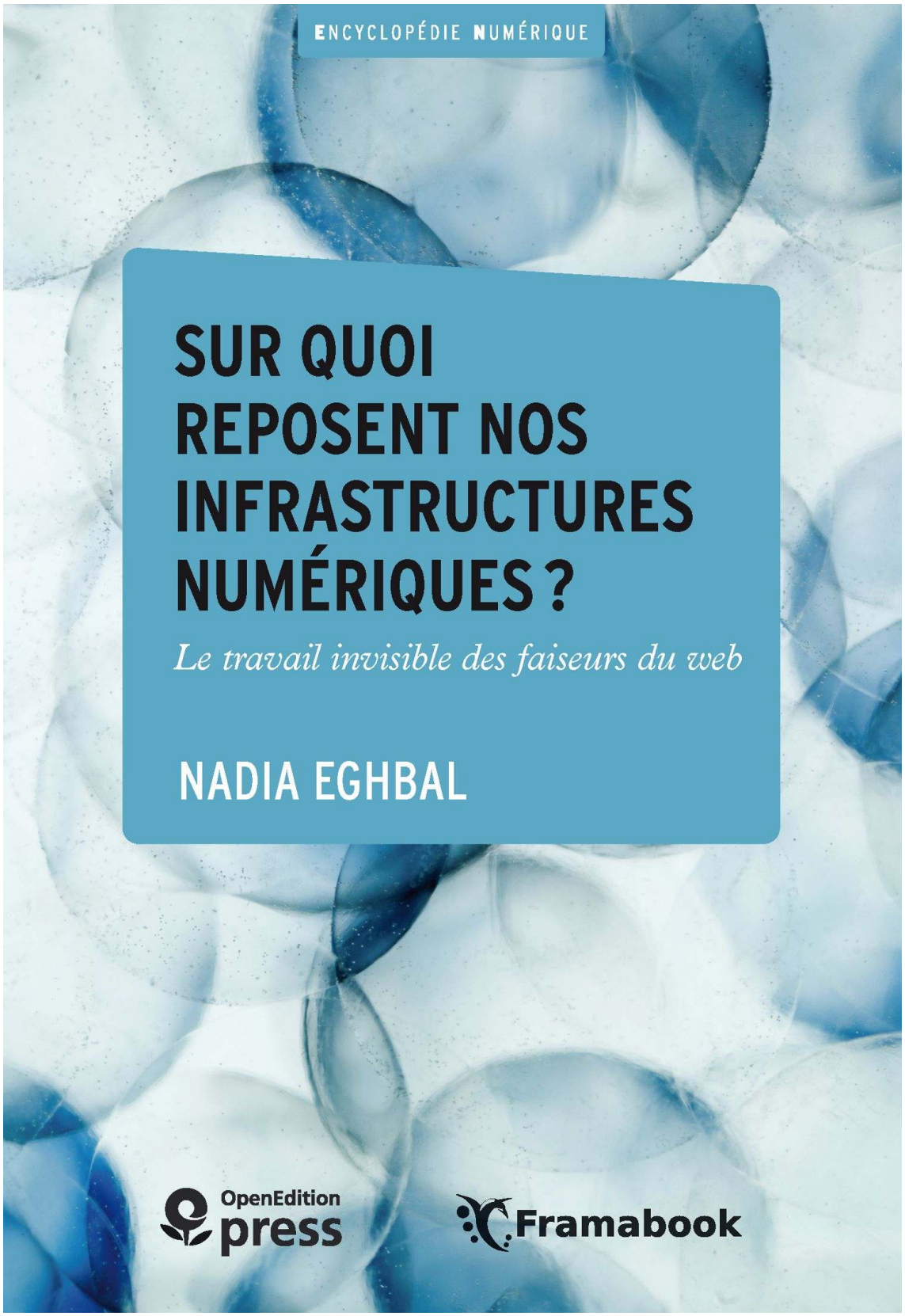
ENCYCLOPÉDIE NUMÉRIQUE
SUR QUOI REPOSENT NOS INFRASTRUCTURES NUMÉRIQUES ?
Le travail invisible des faiseurs du web
NADIA EGHBAL
OpenEdition press
Framabook

Sur quoi reposent nos infrastructures numériques ?

Le travail invisible des faiseurs du web

Nadia Eghbal

DOI : 10.4000/books.oep.1797
Éditeur : OpenEdition Press, Framabook
Année d'édition : 2017
Collection : Encyclopédie numérique
ISBN électronique : 9782821894938

Édition imprimée
ISBN : 9782821894914
Nombre de pages : 172

http://books.openedition.org

Référence électronique
EGHBAL, Nadia. *Sur quoi reposent nos infrastructures numériques ? Le travail invisible des faiseurs du web.* Nouvelle édition [en ligne]. Marseille : OpenEdition Press, 2017 (généré le 24 octobre 2017). Disponible sur Internet : <http://books.openedition.org/oep/1797>. ISBN : 9782821894938. DOI : 10.4000/books.oep.1797.

SUR QUOI REPOSENT NOS **INFRASTRUCTURES NUMÉRIQUES ?**

Le travail invisible des faiseurs du web

NADIA EGHBAL

SUR QUOI REPOSENT NOS **INFRASTRUCTURES NUMÉRIQUES ?**

Le travail invisible des faiseurs du web

OpenEdition press

Framabook

En couverture : ©shotsstudio – stock.adobe.com
Conception graphique : Veronica Holguín – Collectif Surletoit

Traduction : Collectif Framalang (voir présentation du projet page 167)
Correction : Solenne Louis

Nadia Eghbal, *Sur quoi reposent nos infrastructures numériques ?*
Collection « Encyclopédie numérique », 2017
[En ligne] http://books.openedition.org/oep/1797

La version originale de cet ouvrage a reçu le soutien de la Fondation Ford.

ISBN papier : 978-2-8218-9491-4
ISBN électronique : 978-2-8218-9493-8

AVANT-PROPOS

Le problème exposé dans cet ouvrage m'est apparu sur une intuition. Pour avoir travaillé dans des *startups* puis dans des sociétés de capital-risque, j'ai pu constater que des sommes d'argent considérables affluaient dans les entreprises de logiciel. Par ailleurs, en tant que développeuse logiciel amateur, j'étais bien consciente que je n'aurais rien pu produire toute seule. J'utilisais du code gratuit et public (plus connu sous le nom de *code open source*) dont j'assemblais des éléments afin de répondre à des objectifs personnels ou commerciaux. Et franchement, les personnes impliquées dans ces projets avaient, quel que soit leur rôle, fait le plus gros du travail.

Cette observation m'a trotté dans la tête pendant plusieurs années, tandis que j'assistais à l'explosion à droite et à gauche des *bootcamps*[1] où étaient diplômés de nouveaux développeurs de logiciels et que je voyais des *startups* lever plusieurs dizaines de millions de dollars pour vendre des produits qui tournaient sans doute avec plus de code libre que de code propriétaire. Comme j'avais travaillé auparavant dans des associations à but non lucratif, les biens publics et les défis qui leur sont associés me vinrent immédiatement à l'esprit. Pourtant, ce vocabulaire était étrangement absent du langage de mes pairs dans le monde du logiciel.

Après avoir quitté l'an dernier mon travail dans une entreprise de capital-risque, je me suis mis en tête d'étudier ce paradoxe auquel je ne cessais de penser : il existe de précieux logiciels qui ne peuvent pas s'appuyer sur des modèles commerciaux et qui manquent d'aides des pouvoirs publics, quelles qu'elles soient.

C'est plutôt amusant, mais le code *open source* ne figurait pas sur ma liste initiale. Comme mes collègues, j'avais supposé, à tort, que c'était l'exemple même de ressources logicielles à la

1. Un *bootcamp* est une formation accélérée au métier de développeur informatique. À l'origine, le terme désignait les camps d'entraînement des Marines américains.

disposition du public qui bénéficiaient d'un fort soutien. Lorsque j'ai mentionné l'*open source* à mes amis et mentors, ils m'ont aimablement dissuadée de poursuivre mes recherches dans ce domaine, puis incitée à trouver plutôt d'autres exemples de domaines qui avaient vraiment besoin d'aide.

Pourtant, je suis tombée sur un certain nombre de projets *open source* qui mettaient à mal ces préjugés. Il s'est avéré que maintenir les projets dans la durée était un problème connu dans le monde des contributeurs de l'*open source*. Plus je creusais la question et plus je découvrais des billets de blog, des articles et des forums de discussion qui abordaient la tension et l'épuisement éprouvés par ceux qui maintiennent les projets *open source*. Tous mes contacts m'indiquaient une autre personne à joindre et, sans m'en apercevoir, j'ai récolté un nombre incroyable de témoignages à ce sujet.

Je me suis rendu compte que j'avais découvert un problème certes « bien connu » des producteurs (les contributeurs de l'*open source*), mais dont les consommateurs (les entreprises de logiciels et les autres utilisateurs de code *open source*) n'avaient apparemment aucune idée. Cette anomalie m'a incitée à me pencher sur la question.

Par ailleurs, il semble que le milieu de l'*open source* soit lui-même en train d'évoluer, voire de bifurquer. J'ai eu des conversations très diverses avec des interlocuteurs multigénérationnels, tous contributeurs *open source*. Ils semblaient avoir des philosophies et des valeurs divergentes, au point de donner l'impression de ne pas utiliser le même vocabulaire. J'ai appris que dans les trois à cinq dernières années, la production ainsi que la demande dans le monde de l'*open source* avaient explosé grâce à l'amélioration des outils pour les développeurs et à celle de l'organisation du travail. Les contributeurs de l'*open source* d'aujourd'hui sont très différents de ceux d'il y a dix ans, sans parler de ceux d'il y a trente ans. Or ces générations ne communiquent pas entre elles, ce qui rend difficile toute conversation productive sur la maintenance pérenne des logiciels.

Au hasard d'une conversation avec Ethan Zuckerman, du MIT[2] Center for Civic Media[3], j'ai eu l'occasion de partager plus largement mes découvertes.

Bien que ne sachant pas exactement ce qu'il y avait derrière ni si j'employais le vocabulaire adéquat, j'ai décrit à Ethan le problème dont je m'étais rendu compte et il a eu la gentillesse de me mettre en contact avec Jenny Toomey de la Fondation Ford. Jenny m'a suggéré de rassembler les résultats de mes recherches dans un rapport. Au fur et à mesure de sa rédaction a émergé cet ouvrage sur notre société numérique moderne, et sur l'infrastructure cachée qui la sous-tend.

Le présent livre n'aurait jamais vu le jour si Ethan et Jenny n'avaient pas donné sa chance à une idée tout juste ébauchée qui désormais, grâce au travail d'écriture, s'est transformée en quelque chose de construit. Je les remercie chaleureusement d'avoir fait confiance à leur intuition. Je suis aussi reconnaissante envers Michael Brennan et Lori McGlinchey pour leurs conseils, leur regard, et leur enthousiasme au cours de la relecture. Enfin, et c'est sans doute le plus important, j'ai une dette envers toutes les personnes qui travaillent dans l'*open source* et qui ont rendu leur histoire publique pour que des gens comme moi puissent la lire – et particulièrement ceux qui ont pris de leur temps malgré un agenda chargé pour me divertir au détour d'une conversation ou d'un courriel. Cet ouvrage est un concentré de leur sagesse et non de la mienne. Je suis aussi particulièrement reconnaissante envers Russel Keith-Magee, Eric Holscher, Jan Lehnardt, Audrey Petrov et Mikeal Rogers pour les conversations que j'ai pu avoir avec eux. Ils continuent à m'inspirer par leur patience et leur dévouement à l'égard du travail *open source*.

Merci d'avoir été aussi attentionnés.

2. Le Massachusetts Institute of Technology (MIT), en français Institut de technologie du Massachusetts, est un institut de recherche et une université américaine, spécialisée dans les domaines de la science et de la technologie. Située à Cambridge, dans le Massachusetts, à proximité immédiate de Boston, au nord-est des États-Unis, elle est souvent considérée au XXIe siècle comme une des meilleures universités mondiales en sciences et en technologie (source Wikipédia).
3. Le Center for Civic Media est un laboratoire du MIT, « travaillant main dans la main avec différentes communautés pour créer, concevoir, déployer et évaluer collaborativement des outils et des pratiques de communication » (voir la page « À propos » sur civic.mit.edu).

SYNTHÈSE

Tout, dans notre société moderne, des hôpitaux à la Bourse en passant par les journaux et les réseaux sociaux, fonctionne grâce à des logiciels. Mais à y regarder de plus près, vous verrez que les fondations de cette infrastructure logicielle menacent de céder sous la demande. Aujourd'hui, presque tous les logiciels sont tributaires de code dit *open source* : public et gratuit, ce code est créé et maintenu par des communautés de développeurs ou disposant d'autres compétences. Comme les routes ou les ponts que tout le monde peut emprunter à pied ou avec un véhicule, le code *open source* peut être repris et utilisé par n'importe qui, entreprise ou particulier, pour créer des logiciels. Ce code constitue l'infrastructure numérique de la société d'aujourd'hui, et tout comme l'infrastructure matérielle, elle nécessite une maintenance et un entretien réguliers. Aux États-Unis par exemple, plus de la moitié des dépenses de l'État pour les réseaux routiers et ceux de distribution d'eau est consacrée à leur seule maintenance[1].

Toutefois, les ressources financières nécessaires pour soutenir cette infrastructure numérique sont bien plus difficiles à obtenir. La maintenance de code *open source* était relativement abordable à ses débuts, mais de nos jours les financements ne viennent en général que d'entreprises de logiciel, sous forme de mécénat direct ou indirect. Dans la foulée de la révolution de l'ordinateur personnel, au début des années 1980, la plupart des logiciels du commerce étaient propriétaires, et non partagés. Les outils logiciels étaient conçus et utilisés en interne dans chaque entreprise, qui vendait aux clients une licence d'utilisation de ses produits. Beaucoup d'entreprises trouvaient que l'*open source* était un domaine

1. Selon une étude du bureau du budget du Congrès des États-Unis publiée en mars 2015. Voir Congressional Budget Office report : *Public Spending on Transportation and Water Infrastructure, 1956 to 2014*, 2 mars 2015, sur www.cbo.gov.

émergent trop peu fiable pour un usage commercial. Selon elles, les logiciels devaient être vendus, et non donnés.

En fait, partager du code s'est révélé plus facile, plus économique et plus efficace que d'écrire du code propriétaire, et de nos jours tout le monde utilise du code *open source* : les entreprises du Fortune 500[2], le Gouvernement, les grandes entreprises de logiciel, les *startups*… Cependant, cette demande supplémentaire a augmenté la charge de travail de ceux qui produisent et entretiennent cette infrastructure partagée. Or, comme ces communautés sont assez discrètes, les utilisateurs ont mis longtemps à s'en rendre compte. Parmi nous, beaucoup considèrent que lancer un logiciel est aussi normal que presser un bouton pour allumer la lumière, mais c'est ignorer le capital humain qui a rendu cela possible.

Face à cette demande sans précédent, si nous ne soutenons pas notre infrastructure numérique, les conséquences seront nombreuses. Du côté des risques, il y a les failles de sécurité et les interruptions de service causées par l'impossibilité pour les mainteneurs de fournir une assistance suffisante. Du côté des possibilités, les améliorations de ces outils logiciels sont nécessaires pour accompagner la renaissance actuelle des *startups*, qui dépendent étroitement de l'infrastructure numérique. De plus, le travail effectué dans l'*open source* est un atout dans le portfolio des développeurs et facilite leur recrutement, mais ce réservoir de talents est beaucoup moins diversifié que celui de l'industrie informatique dans son ensemble. Une augmentation du nombre de contributeurs serait donc profitable au domaine des technologies de l'information au sens large.

Aucune entreprise ou organisation n'a de raison de s'attaquer seule à ce problème, car le code *open source* est un bien public. C'est pourquoi nous devons réussir à travailler ensemble pour entretenir notre infrastructure numérique. Il existe par exemple la Core Infrastructure Initiative (CII) de la Fondation Linux, et le programme Open Source Support de Mozilla, ainsi que des initiatives de nombre d'entreprises de logiciels, à différents niveaux.

2. Fortune 500 est le classement des 500 premières entreprises américaines, selon l'importance de leur chiffre d'affaires. Il est publié chaque année par le magazine *Fortune* (source Wikipédia).

