligne] Disponible sur : https://carlchenet.com/le-danger-github-revu-et-augmente/ [consulté le 30/08/2018]

CLAUDEL Maxime, *Tesla publie une partie du code source des technologies équipant ses voitures,* Site Numerama.com, 2018. [en ligne] Disponible sur : https://www.numerama.com/tech/376834-tesla-publie-une-partie-du-code-source-des-technologies-equipant-ses-voitures.html [consulté le 31/08/2018]

Code Source, Site wikipedia.org [en ligne] Disponible sur : **https://fr.wikipedia.org/wiki/Code_source**

Collectif, *Forges logicielles et hébergement de projets libres*, Site Linuxfr.org, 2018. [en ligne] https://linuxfr.org/news/forges-logicielles-et-hebergement-de-projets-libres [consulté le 31/08/2018]

FALQUE Jean-Claude, *ZUSE Conrad (1910-1995)*, Site universalis.fr [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/konrad-zuse/ [consulté le 31/08/2018]

Futura, *Loi de Moore*, Site futura-sciences.com [en ligne] Disponible sur: https://www.futura-sciences.com/tech/definitions/informatique-loi-moore-2447/ [consulté le 31/08/2018]

Futura, *Augusta Ada Lovelace*, Site futura-sciences.com [en ligne] Disponible sur : https://www.futura-sciences.com/sciences/personnalites/mathematiques-augusta-ada-lovelace-869/ [consulté le 31/08/2018]

GARIMELLA Neeta, *Understanding and exploiting snapshot technology for data protection, Part 1, Snapshot technology overview, Helps you make informed decisions about implemnting snapshots*, Site ibm.com, 2006. [en ligne] Disponible sur : https://www.ibm.com/developerworks/tivoli/library/t-snaptsm1/index.html [consulté le 30/08/2018]

Google Open Source Code, "*Biding farewell to Google Code"* [en ligne]. Disponible sur : https://opensource.googleblog.com/2015/03/farewell-to-google-code.html

International Standardization for Organization (ISO), *WARC new iso file format to store billions of online data,* Site iso.org, 2009. [en ligne] Disponible sur : https://www.iso.org/news/2009/10/Ref1255.html [consulté le 30/08/2018]

KATZ Daniel S., *Software Heritage and repository metadata : a software citation solution,* Site danielskatzblog.worpress, 2017. [en ligne] Disponible sur :

https://danielskatzblog.wordpress.com/2017/09/25/software-heritage-and-repository-metadata-a-software-citation-solution/ [consulté le 31/08/2018]

LA REDACTION, *Le numérique est-il la réponse à tout nos problèmes ?,* Site Convergence, archivistesqc.wordpress.com, 2018. [en ligne] Disponible sur : https://archivistesqc.wordpress.com/2018/02/19/numerique-2/ [consulté le 30/08/2018]

LAVOISY Oliver, *E.N.I.A.C*, Site universalis.fr [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/e-n-i-a-c/ [consulté le 31/08/2018]

L'express, League of Legends : le jeu multijoueur le plus joué au monde [en ligne] Disponible sur : **https://www.lexpress.fr/culture/jeux-video/league-of-legends-le-jeu-multijoueur-le-plus-joue-au-monde_1721040.html**

Lycée Saint-Charles Borromée, *Le système binaire*, Site info-isn.fr [en ligne] Disponible sur : http://www.info-isn.fr/systeme-binaire.pdf [consulté le 31/08/2018]

MERKLE, Ralph C., *Ralph C. Merkle*, Site merkle.com [en ligne] Disponible sur : http://www.merkle.com/ [consulté le 31/08/2018]

MEYER Jacques, *NEPER ou NAPIER JOHN – (1550-1617)*, Site Universalis.fr, 2018. [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/neper-napier/ [consulté le 30/08/2018]

MOUYSSINAT Michel, *La machine de Schickard*, Site irem.univ-reunion.fr [en ligne] Disponible sur : http://irem.univ-reunion.fr/homocalculus/Data/menu/visite/visite/theme2/r_machine_shickard.htm [consulté le 31/08/2018]

MOUYSSINAT Michel, *La machine de Pascal*, Site irem.univ-reunion.fr [en ligne] Disponible sur : http://irem.univ-reunion.fr/homocalculus/Data/menu/visite/visite/theme2/r_machine_pascal.htm [consulté le 31/08/2018]

MOUYSSINAT Michel, *Leibniz*, Site irem.univ-reunion.fr [en ligne] Disponible sur : http://irem.univ-reunion.fr/homocalculus/Data/menu/visite/visite/theme2/r_leibnitz.htm [consulté le 31/08/2018]

National Archives. *Bulletin 2015-04,* Site archives.gov, 2015. [en ligne] Disponible sur : https://www.archives.gov/records-mgmt/bulletins/2015/2015-04.html [consulté le 30/08/2018]