L'entretien de notre infrastructure numérique est une idée nouvelle pour beaucoup, et les défis que cela pose ne sont pas bien cernés. De plus, l'initiative de cette infrastructure est distribuée entre de nombreuses personnes et organisations, ce qui met à mal les modèles classiques de gouvernance. Une grande partie de ces projets qui contribuent à l'infrastructure n'a même pas de statut juridique. Toute stratégie de maintenance devra donc accepter et exploiter ces aspects décentralisés et communautaires du code *open source*.

Enfin, pour construire un écosystème sain et durable, il sera crucial d'éduquer tout un chacun à ce problème, de faciliter les contributions financières et humaines des institutions, de multiplier le nombre de contributeurs *open source* et de définir les bonnes pratiques et stratégies au sein des projets qui participent de cette infrastructure.

INTRODUCTION

En 1998, une équipe d'experts en sécurité se constitua au Royaume-Uni pour élaborer une panoplie d'outils de chiffrement libres destinés à Internet.

Très vite, tout le monde se mit à parler de leur projet, intitulé OpenSSL (les développeurs avaient pris comme base de départ un projet australien existant, SSLeay). Non seulement il était complet et relativement fiable, mais il était libre. Il n'est pas facile d'écrire de la cryptographie et OpenSSL avait résolu un problème épineux pour les développeurs du monde entier : en 2014, deux tiers des serveurs web utilisaient OpenSSL, et les sites pouvaient donc transmettre de façon sécurisée les codes de cartes de crédit et autres informations sensibles *via* Internet.

Pendant ce temps, le projet était toujours géré de façon informelle par un petit groupe de volontaires. Un conseiller du département de la Défense des États-Unis, Steve Marquess, avait remarqué qu'un contributeur, Stephen Henson, travaillait à temps plein sur OpenSSL. Par curiosité, Marquess lui demanda ce qu'il gagnait et il fut surpris d'apprendre que le salaire de Henson était cinq fois inférieur au sien.

Marquess s'était toujours considéré comme un bon programmeur, mais ses talents faisaient pâle figure à côté de ceux de Henson. Comme bien d'autres, Marquess imaginait à tort que quelqu'un d'aussi talentueux que Henson aurait un salaire à sa mesure.

Henson travaillait sur OpenSSL depuis 1998. Marquess avait rejoint le projet plus récemment, au début des années 2000, et avait travaillé avec Henson pendant plusieurs années avant d'apprendre sa situation financière.

Par son travail au département de la Défense, Marquess savait à quel point OpenSSL était crucial non seulement pour leur propre système, mais aussi pour d'autres industries dans le monde, de l'investissement à l'aéronautique en passant par la santé. Jusqu'alors, il avait « toujours supposé (comme le reste du

monde) que l'équipe d'OpenSSL était grande, active et bien financée »[1]. En réalité, OpenSSL ne rapportait même pas assez pour payer un seul salarié.

Marquess décida de s'impliquer dans le projet : il avait contribué au code de temps à autre, mais il se rendit compte qu'il serait plus utile en tant qu'homme d'affaires. Il commença par négocier des petits contrats de conseil par le biais d'une entreprise à but non lucratif pour maintenir à flot OpenSSL dans ses années les plus dures. Comme le volume des contrats croissait, il créa une entité légale pour collecter ces revenus, l'OpenSSL Software Foundation (OSF). Malgré le nombre de personnes et d'entreprises qui utilisaient leur logiciel, l'OSF ne reçut jamais plus de 2 000 dollars de dons par an. Les revenus bruts de l'activité de conseil et des contrats ne dépassèrent jamais un million de dollars, qui furent presque entièrement dépensés en frais d'hébergement et en tests de sécurité (qui peuvent coûter plusieurs centaines de milliers de dollars).

Il y avait juste de quoi payer le salaire d'un développeur, Stephen Henson. Cela signifie que les deux tiers du Web reposaient sur un logiciel de chiffrement maintenu par un seul employé à temps plein.

L'équipe d'OpenSSL continua de travailler de façon relativement anonyme jusqu'en avril 2014, quand un ingénieur de chez Google, Neel Mehta, découvrit une faille de sécurité majeure dans OpenSSL. Deux jours plus tard, un autre ingénieur, de l'entreprise finlandaise Codenomicon, découvrit le même problème.

Tous deux contactèrent immédiatement l'équipe d'OpenSSL.

Ce bug, surnommé Heartbleed[2], s'était glissé dans une mise à jour de 2011. Il était passé inaperçu pendant des années. Heartbleed pouvait permettre à n'importe quel pirate suffisamment doué de détourner des informations sécurisées en transit vers des serveurs vulnérables, y compris des mots de passe, des identifiants de cartes de crédit et autres données sensibles.

1. Les propos de Steve Marquess ont été recueillis par l'auteure lors d'interviews par téléphone et par courriel.
2. Pour en savoir plus sur Heartbleed, voir l'article « Heartbleed », sur Wikipédia.

Joseph Steinberg, un éditorialiste spécialisé en cybersécurité, écrivit : « On pourrait dire que Heartbleed est la pire vulnérabilité découverte… depuis qu'Internet a commencé à être utilisé pour des opérations commerciales. »

Grâce à un large écho médiatique, le grand public entendit parler de ce bug informatique, au moins de nom. Des plateformes majeures, comme Instagram, Gmail ou Netflix, furent affectées par Heartbleed.

Certains journalistes attirèrent l'attention sur l'OpenSSL lui-même, et la manière dont l'équipe de développement avait lutté pendant des années pour pouvoir continuer ses travaux. Les experts en sécurité connaissaient les limites d'OpenSSL, mais l'équipe ne parvenait pas à capter les ressources ou l'attention adéquates pour résoudre les problèmes.

Marquess écrivit à propos de Heartbleed : « Ce qui est mystérieux, ce n'est pas qu'une poignée de bénévoles surchargés de travail ait raté ce bug, mais plutôt qu'il n'y ait pas eu davantage de bugs de ce genre. »

Des personnes envoyèrent des dons pour soutenir la fondation, et Marquess les remercia pour leur enthousiasme, mais le premier cycle de dons ne totalisa qu'environ 9 000 dollars : largement en-deçà du nécessaire pour soutenir une équipe dédiée.

Marquess adressa alors à Internet un vibrant plaidoyer pour une levée de fonds :

> Les gars qui travaillent sur OpenSSL ne sont là ni pour l'argent ni pour la gloire (qui, en dehors des cercles geeks, a entendu parler d'eux ou d'OpenSSL avant que les médias ne s'emparent d'Heartbleed ?). Ils travaillent pour la fierté de créer et parce qu'ils se sentent responsables de ce en quoi ils croient.
>
> Il faut des nerfs d'acier pour travailler pendant des années sur des centaines de milliers de lignes d'un code très complexe, où tout le monde peut voir chacune des lignes que vous manipulez, en sachant que ce code est utilisé par des banques, des pare-feux, des systèmes d'armement, des sites web, des smartphones, l'industrie, le Gouvernement, partout. Et tout cela en

> acceptant de ne pas être apprécié à votre juste valeur et d'être ignoré jusqu'à ce que quelque chose tourne mal.
>
> Il devrait y avoir au moins une demi-douzaine de membres à temps plein dans l'équipe au lieu d'un seul pour se consacrer au soin et à la maintenance que demande OpenSSL, sans devoir gérer en même temps l'aspect commercial.
>
> Si vous êtes un décideur dans une multinationale ou un Gouvernement, pensez-y. Je vous en prie. Je me fais vieux, je fatigue et j'aimerais un jour prendre ma retraite.

Après Heartbleed, OpenSSL a obtenu enfin le financement nécessaire – en tout cas jusqu'à présent. L'équipe dispose à l'heure actuelle d'assez d'argent pour payer quatre employés à temps plein pendant trois ans. Mais au bout d'un an et demi de ce financement, Marquess n'est malgré tout pas certain de l'avenir.

Il a admis que Heartbleed a été une bénédiction pour eux, mais trouve « légèrement ironique » que ce soit une faille de cette ampleur qui ait donné plus de visibilité à leur cause. Et quand les fonds seront épuisés et que le monde sera passé à autre chose, Marquess craint qu'ils ne se retrouvent dans la même situation qu'avant Heartbleed, voire pire : la clientèle que Marquess a mis des années à se constituer a disparu, puisque l'équipe se consacre désormais exclusivement à OpenSSL et n'a plus le temps d'honorer d'autres contrats.

Marquess lui-même a bientôt l'âge de la retraite. Il est le seul qui accepte de s'occuper des affaires commerciales et du rôle exécutif associés à OpenSSL comme les impôts, la recherche de clients, et la gestion des donateurs. Le reste de son équipe préfère se concentrer sur l'écriture et la maintenance du code. Il ne peut embaucher personne pour le remplacer quand il prendra sa retraite, parce qu'il ne perçoit actuellement aucun salaire. « Je ne crois pas qu'on puisse tenir comme ça plus d'un an ou deux », a-t-il fait remarquer.

L'histoire d'OpenSSL n'est pas unique, et par bien des aspects, Marquess trouve que lui et son équipe font partie des mieux lotis. Bien d'autres projets sont toujours en manque de reconnaissance et de financement, alors qu'ils constituent l'infrastructure numérique, infrastructure absolument cruciale puisque tous les logiciels d'aujourd'hui, et par conséquent tous les aspects de notre vie quotidienne, en dépendent.

Relever ses courriels, lire les actualités, vérifier le prix des actions, faire des achats en ligne, aller chez le médecin, appeler le service client – qu'on le réalise ou non, tout ce que nous faisons est rendu possible par des projets comme OpenSSL. Sans eux, la technologie sur laquelle repose la société moderne ne pourrait tout simplement pas fonctionner.

Beaucoup de ces projets sont créés et maintenus par des volontaires et offerts au public gratuitement. Tous ceux qui le veulent, de Facebook au programmeur amateur, peuvent utiliser ce code pour créer leurs propres applications. Et ils le font.

S'il est difficile de croire, comme le dit Marquess, « qu'un groupe hétéroclite d'amateurs puisse faire mieux que de gigantesques sociétés avec leur argent et leurs ressources », considérez plutôt que c'est lié à la montée en puissance du travail collaboratif pair à pair dans le monde.

Des *startups* jusqu'ici impensables comme Uber ou Airbnb se sont transformées en l'espace de quelques années en poids lourds du monde des affaires et remettent en question des industries phares comme le transport ou l'hôtellerie. Des musiciens se font un nom sur YouTube ou SoundCloud plutôt qu'en passant par les majors. Créateurs et artistes concrétisent leurs idées *via* des plateformes de financement participatif telles que Kickstarter ou Patreon.

Les autres projets de l'infrastructure sont également issus de la passion et de la créativité de développeurs qui se sont dit : « je pourrais faire ça mieux », et qui collaborent pour développer et livrer du code au monde entier. La différence, c'est que des millions de personnes ont besoin de ce code dans leur vie quotidienne.

Comme le code n'est pas aussi sexy qu'une vidéo virale sur YouTube ou une campagne Kickstarter, le grand public

est très loin de pouvoir l'apprécier à sa juste valeur, si bien que le code qui a révolutionné les technologies de l'information manque très largement du soutien des institutions.

Mais nous ne pourrons ignorer cela plus longtemps.

Ces cinq dernières années, notre dépendance aux logiciels ainsi qu'au code libre et public qui les fait fonctionner s'est accélérée. Les technologies se sont fait une place dans tous les aspects de notre quotidien, et plus nous utilisons de logiciels, plus nous en créons, et plus cela demande de travail de maintenance.

Toutes les *startups* qui réussissent ont besoin d'une infrastructure publique pour assurer leur succès. Pourtant, aucune entreprise n'est assez motivée pour agir seule. Pendant que le monde progresse à toute vitesse vers l'ère moderne des *startups*, du code et des technologies, l'infrastructure reste à la traîne. Les fissures des fondations ne sont pas encore très visibles, mais elles s'élargissent. Après des années de croissance sans précédent qui nous ont propulsés dans une époque de prospérité, nous devons maintenant agir pour nous assurer que le monde que nous avons bâti en si peu de temps ne va pas s'effondrer brutalement sans crier gare.

Pour comprendre comment nous pouvons préserver l'avenir, nous devons d'abord comprendre ce qu'est le logiciel lui-même.

1

HISTOIRE ET CONTEXTE

« Le logiciel libre est devenu un mouvement, un état d'esprit. Soudain, les logiciels d'infrastructure étaient presque gratuits. »

Mark Suster, Upfront Ventures

DE QUOI SONT FAITS LES LOGICIELS

Les sites web ou applications mobiles que nous utilisons, même les plus simples, sont constitués de multiples composants plus petits, tout comme un immeuble est fait de briques et de ciment.

Imaginez par exemple que vous désiriez poster une photo sur Facebook. Vous ouvrez votre appli mobile Facebook, ce qui déclenche le logiciel pour afficher votre fil d'actualités.

Vous téléchargez une photo depuis votre téléphone, ajoutez un commentaire, puis vous cliquez sur « Envoyer ». Une autre partie du logiciel de Facebook, en charge du stockage des données, se souvient de votre identité et poste la photo sur votre profil. Finalement, une troisième partie de ce logiciel happe le message que vous avez saisi sur votre téléphone et le montre à tous vos amis à travers le monde.

Bien que ces opérations aient lieu sur Facebook, en réalité, ce n'est pas Facebook qui a développé toutes les briques nécessaires pour vous permettre de publier sur son application. Ses développeurs ont plutôt utilisé du code libre, public, mis à disposition de tous sur Internet par des bénévoles. Facebook ne publie pas la liste des projets qu'ils utilisent, mais une de ses filiales, Instagram, le fait et remercie certains de ces projets sur sa page d'accueil[1] et dans son application mobile.

Utiliser du code public est plus efficace pour des entreprises comme Facebook ou Instagram que de développer de nouveau par elles-mêmes tous les composants. Développer un logiciel est comparable à la construction d'un immeuble. Une entreprise du bâtiment ne produit pas ses marteaux et ses perceuses, et n'ira pas non plus chercher du bois pour découper les troncs en planches.

1. Voir la page du site Instagram listant les logiciels tiers utilisés par l'application (instagram.com/about/legal/libraries, en anglais).

Elle préférera acheter les outils à un fournisseur et s'adresser à une scierie pour obtenir du bois et finir le travail plus rapidement.

Grâce aux licences permissives, des sociétés telles que Facebook ou Instagram ne sont pas obligées de payer pour ce code, mais sont libres d'en profiter grassement. Ce n'est pas différent d'une entreprise de transport (Instagram) qui utilise les infrastructures routières publiques (code public) pour acheminer ses produits à des fins commerciales (application Instagram).

Mike Krieger, un des cofondateurs d'Instagram, a insisté sur ce point en 2013 et encouragé d'autres entrepreneurs à « emprunter plutôt que construire à chaque fois que c'est possible. Il existe des centaines d'excellents outils qui peuvent vous faire gagner du temps et vous permettre de véritablement vous concentrer sur le développement de votre produit »[2].

Quelques-uns des outils utilisés par une entreprise de logiciels :

- ***Frameworks* (environnements de développement) :** les *frameworks* offrent une base, une sorte d'échafaudage, une structure. Imaginez cela comme un schéma pour toute une application. Comme un plan, un *framework* définit la manière dont l'application se comportera sur mobile, ou comment ses données seront sauvegardées dans une base de données. Rails et Django, par exemple, sont des *frameworks*.
- **Langages :** les langages de programmation constituent l'épine dorsale de la communication des logiciels, comme la langue anglaise[3] qu'emploient les ouvriers du bâtiment sur un chantier pour se comprendre. Les langages de programmation permettent aux divers composants du logiciel d'agir et de communiquer entre eux. Si, par exemple, vous créez un compte sur un site internet et que vous cliquez sur « S'enregistrer », cette application peut utiliser des langages comme le JavaScript ou le Ruby pour sauvegarder vos informations dans une base de données. Parmi les langages de programmation les plus populaires, on peut mentionner le Python, le JavaScript et le C.

2. Voir Mike Krieger, « Picking tech for your startup », *Opbeat blog*, 26/04/2013.
3. Aux États-Unis, bien sûr… (Note des Traducteurs, NdT)

- **Bibliothèques :** les bibliothèques sont des fonctions préfabriquées qui accélèrent l'écriture du code d'un logiciel, tout comme une entreprise du bâtiment achète des fenêtres préfabriquées plutôt que de les assembler à partir des composants de base. Par exemple, au lieu de développer leur propre système d'identification pour les utilisateurs de leur application, les développeurs et développeuses peuvent utiliser une bibliothèque appelée OAuth. Au lieu d'écrire leur propre code pour visualiser des données sur une page web, ils ou elles peuvent utiliser une bibliothèque appelée d3.
- **Bases de données :** les bases de données stockent des informations (profils d'utilisateurs, adresses électroniques ou codes de cartes bancaires, par exemple) qui peuvent être utilisées par l'application. À chaque fois qu'une application a besoin de se souvenir d'une information qui vous concerne, l'application la stocke dans la base de données. Parmi les systèmes de gestion de bases de données (SGBD) les plus populaires, on trouve notamment MySQL et PostgreSQL.
- **Serveurs web et d'applications :** ces serveurs gèrent certaines requêtes envoyées par les utilisateurs sur Internet : on peut les voir comme des centraux téléphoniques qui répartissent les appels. Par exemple, si vous saisissez une URL dans la barre d'adresse de votre navigateur, un serveur web vous répondra en vous envoyant la page concernée. Si vous envoyez un message à un ami sur Facebook, le message sera d'abord envoyé à un serveur d'applications qui déterminera qui vous essayez de contacter, puis transmettra votre message au compte de votre ami. Parmi les serveurs web très répandus, citons Apache et Nginx.

Certains de ces outils, comme les serveurs et les bases de données, sont coûteux, surtout à l'échelle d'une entreprise, ce qui les rend plus faciles à monétiser. Par exemple, Heroku, une plateforme sur le *cloud* (sur un serveur distant) qui fournit des solutions d'hébergement et de bases de données, propose une offre de base gratuite, mais il faut payer pour avoir accès à plus de données ou de bande passante. De

grands sites reposent sur Heroku comme ceux de Toyota ou de Macy, et l'entreprise a été rachetée en 2010 par Salesforce.com pour 212 millions de dollars.

Il est plus difficile de faire payer d'autres types d'outils de développement, tels que les *frameworks*, de nombreuses bibliothèques et langages de programmation, qui sont souvent créés et maintenus par des bénévoles.

Ce type d'outils ressemble plus à des *ressources* qu'à des services qui peuvent être activés ou désactivés : les rendre payants limiterait fortement leur adoption. C'est pourquoi n'importe qui, que ce soit une entreprise milliardaire ou un adolescent apprenti développeur, peut utiliser gratuitement ces composants pour créer ses propres logiciels.

Par exemple, selon la page d'accueil d'Instagram, l'une des bibliothèques utilisées par l'entreprise est Appirater. Il s'agit d'une bibliothèque qui facilite les fonctions de rappels automatiques pour l'évaluation d'une application, à destination des utilisateurs d'iPhone. Elle a été créée en 2009 par Arash Payan, un développeur indépendant basé à Los Angeles. Le projet n'apporte aucun revenu à Payan.

C'est comme si des scieries, des centrales à béton et des magasins de matériaux fournissaient la base à une entreprise du bâtiment, puis continuaient à l'approvisionner.

COMMENT LA GRATUITÉ DES LOGICIELS A TRANSFORMÉ LA SOCIÉTÉ

La première réflexion qui vient à l'esprit est : « Pourquoi ces développeurs ont-ils rendu leur logiciel gratuit ? Pourquoi ne pas le faire payer ? » Les arguments en faveur du logiciel public reposent sur sa riche histoire politique et sociale. Mais d'abord, regardons la vérité en face : notre société n'en serait pas là où elle est aujourd'hui si des développeurs n'avaient pas rendu le logiciel libre et gratuit.

Avec le logiciel libre, la production de logiciel est plus simple et considérablement moins chère

Uber, un service de transport de personnes, a annoncé récemment que des développeurs avaient créé un système permettant de réserver une voiture en utilisant Slack (une application de développement collaboratif) et non l'application mobile Uber. Le projet a été bouclé en 48 heures par une équipe de la App Academy, une école de programmation.

Uber a constaté[1] que l'équipe avait été capable d'achever le projet rapidement, car elle « avait utilisé des bibliothèques ouvertes telles que Rails, Geocoder et Unicorn pour accélérer le développement tout en travaillant sur une base solide ».

En d'autres termes, la quantité de code que l'équipe a dû écrire par elle-même a été fortement réduite, car elle a pu utiliser des bibliothèques libres créées par d'autres.

1. Relaté dans un article de blog : Uber Developers, « Uber + Slack + Weekend. A story of open APIs », Medium.com, 10/11/2015.

Ruby Geocoder, par exemple, est une bibliothèque réalisée en 2010 et maintenue par Alex Reisner[2], un développeur indépendant. Geocoder permet à une application de chercher facilement des noms de rues et des coordonnées géographiques.

Unicorn est un serveur datant de 2009, il est administré par une équipe de sept contributeurs[3] encadrés par Eric Wong, un développeur.

Créer un nouveau logiciel n'a jamais été aussi simple, car il existe de plus en plus de portions de code « prêtes à l'emploi » dont on peut se servir. Pour en revenir à la métaphore de l'entreprise de bâtiment, il n'est plus nécessaire pour construire un immeuble de fabriquer soi-même tout ce dont on a besoin, il est plus simple d'acheter du « préfabriqué » et d'assembler fondation, structure porteuse et murs comme des legos.

Du coup, il est inutile de savoir comment construire un logiciel en partant de zéro pour être qualifié de développeur. Le service des statistiques sur le travail des États-Unis (Bureau of Labor Statistics[4]) estime que l'emploi des développeurs va augmenter de 22 % entre 2012 et 2022, soit bien plus rapidement que la moyenne dans les autres professions.

Le logiciel libre est directement responsable de la renaissance actuelle des startups

Les coûts de lancement d'une entreprise ont considérablement diminué depuis la première bulle internet de la fin des années 1990. Le capital-risqueur et ex-entrepreneur Mark Suster évoquait son expérience dans un billet de blog de 2011[5] :

2. Une brève description d'Alex Reisner est visible sur son site personnel alexreisner.com.
3. Pour connaître les noms des contributeurs d'Unicorn, voir leur site web unicorn.bogomips.org.
4. Aux États-Unis, le Bureau of Labor Statistics fournit des données sur les différents métiers : activités, niveau de formation, salaire moyen, perspective de recrutement, etc. En janvier 2017, il estimait une augmentation de 17 % du nombre de développeurs entre 2014 et 2024 (et 7 % tout emploi confondu). Voir Bureau of Labor Statistics, U.S. Department of Labor, *Occupational Outlook Handbook*, 2016-2017 Edition, « Software Developers ».
5. Voir le billet de Mark Suster, « Understanding Changes in the Software & Venture Capital Industries », *Both Sides*, 28/06/2011.

> Quand j'ai monté ma première entreprise, en 1999, l'infrastructure coûtait 2,5 millions de dollars, simplement pour commencer, et il fallait y ajouter 2,5 millions de dollars de plus pour payer l'équipe chargée de coder, lancer, gérer, démarcher et vendre notre logiciel.
>
> Nous avons à peine perçu le premier changement d'ampleur dans notre industrie. Il a été porté par l'introduction du logiciel libre et plus précisément par ce que l'on a appelé la pile LAMP. Linux (au lieu de UNIX), Apache (un logiciel de serveur web), MySQL (à la place d'Oracle) et PHP. Il y a bien sûr eu des variantes – nous préférions PostgreSQL à MySQL et beaucoup de personnes utilisaient d'autres langages de programmation que PHP.
>
> Le logiciel libre est devenu un mouvement, un état d'esprit. Soudain, les logiciels d'infrastructure étaient presque gratuits. Nous avons payé 10 % du tarif normal pour l'achat des logiciels et le reste de l'argent est allé dans le support. Un tel effondrement de 90 % des coûts engendre de l'innovation, croyez-moi.

La disponibilité actuelle des composants logiciels libres et gratuits (associée à des services d'hébergement moins chers comme Amazon Web Services et Heroku) permet à une *startup* technologique de se lancer sans avoir besoin de millions de dollars. Les entrepreneurs peuvent tout à fait sortir un produit et trouver un marché sans dépenser un seul dollar, la levée de fonds auprès de capital-risqueurs se faisant seulement après avoir prouvé la viabilité de leur projet.

Alan Schaaf, qui a fondé Imgur, un site populaire de partage d'images faisant partie des 50 sites les plus consultés au monde, a justement déclaré que les sept dollars nécessaires à l'achat du nom de domaine représentaient la seule dépense indispensable au démarrage de son entreprise. Imgur était rentable et, avant de lever 40 millions de dollars en 2014 auprès de l'entreprise de capital-risque Andreessen Horowitz, Schaaf n'a eu recours à

aucun fonds extérieur pendant cinq ans[6]. Les capital-risqueurs ainsi que les autres acteurs de l'investissement ont, à leur tour, commencé à investir des montants moindres, développant ainsi de nouvelles formes de fonds d'investissement dont voici trois exemples :

- **Fonds spécialisés dans le capital d'amorçage :** sociétés de capital-risque préférant financer la première levée de fonds, plutôt que de participer à une augmentation de capital ultérieure.
- **Fonds de micro capital-risque :** une définition assez large sous laquelle on regroupe les sociétés de capital-risque disposant de moins de 50 millions de dollars d'actifs.
- **Accélérateurs de *startup* :** des sociétés qui financent de petites sommes, souvent inférieures à 50 000 dollars, et qui conseillent et parrainent également les toutes jeunes entreprises…

Aujourd'hui, avec 10 millions de dollars, on peut financer cent entreprises contre seulement une ou deux dans les années 1990.

Le logiciel libre a simplifié l'apprentissage de la programmation, rendant la technologie accessible à tous, partout dans le monde

Si aujourd'hui vous voulez apprendre à coder chez vous, vous pouvez commencer par étudier Ruby on Rails. Rails est le nom d'un *framework* et Ruby est un langage de programmation. Quiconque dispose d'un accès internet peut installer gratuitement ces outils sur n'importe quel ordinateur. Parce qu'ils sont libres et gratuits, ils sont également très populaires, ce qui signifie qu'il existe quantité d'informations en ligne permettant de bien démarrer, du simple tutoriel au forum d'aide. Cela montre qu'apprendre comment coder est aussi accessible que d'apprendre à lire et écrire l'anglais ou le français.

6. Voir Jessica Guynn, « Imgur scores $40 million in funding from Andreessen Horowitz », *Los Angeles Times*, 03/04/2014.

Pour comparer, l'utilisation de *frameworks* et de langages non *open source* impliquaient : de payer pour y avoir accès, d'utiliser un système d'exploitation et des logiciels spécifiques et d'accepter des contraintes de licence susceptibles d'entraver le dépôt d'un brevet pour un logiciel construit sur la base de ce *framework*. Aujourd'hui, il est difficile de trouver des exemples de *frameworks* qui ne sont pas publics. L'un des plus célèbres exemples de *framework* propriétaire est le .NET, développé et sorti en 2002. En 2014, Microsoft a annoncé la sortie d'une version publique de .NET, appelée .NET Core.

Audrey Eschright, une développeuse, a décrit comment les logiciels *open source* l'ont aidée à apprendre la programmation à la fin des années 1990[7].

> Je voulais apprendre à programmer, mais je n'avais pas d'argent. Pas la version « étudiante fauchée » : ma famille était pauvre mais également dans une situation chaotique… Cela peut sembler étrange aujourd'hui, mais à l'époque il y avait en fait deux options pour quelqu'un qui voulait écrire de véritables logiciels : on pouvait utiliser un ordinateur sous Windows et payer pour les coûteux outils de développement de Microsoft, ou bien on pouvait avoir accès à un système Unix et utiliser le compilateur GCC. Mon but devint donc d'avoir accès à des systèmes Unix pour pouvoir apprendre à programmer et faire des trucs sympas.

Jeff Atwood, un développeur .NET de longue date, a expliqué sa décision d'utiliser Ruby pour un nouveau projet, Discourse, en 2013[8] :

> Quand on habite en Argentine, au Népal ou en Bulgarie par exemple, il est vraiment très difficile de démarrer en programmation avec les outils fournis par Microsoft. Les systèmes d'exploitation, les langages et

7. Voir Audrey Eschright, « For Love and For Money », lifeofaudrey.com, blog personnel.
8. Voir Jeff Atwood, « Why Ruby ? », *Coding Horror. Programming and Human Factors*, blog personnel, 22/03/2013.

> les outils *open source* permettent de mettre qui que ce soit au même niveau, ils constituent le socle sur lequel travaillera, partout dans le monde, la prochaine génération de programmeurs, celle qui nous aidera justement à changer le monde.

Le nombre de *startups* a explosé et dans leur sillage sont apparues beaucoup d'initiatives pour enseigner la programmation à la population : aux enfants et aux adolescents, mais aussi aux membres de communautés défavorisées, aux femmes ou aux personnes en reconversion professionnelle. Parmi ces initiatives, on retrouve Women Who Code, Django Girls, Black Girls Code, One Month et Dev Bootcamp.

Certaines de ces organisations offrent leurs services gratuitement, tandis que d'autres les font payer. Toutes reposent dans leur enseignement sur des logiciels libres et gratuits. Par exemple, Django Girls[9] a appris à coder à plus de 2 000 femmes dans 49 pays. Bien que l'organisation n'ait pas développé Django elle-même, elle a le droit d'utiliser ce logiciel, que les étudiantes téléchargent et utilisent gratuitement dans leur programme d'apprentissage.

Dev Bootcamp apprend à programmer aux personnes qui veulent changer de carrière, et prépare indifféremment tout individu, du professeur d'anglais au vétéran, à devenir développeur professionnel. Le programme coûte entre 12 et 14 000 dollars. Dev Bootcamp enseigne entre autres Ruby[10], JavaScript, Ruby on Rails et SQL. Les étudiants peuvent télécharger et utiliser tous ces outils gratuitement, pour lesquels Dev Bootcamp n'a pas besoin de payer. La société Dev Bootcamp a été rachetée par Kaplan en 2014 pour une somme inconnue.

Si des logiciels aussi importants n'étaient pas gratuits, beaucoup seraient dans l'incapacité de participer à la renaissance technologique actuelle. Il existe encore une multitude d'obstacles économiques et sociaux qui empêchent qu'ils soient encore plus nombreux à participer, comme le prix

9. Voir Djangogirls.org.
10. Voir la présentation de Ruby sur ruby-lang.org.

du matériel nécessaire pour avoir un ordinateur portable et une connexion internet, mais les outils de programmation eux-mêmes ne coûtent rien.

UNE RAPIDE HISTOIRE DES LOGICIELS PUBLIQUEMENT DISPONIBLES ET DE LEURS CRÉATEURS

Bien que nous ayons utilisé « *free software* » pour désigner des logiciels qui ne coûtaient rien à leurs utilisateurs, il faudrait plutôt employer l'expression « logiciel libre ». Cette expression aux riches connotations fait référence en particulier aux propriétés des licences sous lesquelles les logiciels sont publiés. Les partisans du logiciel libre soulignent le fait que « *free* » doit être compris sous l'angle de la liberté politique et non sous celui de la gratuité. Parfois, c'est le terme espagnol « *libre* » qui est utilisé pour marquer cette distinction (à la différence de « gratis », qui signifie gratuit).

Pendant les années 1970, lorsque les ordinateurs n'en étaient qu'à leurs balbutiements, les développeurs devaient construire leurs propres ordinateurs et écrire eux-mêmes des logiciels adaptés. Les logiciels n'étaient pas encore standardisés et n'étaient pas considérés comme des produits rentables.

En 1981, IBM a présenté le « IBM PC » pour « Personal Computer »[1], qui a permis au grand public d'accéder au matériel informatique. En quelques années, les ordinateurs construits sur mesure tombèrent progressivement en déclin quand tout le monde se mit à adopter le standard IBM. IBM est ainsi devenu l'ordinateur le plus présent au sein d'un marché fortement fracturé ; en 1986, IBM avait conquis plus de la moitié du marché des ordinateurs personnels.