National Archives. *NARA Bulletin 2010-05,* Site archives.gov, 2010. [en ligne] Disponible sur : https://www.archives.gov/records-mgmt/bulletins/2010/2010-05.html [consulté le 30/08/2018]

Nombre d'abonnés de World of Warcraft, Site millenium.org[en ligne] Disponible sur : https://www.millenium.org/news/176155.html

PIRE Bernard, *BABBAGE Charles (1792-1871)*, Site universalis.fr [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/charles-babbage/ [consulté le 31/08/2018]

PIRE Bernard, *AIKEN HOWARD HATHAWAY (1900-1973)*, Site universalis.fr [en ligne] Disponible sur : http://www.universalis.fr/encyclopedie/howard-hathaway-aiken/ [consulté le 31/08/2018]

Plan national pour la science ouverte [en ligne], Disponible sur : **http://m.enseignementsup-recherche.gouv.fr/cid132529/le-plan-national-pour-la-science-ouverte-les-resultats-de-la-recherche-scientifique-ouverts-a-tous-sans-entrave-sans-delai-sans-paiement.html**

POISARD C., *Fiche 3 : L'étude du boulier chinois*, Site culturemath.ens.fr, 2006. [en ligne] Disponible sur : http://culturemath.ens.fr/nodeimages/images/fiche3.pdf [consulté le 30/08/2018]

POISARD C., *Fiche 5 : L'étude de la règle de calcul*, Site culturemath.ens.fr, 2006. [en ligne] Disponible sur : http://culturemath.ens.fr/materiaux/poissard/fiche5.pdf [consulté le 30/08/2018]

"Rachat de GitHub : Pourquoi ce rachat et quels sont les plans de Microsoft ?" [en ligne] Disponible sur : **https://www.developpez.com/actu/207642/Rachat-de-GitHub-Pourquoi-ce-rachat-et-quels-sont-les-plans-de-Microsoft/**

ROCHEREUIL Chloé, *R.I.P magnétoscope ! La production de lecteurs VHS va être stoppée*, Site mashable.france24.com, 2016. [en ligne] Disponible sur : http://mashable.france24.com/tech-business/20160721-arret-production-magnetoscope-vhs [consulté le 31/08/2018]

ROUSE Margaret, *Snapshot (Stockage)*, Site lemagit.fr, 2014. [en ligne] Disponible sur : https://www.lemagit.fr/definition/Snapshot-Stockage [consulté le 30/08/2018]

Software Heritage, *Persistent identifiers,* Site docs.softwareheritage.org [en ligne] Disponible sur : https://docs.softwareheritage.org/devel/swh-model/persistent-identifiers.html [consulté le 31/08/2018]

Vikidia, *Système décimal*, Site fr.vikidia.org [en ligne] Disponible sur: https://fr.vikidia.org/wiki/Syst%C3%A8me_d%C3%A9cimal [consulté le 30/08/2018]

Wikipédia, Assassin Creed (film) , Site wikipedia.org [en ligne] Disponible sur : **https://fr.wikipedia.org/wiki/Assassin%27s_Creed_(film**)

Wikipédia, *Calculateur analogique,* Site wikipedia.org [en ligne] Disponible sur : https://fr.wikipedia.org/wiki/Calculateur_analogique [consulté le 30/08/2018]

Wikipédia, *Fonction de hachage,* Site wikipedia.org [en ligne] Disponible sur : https://fr.wikipedia.org/wiki/Fonction_de_hachage [consulté le 30/08/2018]

Wikipédia, *Electronic Discrete Variable Automatic Computer*, Site wikipedia.org [en ligne] Disponible sur: https://fr.wikipedia.org/wiki/Electronic_Discrete_Variable_Automatic_Computer [consulté le 31/08/2018]

ZAWINSKI Jamie, *resignation and postmortem.,* Site jwz.org, 1999 .[en ligne] Disponible sur : https://www.jwz.org/gruntle/nomo.html [consulté le 30/08/2018]

Normes

ISO, 2009, *Information and documentation – WARC file format*, 28500:2009, 26 p.

Association Française de Normalisation (AFNOR), 2009. *Archivage électronique : spécifications relatives à la conception et à l'exploration de systèmes informatiques en vue d'assurer la conservation et l'intégrité des documents stockés dans ces systèmes.* NF Z42-013, 44 pages.

GLOSSAIRE

Code source : ensemble d'instructions écrit dans un langage de programmation lisible par un humain et qui, une fois interprété, compilé ou assemblé, devient un code objet qui peut être exécuté par l'ordinateur[120].

Commit : acte informatique qui consiste à valider formellement une modification du code source d'un logiciel[121].

Communauté du Libre : Communauté de personnes en faveur du logiciel libre, c'est à dire prônant la mise à disposition du code sources des logiciels pour sa réutilisation, son exploitation et son amélioration par tout un chacun. La philosophie marquant la communauté du Libre est à mettre en opposition avec la vision du logiciel propriétaire.