Avec la venue de matériel standardisé est apparue la possibilité de créer des logiciels eux-mêmes standardisés. Soudain, tout le monde avait pour objectif de créer un business autour des logiciels. IBM a engagé une société inconnue à l'époque, du

1. « ordinateur personnel ». (NdT)

nom de Microsoft, pour écrire le système d'exploitation de son nouveau PC. Ce système d'exploitation, MS-DOS, fut publié en 1981. D'autres sociétés lui emboîtèrent le pas, proposant des logiciels sous licences commerciales. Ces licences empêchaient l'utilisateur de copier, modifier ou redistribuer les logiciels.

Il existe encore aujourd'hui de nombreux logiciels propriétaires comme Adobe Photoshop, Microsoft Windows ou GoToMeeting, par exemple. Alors que ces programmes propriétaires peuvent générer du profit pour les entreprises qui créent et distribuent ces produits, leurs restrictions limitent leur portée et leur diffusion. Toute modification apportée au design ou à la conception du programme doit provenir de l'entreprise elle-même. De plus, les logiciels propriétaires sont chers, ils coûtent souvent plusieurs centaines de dollars et n'autorisent l'acheteur dûment identifié à utiliser qu'une seule et unique copie.

Naturellement, certains informaticiens se sont sentis préoccupés par la tournure fermée et propriétaire que prenaient les logiciels, estimant que cela nuisait à leur véritable potentiel. Richard Stallman, un programmeur au laboratoire d'intelligence artificielle du MIT, a particulièrement ressenti la nécessité pour le logiciel d'être libre et modifiable.

Au cours des années qui suivirent, comme plusieurs de ses collègues se mettaient à travailler sur des projets de logiciels propriétaires, Stallman considéra qu'il ne pouvait ignorer la situation plus longtemps. En 1983, il a lancé GNU[2], un système d'exploitation libre, et ce faisant, a déclenché ce qui est devenu le « mouvement du logiciel libre ». Ce mouvement a galvanisé un groupe de personnes qui croyaient que les logiciels pourraient avoir une plus grande portée et bénéficier à la société si ceux-ci étaient mis librement à disposition. Stallman a fondé plus tard la Free Software Foundation[3] en 1985, afin de soutenir GNU ainsi que d'autres projets de logiciels libres.

2. GNU est l'acronyme récursif de *GNU's Not Unix* (GNU n'est pas Unix). Pour résumer, on peut dire que GNU est la version libre du système d'exploitation UNIX (qui lui n'est pas libre). Voir le site gnu.org.
3. La Free Software Foundation (en français, la Fondation pour le logiciel libre) est une organisation à but non lucratif. Pour en savoir plus, voir l'article sur Wikipédia ou le site de la Free Software Foundation sur fsf.org.

La Free Software Foundation définit le logiciel libre comme « un logiciel qui donne à l'utilisateur la liberté de le partager, l'étudier et le modifier ». GNU définit quatre libertés[4] associées à de tels logiciels ;

Un programme est un logiciel libre si vous, en tant qu'utilisateur de ce programme, avez les quatre libertés essentielles :

- la liberté d'exécuter le programme comme vous voulez, pour n'importe quel usage (liberté 0) ;
- la liberté d'étudier le fonctionnement du programme, et de le modifier pour qu'il effectue vos tâches informatiques comme vous le souhaitez (liberté 1) ; l'accès au code source est une condition nécessaire ;
- la liberté de redistribuer des copies, donc d'aider votre voisin (liberté 2) ;
- la liberté de distribuer aux autres des copies de vos versions modifiées (liberté 3) ; en faisant cela, vous donnez à toute la communauté une possibilité de profiter de vos changements ; l'accès au code source est une condition nécessaire.

Le mouvement du logiciel libre a été et continue d'être profondément engagé dans la défense d'intérêts sociaux. En 1998, lorsque Netscape libéra le code source de son navigateur populaire, le débat commença à passer de la politique à la technologie.

Certains technologues pensaient que se concentrer sur les bénéfices pratiques des logiciels libres permettrait de diffuser le message associé à un public plus large.

Ils ont par exemple souligné que le logiciel libre était moins cher à créer et qu'il permettait d'obtenir une meilleure qualité, car le public pouvait trouver des bugs et contribuer en proposant des correctifs. Ce type de pragmatisme se détachait de l'obligation morale exprimée par Stallman et ses partisans quant à l'obligation de promouvoir le logiciel libre. Ces technologues se sont réunis à Palo Alto pour une séance de discussion stratégique.

4. Pour plus de détails sur les quatre libertés du logiciel libre, voir la page « Qu'est-ce que le logiciel libre ? », sur le site de la Free Software Foundation, gnu.org.

Christine Peterson, une spécialiste des nanotechnologies[5] qui était présente, suggéra l'expression « *open source* ».

Peu de temps après, deux personnes qui assistaient aussi à cette rencontre, Bruce Perens et Eric Raymond, créèrent l'Open Source Initiative[6].

Un logiciel dont le code source est disponible publiquement sera qualifié d'« *open source* ». C'est un peu comme avoir une voiture et être capable d'ouvrir le capot pour apprendre comment elle fonctionne plutôt que d'avoir le moteur verrouillé et inaccessible. Les licences *open source* incluent toujours des clauses qui permettent au public d'utiliser, de modifier et de redistribuer le code. Sous cet angle, il n'y a pas de différence juridique entre les licences libres et les licences *open source*. En fait, certains font référence à l'*open source* comme une campagne de publicité pour le logiciel libre.

Cependant, la distinction la plus importante entre ces mouvements reste la culture qu'ils ont fait naître. Le mouvement du logiciel *open source* s'est écarté des aspects sociopolitiques du mouvement du logiciel libre pour se concentrer sur les bénéfices pratiques du développement logiciel et encourager des applications créatives et commerciales plus larges. À ce propos, Stallman a écrit : « L'*open source* est une méthodologie de développement ; le logiciel libre est un mouvement de société. »

Bien que « logiciel libre » et « logiciel *open source* » soient souvent discutés ensemble, ils sont politiquement distincts, le premier étant plus étroitement lié à l'éthique et le second au pragmatisme (dans la suite de cet ouvrage, on utilisera l'expression *open source* afin de souligner son rôle essentiel dans l'infrastructure logicielle) L'*open source* a ouvert un espace permettant l'émergence de différents styles et façons de développer du logiciel, libérés des complexités éthiques. Une organisation peut rendre son code public, mais n'accepter des changements que de certains contributeurs. Une autre organisation peut exiger que le code soit développé en public et accepter des changements de n'importe qui, de manière à ce que davantage

5. Les nanotechnologies selon Wikipédia.
6. L'Open Source Initiative est une organisation dévouée à la promotion des logiciels *open source*. Plus d'infos sur la page Wikipédia et sur le site de l'Open Source Initiative, l'OSI.

de personnes puissent prendre part au processus. En 1997, Eric Raymond a écrit un essai influent, intitulé *La cathédrale et le bazar*[7] (publié plus tard sous la forme d'un livre, en 1999), qui explore ces divers modes de développement.

Aujourd'hui, l'*open source* s'est répandu dans le monde du logiciel pour un certain nombre de raisons, liées à la fois à l'efficacité et au coût. C'est aussi comme cela qu'est bâtie une bonne partie de notre infrastructure numérique. Nous avons discuté de la façon dont la disponibilité de ces logiciels a bénéficié à toute la société, mais l'*open source* a aussi beaucoup apporté à ses créateurs.

L'*open source* revient moins cher à créer

Avant que les logiciels *open source* n'existent, les entreprises high-tech considéraient les programmes comme n'importe quel autre produit payant ; une équipe d'employés développait le produit en interne qui était ensuite vendu au grand public. Ce qui représentait un modèle économique très clair, mais impliquait aussi des coûts de développement accrus. Les logiciels propriétaires nécessitent une équipe payée à plein temps pour assurer le développement, ce qui inclut des développeurs, des designers, des commerciaux et des juristes. Il est bien moins coûteux de simplement confier le développement à une communauté de développeurs bénévoles qui conçoivent et assurent la maintenance du produit.

L'*open source* est plus facile à diffuser

On a plus envie d'adopter un logiciel dont l'usage est gratuit et de le modifier, plutôt qu'un logiciel dont la licence coûte des centaines de dollars et qui a été développé dans une boîte noire. Non seulement les développeurs vont vouloir

7. Eric Raymond est un promoteur de l'*open source*. Son livre *La cathédrale et le bazar*, traduit en français par Sébastien Blondeel, est disponible en ligne sur linux-france.org.

l'utiliser sans frais, mais ils pourraient même inciter leurs amis à l'utiliser eux aussi, ce qui va amplifier sa diffusion.

L'*open source* est plus ouvert à la personnalisation

Les logiciels *open source* sont copiables et adaptables aux besoins de chacun, avec différents degrés de permission. Si un développeur veut améliorer un logiciel existant, il ou elle peut copier le projet et le modifier (une pratique appelée « *forker* » en franglais).

Beaucoup de projets à succès ont commencé comme une modification de logiciels existants, par exemple WordPress (gestionnaire de contenu utilisé par 23 % des sites web dans le monde), PostgreSQL (l'une des bases de données parmi les plus populaires et dont l'adoption est croissante dans le monde entier), Ubuntu (un système d'exploitation) et Firefox (un des navigateurs web parmi les plus populaires). Dans le cas de WordPress, le logiciel a été forké depuis un projet existant appelé b2 (aussi connu sous le nom de cafelog). Deux développeurs, Matt Mullenweg et Mike Little, ont décidé qu'ils souhaitaient une meilleure version de b2 et ont donc forké le projet.

Mullenberg a choisi de copier b2, plutôt qu'un autre projet appelé TextPattern, car les licences b2 étaient plus permissives. Son idée d'origine, de 2003, est décrite ci-dessous ;

> Que faire ? Bon, TextPattern ressemble à tout ce que je rêve d'avoir, mais ça n'a pas l'air d'être sous une licence suffisamment en accord avec mes principes. Heureusement, b2/cafelog est sous GPL GNU General Public Licence, une licence de logiciel libre, ce qui veut dire que je peux utiliser les lignes de code existantes pour créer un fork […] Ce travail ne sera jamais perdu, car si je disparais de la surface de la Terre dans un an, le code que j'aurai écrit sera entièrement accessible à tous ; et si quelqu'un d'autre veut poursuivre le travail, libre à lui.

Si le logiciel était développé dans un environnement fermé et propriétaire, les développeurs n'auraient aucune possibilité de le modifier, à moins de travailler dans l'entreprise propriétaire. S'ils essayaient de réaliser leur propre version qui imite l'original, ils s'exposeraient à des poursuites en lien avec la propriété intellectuelle. Avec les logiciels *open source*, le développeur peut simplement modifier le logiciel lui-même et le distribuer publiquement, comme l'a fait Mullenweg. Les logiciels *open source* permettent ainsi une prolifération rapide des idées.

L'*open source* facilite l'adaptation des employés

Il faut du temps pour étudier une ressource logicielle, qu'il s'agisse d'un nouveau langage de programmation ou d'un nouveau *framework*. Si toutes les entreprises utilisaient leurs propres outils propriétaires, les développeurs auraient moins envie de changer d'entreprise, parce que leurs compétences techniques ne seraient applicables que sur leur lieu de travail actuel.

Il leur faudrait encore une fois apprendre à utiliser les outils propres à leur nouveau lieu de travail.

Quand les entreprises utilisent la technologie *open source*, un développeur détient un ensemble de compétences réutilisables, ce qui lui donne plus de liberté pour travailler là où il le souhaite. Par exemple, de nombreuses entreprises utilisent le même langage de programmation Ruby pour leurs logiciels. De plus, si leur produit lui-même est *open source*, la production appartient autant au développeur qu'à l'entreprise. Ce dernier peut emporter son travail avec lui s'il décide de la quitter (alors qu'il pourrait par exemple être au contraire limité par une clause de confidentialité si le code était propriétaire).

Tous ces bénéfices offrent plus de moyens d'action aux employés que si ces derniers avaient eu affaire à un logiciel propriétaire. De nos jours, de nombreuses entreprises mettent en avant leur utilisation de logiciels *open source* comme tactique de recrutement, parce que cette utilisation favorise le développeur.

L'*open source* est potentiellement plus stable et plus sûre

Théoriquement, quand un projet de logiciel a de nombreux contributeurs et une communauté florissante, le code devrait être moins vulnérable aux failles de sécurité et aux interruptions de service. En effet, dans ce cas, on devrait avoir plus de personnes révisant le code, cherchant des bugs et résolvant tous les problèmes repérés. Dans un environnement de logiciel propriétaire au contraire, seule l'équipe en charge du développement du code verra ce dernier. Par exemple, au lieu de 20 personnes pour examiner le code d'Oracle, un projet *open source* populaire pourrait avoir 2 000 volontaires qui recherchent les failles du code (remarquons que cette croyance n'est pas toujours en accord avec la réalité, et a parfois créé le problème inverse ; on a pu surestimer le nombre de personnes vérifiant des logiciels *open source*, alors même qu'en réalité personne n'en prenait la responsabilité. Nous y reviendrons dans une prochaine section).

Le logiciel *open source* a clairement certains avantages. Comment ces projets s'inscrivent-ils collectivement dans un écosystème plus large ?

2

FONCTIONNEMENT DU SYSTÈME ACTUEL

« Beaucoup de membres de notre groupe travaillent dans la technologie, que ce soit sur le Web ou sur des logiciels. En conséquence, ils travaillent sur des choses qui ne durent pas longtemps. »

Tim Hwang, Bay Area Infrastructure Observatory

QU'EST-CE QU'UNE INFRASTRUCTURE NUMÉRIQUE, ET COMMENT EST-ELLE CONSTRUITE ?

Dans un chapitre précédent, nous avons comparé la création d'un logiciel à la construction d'un bâtiment. Ces logiciels publiquement disponibles contribuent à former notre infrastructure numérique. Pour comprendre ce concept, voyons comment les infrastructures physiques fonctionnent.

Nous dépendons tous d'un certain nombre d'infrastructures physiques qui facilitent notre vie quotidienne. Allumer les lumières, aller au travail, faire la vaisselle : nous ne pensons pas souvent à l'endroit d'où viennent notre eau ou notre électricité, mais nous pouvons heureusement compter sur les infrastructures physiques. Les partenaires publics et privés travaillent de concert pour construire et maintenir nos infrastructures de transport, d'adduction des eaux propres et usées, nos réseaux d'électricité et de communication.

De même, nous ne pensons pas souvent à l'origine des applications et des logiciels que nous utilisons quotidiennement, mais tous recourent à du code libre et public pour fonctionner. Ensemble, dans une société où le numérique occupe une place croissante, ces projets *open source* forment notre infrastructure numérique. Toutefois, il existe plusieurs différences majeures entre les infrastructures physiques et numériques, qui affectent la manière dont ces dernières sont construites et maintenues. Il existe en particulier des différences de coût, de maintenance et de gouvernance.

Les infrastructures numériques sont plus rapides et moins chères à construire

C'est connu, construire des infrastructures matérielles coûte très cher. Les projets sont physiquement de grande envergure et peuvent prendre des mois ou des années à aboutir.

Le gouvernement fédéral des États-Unis a dépensé 96 milliards de dollars en projets d'infrastructure en 2014 et les gouvernements des différents États 320 milliards de dollars au total la même année. Un peu moins de la moitié de ces dépenses (43 %) a été affectée à de nouvelles constructions ; le reste a été consacré à des opérations de maintenance de l'infrastructure existante[1].

Proposer puis financer des projets de nouvelles infrastructures physiques peut être un processus politique très long. Le financement des infrastructures de transport a été un sujet délicat aux États-Unis d'Amérique au cours de la dernière décennie lorsque le gouvernement fédéral a été confronté à un manque de 16 milliards de dollars pour y pourvoir.

Le Congrès a récemment voté la première loi pluriannuelle de financement des transports depuis dix ans, affectant 305 milliards de dollars aux autoroutes et autres voies rapides après des années d'oppositions politiques empêchant la budgétisation des infrastructures au-delà de deux ans.

Même après qu'un nouveau projet d'infrastructure a été validé et a reçu les fonds nécessaires, il faut souvent des années pour le terminer, en raison des incertitudes et des imprévus auxquels il faut faire face.