Copyleft : en opposition avec le Copyright, « gauche d'auteur » désigne, pour une œuvre ou création soumise à la propriété intellectuelle, une licence impliquant la conservation du droit d'auteur, inaliénable, tout en permettant une réutilisation, modification et diffusion de l'objet soumis à celui-ci.

Fork : scission volontaire d'un projet de développement en deux parties concernant le même sujet, et évoluant parallèlement. Cela indique une divergence de points de vue au sein de l'équipe projet, qui peut être néfaste pour l'aboutissement de celui-ci par la suite.

Logiciel : ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitement de données[122].

Logiciel propriétaire : logiciel dont le code source n'est pas rendu accessible, et dont l'exploitation est exclusivement réservée sous forme de licence, à des fins commerciales.

Maintener : administrateur en charge de garantir un code informatique sain, notamment en mettant en place une gestion des droits et des habilitations, en organisant des méthodes de contrôle, et maîtriser les blocs de données susceptibles d'être ajoutés au code.

[120] Code Source, Site wikipedia.org [en ligne] Disponible sur : https://fr.wikipedia.org/wiki/Code_source

[121] COUTURE Stéphane, « L'écriture collective du code source informatique. Le cas du commit comme acte d'écriture », *Revue d'anthropologie des connaissances*, 2012/1 (Vol. 6, n° 1), p. 21-42. DOI : 10.3917/rac.015.0061. [en ligne] Disponible sur: https://www.cairn.info/revue-anthropologie-des-connaissances-2012-1-page-21.htm [consulté le 30/08/2018]

[122] Larousse, le logiciel, site arousse.fr [en ligne] Disponible sur : http://www.larousse.fr/encyclopedie/divers/logiciel/66458

Mirror : réplication des données d'un serveur

Mouvement Open Source : issu de la communauté du Libre, ce mouvement a pour objectif de promouvoir l'ouverture et la diffusion des codes sources dans l'optique d'arriver à une émulation profitable au logiciel lui-même, afin d'en faire profiter un maximum d'utilisateurs, et d'obtenir des logiciels toujours plus puissants et fiables.

Package : regroupement dans un « paquet » du code source et des aspects contextuels utiles à son exploitation, par exemple la documentation du logiciel.

Plateforme de développement : logiciel géant permettant l'accès, via Internet, à des projets de développement logiciel et à leur gestion. Elles permettent généralement le travail collaboratif.

Programmation orientée objet (POO) : méthode de programmation consistant à réaliser des blocs de code, ou « briques logicielles » interagissant entre elles, au lieu d'un seul et même fil directeur, permettant ainsi d'apporter de la souplesse au développement logiciel.

Science ouverte / Open Science : principe de mise à disposition gratuite et libre des recherches scientifiques qui permet à tout un chacun de consulter et d'exploiter ce qui a été fait par les autres. Ce mouvement incite les chercheurs à mettre à disposition leurs travaux de manière gratuite pour favoriser l'exploitation et la capitalisation des connaissances, dans une optique d'amélioration et de partage.

Snapshot : instantané de l'état d'un système à un moment donné

Software Heritage : projet de mise à disposition des codes sources libres créé le 30 juin 2016, dans l'optique de les rendre accessibles à tous via Internet. Cette initiative a pour vocation de préserver le patrimoine du logiciel et de le valoriser, en permettant une meilleure diffusion et de nouvelles exploitations futures de ces codes.

Tiers-hébergement : consiste à proposer des services d'hébergement, de logiciels ou de données, qui ne seront donc pas présents sur l'infrastructure du déposant mais sur celle du tiers-hébergeur. En d'autres termes, l'hébergement est réalisé par un tiers.

Travail collaboratif : consiste à partager une base de travail (document texte, code, etc.) pour que plusieurs collaborateurs aient simultanément ou en différé l'accès à celui-ci, pour y apporter des modifications et des améliorations, tout en ayant la vision sur les ajouts des autres. La gestion du travail collaboratif peut se faire au moyen d'outils dédiés, permettant notamment de tracer les apports de chaque individus ainsi que les accès alloués.

Trunk/branch : Système permettant de rajouter une couche de contrôle avant d'effectuer des modifications importantes au corps du code. Le trunk, tronc en anglais, correspond à l'entité principale qui sert de référence. Des branches peuvent être créées à partir de ce tronc pour effectuer des modifications sans l'impacter lui-même, puis celles-ci peuvent être à nouveau « mergées » au tronc une fois qu'il est jugé utile de le faire.

Versionning : acte consistant à incrémenter les différentes versions d'un fichier à mesure que l'on y apporte des améliorations. Au lieu d'écraser ce fichier avec la nouvelle sauvegarde, on garde ainsi une trace des différentes étapes de construction de la réflexion, comme autant de fichiers, et on est également en mesure de retourner sur une version antérieure en cas de problème important sur la plus récente.

TABLE DES MATIÈRES