Dans le cas du projet Central Artery/Tunnel de Boston, dans le Massachusetts, connu aussi sous le nom de Big Dig[2], neuf ans se sont écoulés entre la planification et le début des travaux. Son coût prévu était de 2,8 milliards de dollars et il devait être achevé en 1998. Finalement, le projet a coûté 14,6 milliards de

1. Voir Congressional Budget Office, *Public Spending on Transportation and Water Infrastructure, 1956 to 2014*, 02/03/2015. Il s'agit d'un document détaillant les dépenses publiques d'infrastructures physiques (eaux, transports) entre 1956 et 2014.
2. Voir la page Wikipédia « Big Dig » qui détaille l'historique et les aléas de ce projet autoroutier souterrain.

dollars et n'a pas été achevé avant 2007, ce qui en fait le projet d'autoroute le plus cher des États-Unis.

En revanche, les infrastructures numériques ne souffrent pas des coûts associés à la construction des infrastructures physiques comme le zonage[3] ou l'achat de matériel. Il est donc plus facile pour tout le monde de proposer une nouvelle idée et de l'appliquer en un temps très court. MySQL, le second système de gestion de base de données le plus utilisé dans le monde et partie intégrante d'une collection d'outils indispensables qui aidèrent à amorcer le premier boum technologique, fut lancé par ses créateurs, Michael Widenius et David Axmark, en mai 1995. Ils mirent moins de deux années à le développer[4].

Il a fallu à Ruby[5], un langage de programmation, moins de trois ans entre sa conception initiale en février 1993 et sa publication en décembre 1995. Son auteur, l'informaticien Yukihiro Matsumoto, a décidé de créer le langage après une discussion avec ses collègues.

Les infrastructures numériques se renouvellent fréquemment

Comme l'infrastructure numérique est peu coûteuse à mettre en place, les barrières à l'entrée sont plus basses et les outils de développement changent plus fréquemment.

L'infrastructure physique est construite pour durer, c'est pourquoi ces projets mettent si longtemps à être planifiés, financés et construits. Le métro de Londres, le système de transport en commun rapide de la ville, fut créé en 1863 ; les tunnels creusés à l'époque sont encore utilisés aujourd'hui.

Le pont de Brooklyn, qui relie les arrondissements de Brooklyn et de Manhattan à New York City, fut achevé en 1883 et n'a pas subi de rénovations majeures avant 2010, plus de cent ans après. L'infrastructure numérique nécessite non

3. Le zonage en termes d'urbanisme consiste à réglementer et contrôler l'utilisation du sol. « Le mot est dérivé de la pratique de diviser le territoire municipal en zones et d'attribuer à chacun des usages permis. » (source : Wikipédia)
4. Voir la page Wikipédia en français consacrée à MySQL.
5. « Ruby est un langage de programmation libre. Il est interprété, orienté objet et multiparadigme. » (source : Wikipédia)

seulement une maintenance et un entretien fréquents pour être compatible avec d'autres logiciels, mais son utilisation et son adoption changent également fréquemment. Un pont construit au milieu de New York City aura un usage garanti et logique, proportionnellement à la hausse ou à la diminution de la population. Mais un langage de programmation ou un *framework* peut être extrêmement populaire durant plusieurs années, puis tomber en désuétude lorsqu'apparaît quelque chose de plus rapide, plus efficace ou simplement plus à la mode.

Un graphique disponible sur le site www.openhub.net montre par exemple l'activité des développeurs de code source selon plusieurs langages différents[6] : le langage C, l'un des langages les plus fondamentaux et les plus utilisés, a vu sa part de marché diminuer alors que de nouveaux langages apparaissaient. Python et JavaScript, deux langages actuellement très populaires, ont quant à eux vu leur utilisation augmenter régulièrement avec le temps. Go, développé en 2007, a connu plus d'activité dans les dernières années.

Tim Hwang, dirigeant du Bay Area Infrastructure Observatory, qui organise des visites de groupe sur des sites d'infrastructures physiques, faisait remarquer la différence dans une interview de 2015 donnée au *California Sunday Magazine*[7] :

> Beaucoup de membres de notre groupe travaillent dans la technologie, que ce soit sur le Web ou sur des logiciels. En conséquence, ils travaillent sur des choses qui ne durent pas longtemps. Leur approche c'est « on a juste bidouillé ça, et on l'a mis en ligne » ou « on l'a simplement publié, on peut travailler plus tard sur les bugs ». Beaucoup d'infrastructures sont construites pour durer cent ans. On ne peut pas se permettre d'avoir des bugs. Si on en a, le bâtiment s'écroule. On ne peut pas l'itérer. C'est une façon de concevoir qui échappe à l'expérience quotidienne de nos membres.

6. https://www.openhub.net/languages/compare.
7. Allison Arieff, « The Infrastructure Tourist. Tim Hwang wants to show you how the world works », *The California Sunday Magazine*, 29/10/2015.

Cependant, comme l'infrastructure numérique change très fréquemment, les projets plus anciens ont plus de mal à trouver des contributeurs, parce que beaucoup de développeurs préfèrent travailler sur des projets plus récents et plus excitants. Ce phénomène est parfois nommé le « syndrome de la pie » chez les développeurs, ces derniers étant attirés par les choses « nouvelles et brillantes », et non par les technologies qui fonctionnent le mieux pour eux et pour leurs utilisateurs.

Les infrastructures numériques n'ont pas besoin d'autorité organisatrice pour déterminer ce qui doit être construit ou utilisé

En définitive, la différence la plus flagrante entre une infrastructure physique et une infrastructure numérique, et c'est aussi un des défis majeurs pour sa durabilité, c'est qu'il n'existe aucune instance décisionnelle pour déterminer ce qui doit être créé et utilisé dans l'infrastructure numérique. Les transports, les réseaux d'adduction des eaux propres et usées sont généralement gérés et possédés par des collectivités, qu'elles soient fédérales, régionales ou locales. Les réseaux électriques et de communication sont plutôt gérés par des entreprises privées. Dans les deux cas, les infrastructures sont créées avec une participation croisée des acteurs publics et privés, que ce soit par le budget fédéral, par les entreprises privées ou les contributions payées par les usagers.

Dans un État stable et développé, nous nous demandons rarement comment une route est construite ou un bâtiment électrifié. Même pour des projets financés ou propriétés du privé, le gouvernement fédéral a un intérêt direct à ce que les infrastructures physiques soient construites et maintenues.

De leur côté, les projets d'infrastructures numériques sont conçus et construits en partant du bas. Cela ressemble à un groupe de citoyens qui se rassemblent et décident de construire un pont ou de créer leur propre système de

recyclage des eaux usées. Il n'y a pas d'organe officiel de contrôle auquel il faudrait demander l'autorisation pour créer une nouvelle infrastructure numérique.

Internet lui-même possède deux organes de contrôle qui aident à définir des standards : l'IETF[8] (Internet Engineering Task Force) et le W3C (World Wide Web Consortium). L'IETF aide à développer et définit des standards recommandés sur la façon dont les informations sont transmises sur Internet. Par exemple, ils sont la raison pour laquelle les URL commencent par « HTTP ». Ils sont aussi la raison pour laquelle nous avons des adresses IP – des identifiants uniques assignés à votre ordinateur lorsqu'il se connecte à un réseau. À l'origine, en 1986, il s'agissait d'un groupe de travail au sein du gouvernement des États-Unis, mais l'IETF est devenue une organisation internationale indépendante en 1993.

L'IETF elle-même fonctionne grâce à des bénévoles et il n'y a pas d'exigences pour adhérer : n'importe qui peut rejoindre l'organisation en se désignant comme membre. Le W3C (World Wide Web Consortium) aide à créer des standards pour le World Wide Web. Ce consortium a été fondé en 1994 par Tim Berners-Lee. Le W3C a tendance à se concentrer exclusivement sur les pages web et les documents (il est, par exemple, à l'origine de l'utilisation du HTML pour le formatage basique des pages web). Il maintient les standards autour du langage de balisage HTML et du langage de formatage de feuilles de style CSS, deux des composants de base de n'importe quelle page web. L'adhésion au W3C, légèrement plus formalisée, nécessite une inscription payante. Ses membres vont des entreprises aux étudiants en passant par des particuliers.

L'IETF et le W3C aident à gérer les standards utilisés par les pièces les plus fondamentales d'Internet, mais ceux qui concernent la couche supérieure – les choix du langage utilisé pour créer le logiciel, des *frameworks* pour les créer, des bibliothèques à utiliser – sont entièrement gérés dans le domaine public. Bien entendu, de nombreux projets de logiciels propriétaires, particulièrement ceux qui sont régis par de très nombreuses normes, dans des secteurs comme l'aéronautique

8. Voir la page Wikipédia en français à propos de l'IETF.

ou la santé, peuvent avoir des exigences concernant les outils utilisés. Ils peuvent même développer des outils propriétaires pour leur propre utilisation.

Avec les infrastructures physiques, si le gouvernement construit un nouveau pont entre San Francisco et Oakland, ce pont sera certainement utilisé. De la même façon, lorsque le W3C décide d'un nouveau standard, tel qu'une nouvelle version de HTML, il est formellement publié et annoncé. Par exemple, en 2014, le W3C a annoncé HTML 5, la première révision majeure de HTML depuis 1997, qui a été développé pendant sept ans.

En revanche, un informaticien ou une informaticienne qui souhaite créer un nouveau langage de programmation est libre de le publier et ce langage peut être adopté ou non. La barre d'adoption est encore plus basse pour les *frameworks* et bibliothèques : parce qu'ils sont plus faciles à créer et plus faciles pour un utilisateur à apprendre et implémenter, ces outils sont itérés plus fréquemment.

Mais le plus important, c'est que personne ne contraint ni même n'encourage fortement quiconque à utiliser ces projets. Certains d'entre eux restent plus théoriques que pratiques, d'autres sont totalement ignorés. Il est difficile de prédire ce qui sera véritablement utilisé avant que des personnes ne commencent justement à l'utiliser.

Les développeurs aiment se servir de l'utilité comme indicateur de l'adoption ou non d'un projet. Les nouveaux projets doivent améliorer un projet existant ou résoudre un problème chronique pour être considérés comme utiles et dignes d'être adoptés. Si vous demandez aux développeurs pourquoi leur projet est devenu si populaire, beaucoup hausseront les épaules et répondront : « C'était la meilleure chose disponible. » Contrairement aux *startups* technologiques, les nouveaux projets d'infrastructure numérique reposent sur les effets de réseau pour être adoptés par le plus grand nombre.

L'existence d'un groupe noyau de développeurs motivés par le projet ou d'une entreprise de logiciels qui l'utilise contribue à sa diffusion. Un nom facilement mémorisable, une bonne promotion, ou un beau site internet peuvent

ajouter au facteur « nouveauté » du projet. La réputation d'un développeur dans sa communauté est aussi un facteur déterminant dans la diffusion d'un projet.

Mais, en fin de compte, une nouvelle infrastructure numérique peut venir d'à peu près n'importe où, ce qui signifie que chaque projet est géré et maintenu d'une façon qui lui est propre.

COMMENT LES PROJETS D'INFRASTRUCTURE NUMÉRIQUE SONT-ILS GÉRÉS ET MAINTENUS ?

Nous avons établi que les infrastructures numériques sont aussi nécessaires à la société moderne que le sont les infrastructures physiques. Bien qu'elles ne soient pas sujettes aux coûts élevés et aux obstacles politiques auxquels sont confrontées ces dernières, leur nature décentralisée les rend cependant plus difficiles à cerner. Sans une autorité centrale, comment les projets *open source* trouvent-ils les ressources dont ils ont besoin ?

En un mot, la réponse est différente pour chaque projet. Cependant, on peut identifier plusieurs lieux d'où ces projets peuvent émaner : au sein d'une entreprise, *via* une *startup*, de développeurs individuels ou collaborant en communauté.

Au sein d'une entreprise

Parfois, le projet commence au sein d'une entreprise. Voici quelques exemples qui illustrent par quels moyens variés un projet *open source* peut être soutenu par les ressources de cette dernière :

- **Go**[1], le nouveau langage de programmation évoqué précédemment, a été développé au sein de Google en 2007 par les ingénieurs Robert Griesemer, Rob Pike et Ken Thompson, pour qui la création de Go était une expérimentation. Go est *open source* et accepte les contributions

1. Go est un langage de programmation compilé, il se fonde sur le langage C. Pour plus d'information, voir la page Wikipédia « Go (langage) ».

de la communauté dans son ensemble. Cependant, ses principaux mainteneurs sont employés à plein temps par Google pour travailler sur le langage.

- **React**[2] est une nouvelle bibliothèque JavaScript dont la popularité grandit de jour en jour. React a été créée par Jordan Walke, ingénieur logiciel chez Facebook, pour un usage interne sur le fil d'actualités Facebook. Un employé d'Instagram (qui est une filiale de Facebook) a également souhaité utiliser React, qui finalement a été placée en *open source*, deux ans après son développement initial. Facebook a dédié une équipe d'ingénieurs à la maintenance du projet, mais React accepte aussi les contributions de la communauté publique des développeurs.
- **Swift**[3], le langage de programmation utilisé pour iOS, OS X et les autres projets d'Apple, est un exemple de projet qui n'a été placé que récemment en *open source*. Swift a été développé par Apple en interne pendant quatre ans et publié en tant que langage propriétaire en 2014. Les développeurs avaient le droit d'utiliser Swift pour écrire des programmes pour Apple, mais ne pouvaient pas contribuer au développement du cœur du langage. En 2015, Swift a été rendu *open source* sous la licence Apache 2.0.

Pour une entreprise, les incitations à maintenir un projet *open source* sont nombreuses. Ouvrir un projet au public peut signifier moins de travail pour l'entreprise, grâce essentiellement aux améliorations de la collaboration de masse.

Cela stimule la bonne volonté et l'intérêt des développeurs, qui peuvent alors être incités à utiliser d'autres ressources de l'entreprise pour construire leurs projets.

Disposer d'une communauté active de programmeurs crée un vivier de talents à recruter. Et parfois, l'ouverture du code d'un projet aide une entreprise à renforcer sa base d'utilisateurs et sa marque, ou même à l'emporter sur la concurrence. Plus une entreprise peut capter de parts de marché, même à travers des outils qu'elle distribue

2. Pour plus d'information sur cette bibliothèque JavaScript créée par Facebook, voir la page Wikipédia « React (JavaScript) ».
3. Pour plus d'info sur ce langage de programmation créé par Apple, voir la page Wikipédia « Swift (langage d'Apple) ».

gratuitement, plus elle devient influente. Ce n'est pas très différent du concept commercial de « produit d'appel ».

Même quand un projet est créé en interne, s'il est *open source*, alors il peut être librement utilisé ou modifié selon les termes d'une licence libre, et n'est pas considéré comme relevant de la propriété intellectuelle de l'entreprise au sens traditionnel. De nombreux projets d'entreprise utilisent des licences libres standard qui sont considérées comme acceptables par la communauté des développeurs telles que les licences Apache 2.0 ou BSD. Cependant, dans certains cas, les entreprises ajoutent leurs propres clauses. La licence de React, par exemple, comporte une clause additionnelle qui pourrait potentiellement créer des conflits de revendications de brevet avec les utilisateurs de React.

En conséquence, certaines entreprises et individus sont réticents à utiliser React, et cette décision est fréquemment décrite comme un exemple de conflit avec les principes de l'*open source*.

Via une *startup*

Certains projets d'infrastructures empruntent la voie traditionnelle de la *startup*, ce qui inclut des financements en capital-risque.Voici quelques exemples :

- **Docker**, qui est peut-être l'exemple contemporain le plus connu, aide les applications logicielles à fonctionner à l'intérieur d'un conteneur. (Les conteneurs procurent un environnement propre et ordonné pour les applications logicielles, ce qui permet de les faire fonctionner plus facilement partout.) Docker est né en tant que projet interne chez dotCloud, une société de plateforme en tant que service (ou PaaS, pour *Platform as a Service* en anglais), mais le projet est devenu si populaire que ses fondateurs ont décidé d'en faire la principale activité de l'entreprise. Le projet Docker a été placé en *open source* en 2013. Il a collecté 180 millions de dollars, avec une valeur estimée à plus d'un

milliard de dollars[4]. Son modèle économique repose sur du support technique, des projets privés et des services. Les revenus de Docker pour l'année 2014 ne dépassaient pas 10 millions de dollars.

- **Npm** est un gestionnaire de paquets publié en 2010 pour aider les développeurs de Node.js à partager et à gérer leurs projets. Npm a collecté près de 11 millions de dollars de financements depuis 2014 de la part de True Ventures et de Bessemer Ventures, entre autres. Son modèle économique se concentre sur des fonctionnalités payantes en faveur de la vie privée et de la sécurité.
- **Meteor** est un *framework* JavaScript publié pour la première fois en 2012. Il a bénéficié d'un programme d'incubation au sein de Y Combinator, un prestigieux accélérateur de *startups* qui a également été l'incubateur d'entreprises comme Airbnb et Dropbox. À ce jour, Meteor a reçu plus de 30 millions de dollars de financements de la part de firmes comme Andreessen Horowitz ou Matrix Partners. Le modèle économique de Meteor s'appuie sur une plateforme d'entreprise nommée Galaxy, sortie en octobre 2015, qui permet de faire fonctionner et de gérer les applications Meteor.

L'approche fondée sur le capital-risque est relativement nouvelle, et se développe rapidement[5]. Lightspeed Venture Partners a constaté qu'entre 2010 et 2015, les sociétés de capital-risque ont investi plus de 4 milliards de dollars dans des entreprises *open source*, soit dix fois plus que sur les cinq années précédentes.

Le recours aux fonds de capital-risque pour soutenir les projets *open source* a été accueilli avec scepticisme par les développeurs (et même par certains acteurs du capital-risque eux-mêmes), du fait de l'absence de modèles économiques ou de revenus prévisibles pour justifier les estimations. Steve Klabnik, un mainteneur du langage Rust, explique[6] le soudain intérêt des capital-risqueurs pour le financement de l'*open source* :

4. Concernant la valorisation de Docker, voir Jordan Novet, « Docker, now valued at $1B, paid someone $799 for its logo on 99designs », *VB*, 13/07/2015.
5. Sur le financement des *startups*, voir John Vrionis, « It's actually open source software that's eating the world », *VB*, 06/12/2015.
6. Voir Steve Klabnik, « Is npm worth $2.6MM? », blog personnel, 12/02/2014.

> Je suis un investisseur en capital-risque. J'ai besoin qu'un grand nombre d'entreprises existent pour gagner de l'argent… J'ai besoin que les coûts soient bas et les profits élevés. Pour cela, il me faut un écosystème de logiciels *open source* en bonne santé. Donc je fais quoi ?… Les investisseurs en capital-risque sont en train de prendre conscience de tout ça, et ils commencent à investir dans les infrastructures. […] Par bien des aspects, le matériel *open source* est un produit d'appel, pour que tu deviennes accro… puis tu l'utilises pour tout, même pour ton code propriétaire. C'est une très bonne stratégie commerciale, mais cela place GitHub au centre de ce nouvel univers. Donc pour des raisons similaires, a16z a besoin que GitHub soit génial, pour servir de tremplin à chacun des écosystèmes *open source* qui existeront à l'avenir… Et a16z a suffisamment d'argent pour en « gaspiller » en finançant un projet sur lequel ils ne récupéreront pas de bénéfices directs, parce qu'ils sont assez intelligents pour investir une partie de leurs fonds dans le développement de l'écosystème.

GitHub, créé en 2008, est une plateforme de partage et stockage de code, disponible en mode public ou privé, doté d'un environnement ergonomique. Il héberge de nombreux projets *open source* populaires et, surtout, il est devenu l'épicentre culturel de la croissance explosive de l'*open source* (dont nous parlerons plus loin).

GitHub[7] n'a reçu aucun capital-risque avant 2012, quatre ans après sa création. Avant cette date, GitHub était une entreprise rentable. Depuis 2012, GitHub a reçu au total 350 millions de dollars de financements en capital-risque.

Andreessen Horowitz (alias a16z), la firme d'investissement aux 4 milliards de dollars qui a fourni l'essentiel du capital de leur première levée de fonds de 100 millions de dollars, a déclaré qu'il s'agissait là de l'investissement le plus important qu'elle ait jamais fait jusqu'alors.

7. Sur le financement de GitHub, voir la page Crunchbase consacrée à GitHub.

En d'autres termes, la théorie de Steve Klabnik est que les sociétés de capital-risque qui investissent dans les infrastructures *open source* promeuvent ces plateformes en tant que « produit d'appel », même quand il n'y a pas de modèle économique viable ou de rentabilité à en tirer, parce que cela permet de faire croître l'ensemble de l'écosystème. Plus GitHub a de ressources, plus l'*open source* est florissant. Plus l'*open source* est florissant et mieux se portent les *startups*. À lui seul, l'intérêt des sociétés d'investissement pour l'*open source*, particulièrement quand on considère l'absence de véritable retour financier, est une preuve du rôle clé que joue l'*open source* dans l'écosystème plus large des *startups*.

Par ailleurs, il est important de noter que la plateforme GitHub en elle-même n'est pas un projet *open source* et n'est donc pas un exemple de capital-risque finançant directement l'*open source*. GitHub est une plateforme à code propriétaire qui héberge des projets *open source*. C'est un sujet controversé pour certains contributeurs *open source*.

Par des personnes ou un groupe de personnes

Enfin, de nombreux projets d'infrastructures numériques sont intégralement développés et maintenus par des développeurs indépendants ou des communautés de développeurs. Voici quelques exemples :

- **Python**, un langage de programmation, a été développé et publié par un informaticien, Guido van Rossum, en 1991. Van Rossum déclarait[8] qu'il était « à la recherche d'un projet de programmation *passe-temps*, qui le tiendrait occupé pendant la semaine de Noël ». Le projet a décollé, et Python est désormais considéré de nos jours comme l'un des langages de programmation les plus populaires[9]. Van Rossum[10] reste le principal auteur de Python (aussi connu parmi les développeurs sous le nom de « dictateur bienveillant

8. Guido van Rossum, « Foreword for *Programming Python* (1st ed.) », python.org.
9. Concernant la popularité des langages de programmation, voir « Most Popular Coding Languages of 2015 », *Blog Codeeval*, 09/02/2015.
10. Biographie de van Rossum sur Wikipédia.

à vie » et il est actuellement employé par Dropbox, dont les logiciels reposent fortement sur Python). Python est en partie géré par la Python Software Foundation[11], créée en 2001, qui bénéficie de nombreux sponsors commerciaux, parmi lesquels Intel, HP et Google.

- **RubyGems** est un gestionnaire de paquets qui facilite la distribution de programmes et de bibliothèques associés au langage de programmation Ruby. C'est une pièce essentielle de l'infrastructure pour tout développeur Ruby. Parmi les sites web utilisant Ruby, on peut citer par exemple[12] Hulu, Airbnb et Bloomberg. RubyGems a été créé en 2003 et est géré par une communauté de développeurs. Certains travaux de développement sont financés par Ruby Together, une fondation qui accepte les dons d'entreprises et de particuliers.
- **Twisted**, une bibliothèque Python, fut créée en 2002 par un programmeur nommé Glyph Lefkowitz. Depuis lors, son usage s'est largement répandu auprès d'individus et d'organisations, parmi lesquelles[13] Lucasfilm et la NASA. Twisted continue d'être géré par un groupe de volontaires. Le projet est soutenu par des dons corporatifs/commerciaux et individuels ; Lefkowitz en reste l'architecte principal et gagne sa vie en proposant ses services de consultant[14].

Comme le montrent tous ces exemples, les projets *open source* peuvent provenir de pratiquement n'importe où, ce qui est en général considéré comme une bonne chose. Cela signifie que les projets utiles ont le plus de chances de réussir, car ils évitent d'une part les effets de mode futiles inhérents aux *startups*, et d'autre part la bureaucratie propre aux gouvernements. La nature décentralisée de l'infrastructure numérique renforce également les valeurs de démocratie et d'ouverture d'Internet, qui permet en principe à chacun de créer le prochain superprojet, qu'il s'agisse d'une entreprise ou d'un individu.

11. Fondation du logiciel Python (NdT).
12. Pour avoir un panorama des sites utilisant Ruby, voir Kelli Smith, « 37 Sites You LOVE Built With Ruby On Rails », *Skillcrush*, 01/02/2017.
13. Voir *Sucess stories*, sur le site Twisted Matrix Labs.
14. Voir le CV de Glyph Lefkowitz, sur twistedmatrix.com.

D'un autre côté, un grand nombre de projets utiles proviendront de développeurs indépendants, qui se trouveront tout à coup à la tête d'un projet à succès et devront prendre des décisions cruciales pour son avenir. Une étude de 2015 menée par l'Université fédérale de Minas Gerais au Brésil a examiné 133 des projets les plus activement utilisés sur GitHub, parmi les langages de programmation, et a découvert que 64 % d'entre eux, presque les deux tiers, dépendaient pour leur survie d'un ou deux développeurs seulement[15].

Bien qu'il puisse y avoir une longue traîne de contributeurs occasionnels ou ponctuels pour de nombreux projets, les responsabilités principales de la gestion du projet ne reposent que sur un très petit nombre d'individus.

Coordonner des communautés internationales de contributeurs aux avis arrêtés, tout en gérant les attentes d'entreprises classées au Fortune 500 qui utilisent votre projet, voilà des tâches qui seraient des défis pour n'importe qui. Il est impressionnant de constater combien de projets ont déjà été accomplis de cette manière. Ces tâches sont particulièrement difficiles dans un contexte où les développeurs manquent de modèles clairement établis, mais aussi de soutien institutionnel pour mener à bien ce travail. Au cours d'interviews menées pour cette étude, beaucoup de développeurs se sont plaints en privé qu'ils n'avaient aucune idée de qui ils pouvaient solliciter pour obtenir de l'aide et qu'ils préféreraient « juste coder ».

Pourquoi continuent-ils à le faire ? La suite de cette étude se concentrera sur pourquoi et comment les contributeurs de l'*open source* maintiennent des projets à grande échelle, et sur les raisons pour lesquelles c'est important pour nous tous.

15. Voir l'article de l'Université de Minas Gerais : Guilherme Avelino, Marco Tulio Valente, Andre Hora, « What is the Truck Factor of Popular GitHub Applications? A First Assessment », *PeerJPreprints*, 02/01/2017.

POURQUOI DES PERSONNES CONTINUENT-ELLES DE CONTRIBUER À CES PROJETS, ALORS QU'ELLES NE SONT PAS PAYÉES POUR LE FAIRE ?

De nombreuses infrastructures numériques sont entretenues par des contributeurs individuels ou par une communauté de contributeurs. Dans la plupart des cas, ces contributeurs ne sont pas directement rémunérés pour ce travail. En réalité, ils contribuent pour des raisons propres aux communautés *open source*, pour se construire une réputation, par exemple, ou dans un esprit de service public. Ce chapitre explorera certaines de ces motivations plus en détail.

Contribuer à l'*open source* pour se construire une réputation

Se construire une réputation est peut-être la motivation la plus concrète pour contribuer à un projet *open source*. Pour les développeurs, rédacteurs techniques et autres, ces projets sont une occasion de faire leurs preuves en public et leur offrent une chance de participer à quelque chose de grand et d'utile.

Dans le cadre d'un programme appelé Google Summer of Code[1], Google propose des bourses d'été à des étudiants développeurs pour qu'ils contribuent à des projets *open source* populaires. Le programme fonctionne bien, parce que les

1. L'été du code de Google (NdT).

développeurs sont des étudiants, novices dans le domaine de l'informatique et avides de démontrer leurs talents.

Les développeurs, en particulier, tirent profit de leurs expériences de contribution à l'*open source* pour se constituer un portfolio. De plus, en participant à des projets populaires dotés de communautés actives, un développeur a des chances de se construire une réputation en devenant « connu ».

GitHub, site web déjà cité, est une plateforme populaire pour l'élaboration collaborative de code. Quand un développeur contribue à un projet public de logiciel, ses contributions apparaissent sur son profil. Le profil GitHub d'un développeur peut faire office de portfolio pour des sociétés de logiciels, mais seules les contributions effectuées pour des projets publics (c'est-à-dire *open source*) sont visibles par tous.

Cependant, les motivations fondées sur la réputation ne sont pas dénuées de risques, surtout parmi les jeunes développeurs. Un développeur dont la carrière est encore naissante peut contribuer à un projet *open source* dans le seul but de trouver un employeur, puis arrêter de le faire une fois cet objectif atteint. De plus, un développeur cherchant uniquement à se construire un portfolio risque de proposer des contributions de moindre qualité, qui ne seront pas acceptées et qui pourront même ralentir le processus de développement du projet. Enfin, si le but d'une contribution publique de la part d'un développeur est de se construire une réputation, alors ce développeur sera tenté de contribuer uniquement aux projets les plus populaires et attractifs (une extension du « syndrome de la pie », déjà évoqué), et de ce fait, les projets plus anciens auront du mal à trouver de nouveaux contributeurs.

Le projet est devenu de manière inattendue très populaire et son développeur se sent obligé d'en assurer le suivi

Des entreprises, des particuliers ou des organisations peuvent devenir dépendants d'un projet *open source* populaire. En d'autres termes, le code est utilisé dans des logiciels

opérationnels, écrits et déployés par d'autres, qui peuvent servir pour un tas de choses : achats en ligne ou soins médicaux. En raison de la complexité du réseau de dépendances (dont beaucoup ne sont même pas connues de l'auteur du projet, puisqu'il ne peut pas savoir exactement qui utilise son code), la personne qui maintient le projet peut se sentir moralement obligée de continuer à l'entretenir.

Arash Payan, le développeur d'Appirater, a publié son projet en 2009. Au sujet de sa décision de continuer à le maintenir, il déclare :

> Ce n'est pas quelque chose de très excitant, mais il y a tellement de personnes qui utilisent le projet (qui en dépendent, même ?) pour leurs applications, que je me sens obligé d'en prendre soin correctement. Personnellement, j'ai quitté iOS, donc maintenir une bibliothèque iOS n'est pas exactement la première chose que je veux faire de mon temps libre.

Payan estime que maintenir le projet à jour lui prend une à deux heures par mois maximum, donc cela ne le dérange pas.

Mais certains projets devenus étonnamment populaires prennent plus de temps à être maintenus. Andrey Petrov[2] est un développeur indépendant qui a écrit une bibliothèque Python appelée urllib3. Sa publication en 2008 fut une avancée majeure pour la bibliothèque standard déjà existante, et elle est devenue populaire parmi les développeurs Python. Aujourd'hui, tous les utilisateurs de Python en dépendent.

Andrey en a fait un projet *open source* dans l'espoir que d'autres personnes soutiendraient son développement et sa maintenance. C'est un développeur indépendant ; bien qu'il apprécie de maintenir urllib3, il ne peut s'en occuper que pendant son temps libre puisqu'il n'est pas payé pour ce travail. Cory Benfield, qui est employé par la Hewlett Packard

2. Andrey Petrov, « Urllib3, Stripe, and Open Source Grants. Helping the future come a little sooner », Medium.com, 07/07/2014.

Enterprise (HPE) pour aider à maintenir des bibliothèques Python d'importance critique (que HPE utilise et dont HPE dépend), est désormais affecté à la maintenance de urllib3 dans le cadre de son travail ; cela a allégé la charge de travail d'Andrey.

Le projet est une passion plus qu'un travail

Eric Holsher est l'un des créateurs de Read the Docs, une plateforme qui héberge de la documentation sur des logiciels. La documentation est l'équivalent d'un mode d'emploi. De même qu'on a besoin d'un mode d'emploi pour monter un meuble, un développeur a besoin de documentation pour savoir comment implémenter un projet. Sans la documentation adéquate, il lui serait difficile de savoir par où commencer.

Read the Docs[3] fournit de la documentation pour 18 000 logiciels, y compris pour des entreprises clientes, et compte plus de 15 millions de pages consultées chaque mois.

Bien qu'il génère des revenus grâce à de grosses sociétés clientes, le projet Read the Docs est toujours majoritairement financé par les dons de ses utilisateurs. Le coût des serveurs est quant à lui couvert grâce au mécénat de l'entreprise Rackspace.

Eric et son cofondateur, Anthony Johnson, s'occupent de la maintenance du projet et n'en tirent pas de revenus réguliers bien qu'ils s'y consacrent à temps plein. Une subvention ponctuelle de 48 000 dollars de la Fondation Mozilla[4], reçue en décembre 2015, les aidera à court terme à couvrir leurs salaires. Ils expérimentent actuellement un modèle publicitaire[5] (qui ne piste pas ses utilisateurs) qui rendrait le modèle viable.

Eric remarque que la difficulté ne réside pas uniquement dans le travail de développement, mais aussi dans les fonctions

3. Voir la page pour soutenir Read the Docs : readthedocs.org/sustainability.
4. Voir la page (datée du 10/12/2015) de la Fondation Mozilla relative aux projets qu'elle finance : « Mozilla Open Source Support: First Awards Made ».
5. Voir blog.readthedocs.com/ads-on-read-the-docs.

extérieures au code, comme le support client, qui nécessite qu'un mainteneur soit d'astreinte chaque week-end en cas de problème. Pour expliquer pourquoi il continuait de maintenir le projet, Eric l'a qualifié de « travail-passion » :

> Soit les humains sont irrationnels, soit ils ne sont pas intéressés seulement par l'argent. Il y a clairement une autre motivation pour moi dans ce cas. C'est un travail-passion. Si je le voulais, je pourrais clore ce projet demain et ne plus jamais y revenir, mais je travaille dessus depuis cinq ans et je n'ai aucune envie que ça se termine.

Eric est motivé pour travailler sur Read the Docs parce qu'il perçoit la valeur que cela crée pour les autres. Pour beaucoup de mainteneurs de projets, cet impact sur autrui est la première motivation, parce qu'ils peuvent directement constater l'effet positif de leur travail sur la vie d'autres personnes. En ce sens, l'*open source* a beaucoup de points communs avec le secteur à but non lucratif. Cependant, tout comme dans le secteur à but non lucratif, cette mentalité de « travail-passion » peut augmenter la difficulté qu'ont les communautés *open source* à aborder le sujet qui fâche : comment pérenniser des projets qui nécessitent davantage de ressources et d'attention que n'en peuvent fournir les contributeurs actuels ?

3

DÉFIS À RELEVER

« Si l'*open source* est une incroyable force pour la qualité et pour la communauté, c'est précisément parce qu'elle n'a pas été définie en termes de marché. Dans ce cadre, la plupart des projets *open source* n'auraient jamais eu leur chance. »

David Heinemeier Hansson, Ruby on Rails

LA RELATION COMPLIQUÉE DE L'*OPEN SOURCE* AVEC L'ARGENT

L'argent est un sujet tabou dans les projets *open source*, et ce depuis les premiers jours du mouvement du logiciel libre qui émergea en réponse directe aux pratiques commerciales des logiciels propriétaires. Dans le contexte du mouvement du logiciel libre, l'aversion pour l'argent est tout à fait compréhensible. L'argent est ce qui permettait de commercialiser les logiciels dans les années 1980 et il a fallu des décennies pour revenir sur cet état d'esprit et promouvoir les avantages liés à l'élaboration de logiciels qui soient libres d'utilisation, de distribution et de modification. Même si de nos jours, nous prenons les logiciels libres pour acquis, dans les années 1980, c'était une véritable contre-culture, un état d'esprit révolutionnaire.

Au sein même des communautés *open source*, il existe une croyance répandue selon laquelle l'argent est de nature à corrompre l'*open source*. Et en effet, le nombre de projets nés d'un « travail-passion » est assez incroyable. Aujourd'hui, le développement de logiciel est considéré comme un domaine lucratif, grâce auquel les écoles de programmation appâtent leurs futurs étudiants avec des promesses de premiers salaires en dollars à six chiffres. Par contraste, il y a quelque chose de pur et d'admirable dans le fait de créer un logiciel simplement pour le plaisir.

D'un point de vue plus pratique, les projets *open source* se créent traditionnellement autour d'un besoin réel et identifiable. Quelqu'un estime qu'un projet pourrait être mieux fait, décide de le *forker*, effectue des améliorations, puis les diffuse pour qu'on en fasse usage. Le pragmatisme est au cœur de la culture *open source*, comme le prouve sa scission stratégique avec le mouvement du logiciel libre à la fin des années 1990. Certains contributeurs *open source* craignent, peut-être avec raison, que l'argent n'introduise un développement « artificiel » du

système, avec des développeurs qui lancent de nouveaux projets simplement pour acquérir des financements, plutôt que pour répondre à un besoin réel.

David Heinemeier Hansson (aussi connu sous le pseudo de DHH), qui a créé le *framework* populaire Ruby on Rails, mettait en garde en 2013[1] contre les mélanges entre *open source* et argent :

> Si l'*open source* est une incroyable force pour la qualité et pour la communauté, c'est précisément parce qu'elle n'a pas été définie en termes de marché. Dans ce cadre, la plupart des projets *open source* n'auraient jamais eu leur chance.
>
> Prenez Ruby on Rails. […] C'est une réalisation colossale pour l'humanité ! Des milliers de gens ont collaboré pendant une décennie entière pour produire une structure et un écosystème incroyablement aboutis, disponibles gratuitement pour tous. Prenez une seconde pour méditer sur l'ampleur de cette réussite. Pas seulement pour Rails, évidemment, mais pour de nombreux autres projets *open source*, encore plus grands, avec une filiation plus longue et encore plus de succès.
>
> C'est en considérant ce fantastique succès, dû aux règles de vie d'une communauté, que nous devrions être extraordinairement prudents avant de laisser les lois du marché corrompre l'écosystème.

Structurellement, le meilleur atout de l'*open source*, son penchant pour la démocratie, est aussi sa faiblesse. Beaucoup de projets *open source* ne sont rien de plus qu'un dépôt numérique public où est stocké du code auquel un groupe de personnes contribue régulièrement : l'équivalent d'une association officieuse sur un campus universitaire. Il n'y a pas de structure

1. Voir sur le site de David Heinemeier-Hansson, « The Perils of Mixing Open Source and Money », 12/11/2013.

légale et il n'y a pas de propriétaire ou de chef clairement défini. Les « mainteneurs » ou les contributeurs principaux émergent souvent *de facto*, en fonction de qui a créé le projet ou de qui y a investi beaucoup de temps ou d'efforts. Cependant, même dans ces cas-là, on répugne à introduire dans certains projets une hiérarchie favorisant clairement un contributeur par rapport à un autre.

En avril 2008, Jeff Atwood, un développeur .NET bien connu et dont nous avons déjà parlé[2], a annoncé qu'il donnait 5 000 dollars au projet *open source* ScrewTurn Wiki. Il s'agit d'un projet de wiki développé par Dario Solara, un autre développeur .NET, et maintenu par des volontaires. Atwood a dit à Dario que le don était « sans condition » : Solara pouvait utiliser cette somme de la manière qu'il jugerait la plus utile au projet.

Plusieurs mois plus tard, Atwood demanda à Solara comment il avait décidé de la dépenser. Solara lui répondit que l'argent de la donation était « encore intact. Ce n'est pas facile de l'utiliser… Que suggères-tu ? » Atwood a écrit[3] que cette réponse l'avait « terriblement déçu ».

La nature décentralisée du monde *open source* en a fait ce qu'il est : des logiciels produits de façon participative, que n'importe qui peut élaborer, partager et améliorer. Mais quand vient le moment de discuter des besoins organisationnels ou de la viabilité, il peut être difficile de prendre des décisions faisant autorité.

Ces transitions vers une viabilité à long terme peuvent être interminables et douloureuses. Un des exemples les plus connus est le noyau Linux, un projet *open source* utilisé dans de nombreux systèmes d'exploitation à travers le monde, parmi lesquels Android et Chrome OS. Il a été créé en 1991 par Linus Torvalds, un étudiant en informatique.

Au fur et à mesure que le noyau Linux gagnait en popularité, Linus rechignait à discuter de l'organisation du développement du projet, préférant tout gérer seul. L'inquiétude et aussi la colère à l'égard de Torvalds grandirent chez les développeurs du projet, déclenchant de « vraies grosses disputes » selon ce

2. Voir partie 1, chapitre 2.
3. Voir « Is Money Useless to Open Source Projects? », blog *Codding Horror*, 28/07/2008.

dernier. Le conflit atteignit son apogée en 2002, on évoqua même un possible schisme.

Torvalds attribua ces conflits internes à un manque d'organisation, plutôt qu'à un quelconque problème technique :

> Nous avons eu de vraies grosses disputes aux alentours de 2002, quand j'appliquais des correctifs à droite à gauche et que les choses ne fonctionnaient vraiment pas. C'était très pénible pour tout le monde, y compris pour moi. Personne n'apprécie vraiment les critiques, d'autant moins lorsqu'elles sont virulentes, et comme ce n'était pas un problème strictement technique, on ne pouvait pas juste montrer un correctif et dire : « Hé, regardez, ce patch améliore les performances de 15 % » ou quoi que ce soit de ce genre. Il n'y avait pas de solution technique. L'issue a été d'utiliser de meilleurs outils et d'avoir une organisation du travail qui nous permette de mieux distribuer les tâches.

La Fondation Linux a été créée en 2007 pour aider à protéger et à maintenir Linux et ses projets associés. Torvalds ne pilote pas la Fondation Linux lui-même, il a préféré recevoir un salaire régulier en tant que « Compagnon Linux », et travailler sur ses projets en tant qu'ingénieur.

Malgré le fait que le logiciel *open source* soit admirablement ancré dans une culture du volontariat et de la collaboration relativement préservée de motivations extrinsèques, la réalité est que notre économie et notre société, depuis les sociétés multimillionnaires jusqu'aux sites web gouvernementaux, dépendent de l'*open source*.

Dans l'ensemble, c'est probablement une évolution positive pour la société. Cela signifie que les logiciels ne sont plus limités à un développement privé et propriétaire, comme cela a été le cas pendant des dizaines d'années. Le fait que le gouvernement américain ou un réseau social possédant des milliards d'utilisateurs intègrent des logiciels construits par une communauté, annonce un futur optimiste pour la démocratie.

De plus, de nombreux projets fonctionnent très bien de manière communautaire lorsqu'ils sont d'une des deux tailles

extrêmes possibles, c'est-à-dire soit des petits projets qui ne demandent pas de maintenance significative (comme dans l'exemple de Arash Payan et Appirater[4]), soit de très gros projets qui reçoivent un soutien important de la part d'entreprises (comme Linux).

Cependant, beaucoup de projets sont coincés quelque part entre les deux : assez grands pour nécessiter une maintenance significative, mais pas d'une taille suffisante pour que des entreprises réclament d'offrir leur soutien. Ces projets sont ceux dont l'histoire passe inaperçue, ceux dont on ne parle pas. Des deux côtés, on dit aux développeurs de ces projets « moyens » qu'ils sont le problème : du côté des « petits projets », on pense qu'ils devraient simplement mieux s'organiser et du côté des « gros projets », on pense que si leur projet était « assez bon », il aurait déjà retenu l'attention des soutiens institutionnels.

Il existe aussi des intérêts politiques autour de la question de l'aide financière qui rendent encore plus difficile la prospection d'une source de financement fiable. On peut imaginer qu'une entreprise seule ne souhaite pas sponsoriser le développement d'un travail qui pourrait également bénéficier à son concurrent, qui lui n'aurait rien payé. Un mécène privé peut exiger des privilèges spécifiques qui menacent la neutralité d'un projet. Par exemple, dans les projets en lien avec la sécurité, le fait d'exiger d'être le seul à qui sont révélées les potentielles failles (c'est-à-dire payer pour être le seul à connaître les failles de sécurité plutôt que de les rendre publiques) est un type de requête controversé. Des gouvernements peuvent également avoir des raisons politiques pour financer le développement d'un projet en particulier ou pour demander des faveurs spéciales comme une *backdoor* (une porte dérobée, c'est-à-dire un accès secret qui permet d'outrepasser les authentifications de sécurité), même si le projet est utilisé dans le monde entier.

Les récents démêlés légaux entre le FBI et Apple sont un bon révélateur des tensions qui existent entre technologie et Gouvernement, au-delà même des projets *open source*.

4. Voir partie 1, chapitre 1.

Le FBI a, de manière répétée, et par des assignations en justice, demandé l'aide d'Apple pour déverrouiller des téléphones afin de permettre de résoudre des enquêtes criminelles. Apple a toujours refusé ces requêtes. En février 2016, le FBI a sollicité Apple pour déverrouiller le téléphone d'un des tireurs d'une attaque terroriste récente à San Bernardino, en Californie. Apple s'y est opposé et a publié une lettre sur son site[5], déclarant :

> Tout en croyant que les intentions du FBI sont bonnes, nous pensons qu'il serait mauvais pour le Gouvernement de nous forcer à ajouter une *backdoor* dans nos produits. Et finalement, nous avons peur que cette demande ne mette en danger les libertés que notre Gouvernement est censé protéger.

En mars 2016, le FBI a trouvé une tierce partie pour l'aider à déverrouiller l'iPhone et a laissé tomber l'affaire.

Une des plus grandes forces de l'*open source* est que le code est considéré comme un bien public et beaucoup de projets prennent leur gestion au sérieux. Il est important à titre personnel, pour beaucoup de développeurs de projets, que personne ne puisse prendre seul le contrôle d'une chose que le public utilise et dont il bénéficie. Toutefois, cet engagement à rester neutre a un prix, puisque beaucoup de ressources disponibles pour les développeurs de nos jours (comme les capitaux-risques ou les donations d'entreprises) attendent en contrepartie d'influer sur le projet ou des retours sur investissement.

Le logiciel *open source* est créé et utilisé aujourd'hui à une vitesse jamais vue auparavant. Beaucoup de projets *open source* sont en train d'expérimenter la difficile transition d'une création désintéressée à une infrastructure publique essentielle.

Ces dépendances toujours plus nombreuses signifient que nous avons pour responsabilité partagée de garantir à ces projets le soutien dont ils ont besoin.

5. Voir Apple, « A Message to Our Customers », Apple.com, 16/02/2016.

POURQUOI LES PROBLÈMES DE SUPPORT DES INFRASTRUCTURES NUMÉRIQUES SONT DE PLUS EN PLUS PRESSANTS

L'*open source*, grâce à ses points forts cités plus haut dans cet ouvrage[1], est rapidement en train de devenir un standard pour les projets d'infrastructure numérique et dans le développement logiciel en général. Black Duck, une entreprise qui aide ses clients à gérer des programmes *open source*, dirige une enquête annuelle qui interroge les entreprises sur leur utilisation de l'*open source*. (Cette enquête est l'un des rares projets de banque de données qui existe sur le sujet.) Dans leur étude de 2015[2], 78 % des 1 300 entreprises interrogées déclarent que les logiciels qu'elles ont créés pour leurs clients sont construits grâce à l'*open source*, soit presque le double du chiffre de 2010.

L'*open source* a vu sa popularité s'accroître de manière impressionnante ces cinq dernières années, pas seulement grâce à ses avantages évidents pour les développeurs et les consommateurs, mais également grâce à de nouveaux outils qui rendent la collaboration plus facile. Pour comprendre pourquoi les infrastructures numériques rencontrent des problèmes de support grandissants, nous devons saisir comment le développement de logiciels *open source* prolifère.

1. Voir partie 1, chapitre 2 et chapitre 3.
2. Voir *The Ninth Annual Future of Open Source Survey* sur le site BlackDuck. Les résultats 2016 de cette même étude sont également disponibles.

GitHub, un espace standardisé pour collaborer sur du code

On n'insistera jamais trop sur le rôle clé de GitHub dans la diffusion de l'*open source* auprès du grand public. L'*open source* a beau exister depuis près de trente ans, jusqu'en 2008, contribuer à des projets *open source* n'était pas si facile. Le développeur motivé devait d'abord découvrir qui était le mainteneur du projet, trouver une manière de le contacter, puis proposer ses changements en utilisant le format choisi par le mainteneur (par exemple une liste courriel ou un forum). GitHub a standardisé ces méthodes de communication : les mainteneurs sont listés de façon transparente sur la page du projet, et les discussions sur les changements proposés ont lieu sur la plateforme GitHub.

GitHub a aussi créé un vocabulaire qui est désormais standard parmi les contributeurs à l'*open source*, tel que la *pull request* (où un développeur soumet à l'examen de ses pairs une modification à un projet), et changé le sens du terme *fork* (historiquement, créer une copie d'un projet et le modifier pour le transformer en un nouveau projet ; littéralement, « *fork* » signifie « bifurcation »). Avant GitHub, *forker* un projet revenait à dire qu'il y avait un différend irréconciliable au sujet de la direction qu'un projet devrait prendre. *Forker* était considéré comme une action grave : si un groupe de développeurs décidait de *forker* un projet, cela signifiait qu'il se scindait en deux factions idéologiques. *Forker* était aussi utilisé pour développer un nouveau projet qui pouvait avoir une utilisation radicalement différente du projet initial.

Ce type de « *fork* de projet » existe toujours, mais GitHub a décidé d'utiliser le terme *fork* pour encourager à davantage d'activité sur sa plateforme. Un *fork* GitHub, contrairement à un *fork* de projet, est une copie temporaire d'un projet sur laquelle on effectue des modifications et qui est généralement refusionnée avec le projet. Le *fork* en tant que pratique quotidienne sur GitHub a ajouté une connotation positive, légère au terme : à savoir prendre l'idée de quelqu'un et l'améliorer.

GitHub a aussi aidé à standardiser l'utilisation d'un système de contrôle de version appelé Git. Les systèmes de contrôle de versions sont un outil qui permet de garder une trace de

chaque contribution apportée sur un morceau de code précis. Par exemple, si le développeur 1 et le développeur 2 corrigent simultanément différentes parties du même code, enregistrer chaque changement dans un système de contrôle de version permet de faire en sorte que leurs changements n'entrent pas en conflit.

Il existe plusieurs systèmes de contrôle de versions, par exemple Apache Subversion et Concurrent Versions System (CVS). Avant GitHub, Git était un système de contrôle de version assez méconnu. En 2010, Subversion était utilisé dans 60 % des projets logiciels, contre 11 % pour Git[3].

C'est Linus Torvalds, le créateur de Linux, qui a conçu Git en 2005. Son intention était de mettre à disposition un outil à la fois plus efficace et plus rapide, qui permette de gérer de multiples contributions apportées par de nombreux participants. Git était vraiment différent des systèmes de contrôle de version précédents, et donc pas forcément facile à adopter, mais son *workflow*[4] décentralisé a résolu un vrai problème pour les développeurs.

GitHub a fourni une interface utilisateur intuitive pour les projets *open source* qui recourent à Git, ce qui rend l'apprentissage plus facile pour les développeurs. Plus les développeurs utilisent GitHub, plus cela les incite à continuer d'utiliser Git. En 2016, Git est utilisé par 38 % des projets de logiciels, tandis que la part de Subversion est tombée à 47 %[5]. Bien que Subversion soit encore le système de contrôle de version le plus populaire, son usage décline. L'adoption généralisée de Git rend plus facile pour un développeur la démarche de se joindre à un projet sur GitHub, car la méthode pour faire des modifications et pour les communiquer est la même pour tous les projets. Apprendre à contribuer à un seul des projets vous permet d'acquérir les compétences pour contribuer à des centaines d'autres. Ce n'était pas le cas avant GitHub, où des systèmes de contrôle de versions différents étaient utilisés pour chaque projet.

3. Voir Stephen O'Grady, « DVCS and Git Usage in 2013 », *RedMonk*, 19/12/2013.
4. Le *workflow* est un moyen de représenter un flux de travail. Pour en savoir plus, voir l'article « Workflow » sur Wikipédia.
5. Voir les données sur BlackDuck.com. Les données du site sont actualisées régulièrement. Les chiffres mentionnés ont été consultés le 06/01/2016. Au 16/01/2017, la part de Git est passée à 40 % et celle de Subversion à 46 %.

Enfin, GitHub a créé un espace de sociabilité qui permet de discuter et de tisser des liens au-delà de la stricte collaboration sur du code. La plateforme est devenue *de facto* une sorte de communauté pour les développeurs, qui l'utilisent pour communiquer ensemble et exposer leur travail. Ils peuvent y démontrer leur influence et présenter un portfolio de leur travail comme jamais auparavant.

Les usages de GitHub sont un reflet de son ascension vertigineuse. En 2011[6], il n'y avait que 2 millions de dépôts (*repository*). Aujourd'hui, GitHub a 14 millions d'utilisateurs et plus de 35 millions de dépôts[7] (ce qui inclut aussi les dépôts forkés, le compte des dépôts uniques s'élève plutôt à 17 millions). Brian Doll, de chez GitHub, a noté qu'il a fallu quatre ans pour atteindre le million de dépôts, mais que passer de 9 millions à 10 millions n'a pris que quarante-huit jours[8].

En comparaison, SourceForge, la plateforme qui était la plus populaire pour héberger du code *open source* avant l'apparition de GitHub, avait 150 000 projets en 2008. Environ 18 000 d'entre eux étaient actifs[9].

Stack Overflow, un espace standard pour s'entraider sur du code

L'une des autres plateformes importantes de l'*open source* est Stack Overflow, un site de questions/réponses populaire parmi les développeurs, créé en 2008 par Jeff Atwood (développeur déjà mentionné précédemment[10]) et par le blogueur Joel Spolsky. En avril 2014, Stack Overflow avait plus de 4 millions d'utilisateurs enregistrés et plus de 11 millions de questions résolues[11] (à noter qu'il n'est pas nécessaire de s'enregistrer pour voir les questions ou leurs réponses).

6. Voir GitHub, « Those are some big numbers », sur le blog de GitHub, 20/04/2011.
7. Chiffres issus de l'article « GitHub » sur Wikipédia.
8. Voir GitHub, « 10 Million Repositories » sur le blog de Github, 23/12/2013.
9. Amit Deshpande et Dirk Riehle, « The Total Growth of Open Source », *Software Research and the Industry*, Proceedings of the Fourth Conference on Open Source Systems, 2008.
10. Voir partie 3, chapitre 1 et partie 1, chapitre 2.
11. Chiffres issus de l'article « Stack Overflow » sur Wikipédia.

Stack Overflow est devenu *de facto* une plateforme d'entraide pour les développeurs, qui peuvent poser des questions de programmation, trouver des réponses à des problèmes de code spécifiques ou juste échanger des conseils sur la meilleure façon de créer un aspect précis d'un logiciel. On pourrait définir la plateforme comme un « support client » participatif pour les développeurs à travers le monde. Même si Stack Overflow n'est pas un endroit où l'on écrit directement du code, c'est un outil de collaboration essentiel pour les développeurs individuels, qui facilite grandement la résolution de problèmes et permet de coder plus efficacement. Cela signifie qu'un développeur individuel est capable de produire plus, en moins de temps, ce qui améliore le rendement global. Stack Overflow a également permis à certains utilisateurs d'apprendre de nouveaux concepts de développement (ou même de s'initier au code tout court) et a rendu le codage plus facile et plus accessible à tous.

Tendances macro dans un paysage en mutation constante

La popularité hors normes de l'*open source* a amené à des changements significatifs dans la manière dont les développeurs d'aujourd'hui parlent, pensent et collaborent pour des logiciels.

Premièrement, les attentes et exigences en termes de licence ont changé, reflétant un monde qui considère désormais l'*open source* comme une norme, et pas l'exception : un triomphe sur l'univers propriétaire des années 1980. Les politiques de GitHub et de Stack Overflow reflètent toutes deux cette réalité.

Dès le départ, Stack Overflow a choisi d'utiliser une licence Creative Commons de type CC-BY-SA[12] pour tous les contenus postés sur son site. La licence était cependant limitante, car elle requérait des utilisateurs qu'ils mentionnent l'auteur de chaque morceau de code utilisé, et qu'ils placent leurs propres contributions sous la même licence.

Beaucoup d'utilisateurs choisissaient d'ignorer la licence ou n'étaient même pas au courant de ses restrictions, mais pour

12. Voir le Contenu de la licence CC-BY-SA sur creativecommons.org.

les développeurs travaillant avec des contraintes plus strictes (par exemple dans le cadre d'une entreprise), elle rendait Stack Overflow compliqué à utiliser. S'ils postaient une question demandant de l'aide pour leur code, et qu'une personne extérieure réglait le problème, alors légalement, ils devaient attribuer le code à cette personne.

En conséquence, les dirigeants de Stack Overflow ont annoncé leur volonté de déplacer toutes les nouvelles contributions de code sous la licence MIT[13], qui est une licence *open source* comportant moins de restrictions[14]. En avril 2016, ils débattent encore activement et sollicitent des retours de leur communauté pour déterminer le meilleur moyen de mettre en œuvre un système plus permissif. Cette démarche est un encouragement à la fois pour la popularité de Stack Overflow et pour la prolifération de l'*open source* en général. Qu'un développeur travaillant dans une grosse entreprise de logiciel puisse légalement inclure le code d'une personne complètement extérieure dans un produit pour lequel il est rémunéré est en effet un accomplissement pour l'*open source*.

À l'inverse, GitHub fit initialement le choix de ne pas attribuer de licence par défaut aux projets postés sur sa plateforme, peut-être par crainte que cela ne freine son adoption par les utilisateurs et sa croissance[15]. Ainsi, les projets postés sur GitHub accordent le droit de consulter et de *forker* le projet, mais sont à part ça sous copyright, sauf si le développeur spécifie qu'il s'agit d'une licence *open source*.

En 2013, GitHub décida enfin de prendre davantage position sur la question des licences, avec notamment la création et la promotion d'un microsite, choosealicense.com[16], pour aider les utilisateurs à choisir une licence pour leur projet. Ils encouragent aussi désormais leurs utilisateurs à choisir une licence parmi une liste d'options au moment de créer un nouveau dépôt[17].

13. Voir The MIT License – Clarity on Using Code on Stack Overflow and Stack Exchange sur meta.stackexchange.com, 14/01/2016.
14. « A New Code License: The MIT, this time with Attribution Required », sur meta.stackexchange.com, 15/01/2016.
15. Simon Philipps, « GitHub needs to take open source seriously », *Infoworld*, 30/11/2012.
16. Voir le site Choosealicense.com.
17. *Idem*.

Ce qui est intéressant, cependant, c'est que la plupart des développeurs ne se préoccupaient pas de la question des licences : soit ils ignoraient que leurs projets *open source* n'étaient pas légalement protégés, soit ils n'en tenaient pas compte. Une étude informelle réalisée en 2013 par le Software Freedom Law Center (Centre du droit de la liberté des logiciels) sur un échantillon de 1,6 million de dépôts GitHub révéla que seuls 15 % d'entre eux avaient spécifié une licence[18]. Aussi, les entretiens avec des développeurs réalisés pour cette étude suggèrent que beaucoup se fichent de spécifier une licence, ou se disent que si elle leur est exigée, ils pourront toujours en ajouter une ultérieurement.

Ce manque d'intérêt pour les licences a amené James Governor, cofondateur de la firme d'analyse de développeurs Red Monk, à constater en 2012 que « les jeunes dévs aujourd'hui font du POSS – Post open source software[19] –, envoient chier les licences et la gestion, contribuent juste à GitHub ». En d'autres termes, faire de l'information ouverte par défaut est devenu une telle évidence culturelle aujourd'hui que les développeurs ne s'imaginent plus faire les choses autrement – un contexte bien différent de celui des rebelles politisés du logiciel libre des années 1980. Ce retournement des valeurs, quoique inspirant au niveau global, peut cependant amener à des complications légales pour les individus quand leurs projets gagnent en popularité ou sont utilisés à des fins commerciales.

Mais, en rendant le travail collaboratif sur le code aussi facile et standardisé, l'*open source* se retrouve aux prises avec une série d'externalités perverses.

L'*open source* a rendu le codage plus facile et plus accessible au monde. Cette accessibilité accrue, à son tour, a engendré une nouvelle catégorie de développeurs, moins expérimentés, mais qui savent comment utiliser les composants préfabriqués par d'autres pour construire ce dont ils ont besoin.

En 2012, Jeff Atwood, cofondateur de Stack Overflow, rédigea un article de blog intitulé ironiquement « Pitié, n'apprenez

18. Voir Neil McAllister, « Study: Most projects on GitHub not open source licensed », *The Register*, 18/04/2013.
19. Déclaration issue d'un tweet du 17/09/2012. *Dévs* est une abréviation de développeur. PPOS pour Post Open Source Software soit Post logiciel *open source*.

pas à coder », où il se plaint de la mode des stages et des écoles de code. Tout en se félicitant du désir des personnes non techniciennes de comprendre le code d'un point de vue conceptuel, Atwood émet des réserves[20] et se demande si « introduire parmi la main-d'œuvre ces codeurs naïfs, novices, voire même-pas-vraiment-sûrs-d'aimer-ce-truc-de-programmeur, a vraiment des effets positifs pour le monde ».

Dans ces circonstances, le modèle de développement de l'*open source* change de visage. Avant l'ascension de GitHub, il y avait moins de projets *open source*. Les développeurs formaient donc un groupe plus restreint, mais en moyenne plus expérimenté : ceux qui utilisaient du code partagé par d'autres étaient susceptibles d'être également ceux qui contribuent en retour.

Aujourd'hui, l'intense développement de l'éducation au code implique que de nombreux développeurs inexpérimentés inondent le marché. Cette dernière génération de développeurs novices emprunte du code libre pour écrire ce dont elle a besoin, mais elle est rarement capable, en retour, d'apporter des contributions substantielles aux projets. Beaucoup sont également habitués à se considérer comme des « utilisateurs » de projets *open source*, davantage que comme les membres d'une communauté. Les outils *open source* étant désormais plus standardisés et faciles à utiliser, il est bien plus simple aujourd'hui pour un néophyte de débarquer sur un forum GitHub et d'y faire un commentaire désobligeant ou une requête exigeante – ce qui épuise et exaspère les mainteneurs.

Cette évolution démographique a aussi conduit à un réseau de logiciels bien plus fragmenté, avec de nombreux développeurs qui publient de nouveaux projets et qui créent un réseau embrouillé d'interdépendances. Se qualifiant lui-même de « développeur-pie en rémission »[21], Drew Hamlett a écrit en janvier 2016 un post de blog devenu très populaire intitulé « Le triste état du développement web »[22]. L'article traite de

20. « Please Don't Learn to Code », *Coding Horror*, 15/05/2012.
21. Voir partie 2, chapitre 1 et chapitre 3. « Pie » est un surnom pour les développeurs opportunistes, d'après le nom de l'oiseau, la pie réputée voleuse (NdT).
22. Drew Hamlett, « The Sad State of Web Development », *Medium.com*, 10/01/2016.

l'évolution du développement web, se référant spécifiquement à l'écosystème Node.js :

> Les individus qui sont restés dans la communauté Node ont sans aucun doute créé l'écosystème le plus techniquement compliqué qui ait jamais existé. Personne n'arrive à y créer une bibliothèque qui fasse quoi que ce soit. Chaque projet qui émerge est encore plus ambitieux que le précédent… mais personne ne construit rien qui fonctionne concrètement. Je ne comprends vraiment pas. La seule explication que j'ai trouvée, c'est que les développeurs sont juste continuellement en train d'écrire et de réécrire en boucle des applis Node.js.

Aujourd'hui, il y a tellement de projets qui sont élaborés et publiés qu'il est tout simplement impossible pour chacun d'eux de développer une communauté suffisamment importante et viable, avec des contributeurs réguliers qui discuteraient avec passion des modifications à apporter lors de débats approfondis sur des listes courriels. Au lieu de cela, beaucoup de projets sont maintenus par une ou deux personnes seulement, alors même que la demande des utilisateurs pour ces projets peut excéder le travail nécessaire à leur simple maintenance.

GitHub a rendu simples la création et la contribution à de nouveaux projets. Cela a été une bénédiction pour l'écosystème *open source*, car les projets se développent plus rapidement. Mais cela peut aussi parfois tourner à la malédiction pour les mainteneurs de projets, car davantage de personnes peuvent facilement signaler des problèmes ou réclamer de nouvelles fonctionnalités, sans pour autant contribuer elles-mêmes en retour. Ces interactions superficielles ne font qu'alourdir la charge de travail des mainteneurs, dont on attend qu'ils répondent à une quantité croissante de requêtes.

Il ne serait pas déraisonnable d'affirmer qu'un monde « post-*open source* » implique une réflexion non seulement autour des licences, ainsi que James Governor l'exprimait dans son commentaire originel, mais aussi autour du processus de développement lui-même.

Noah Kantrowitz, développeur Python de longue date et membre de la Python Software Foundation, a résumé ce changement dans un post de blog[23] souvent cité :

> Dans les débuts du mouvement *open source*, il y avait assez peu de projets, et en général, la plupart des gens qui utilisaient un projet y contribuaient en retour d'une façon ou d'une autre. Ces deux choses ont changé à un point difficilement mesurable.
>
> […] Alors même que nous allons de plus en plus vers des outils de niche, il devient compliqué de justifier l'investissement en temps requis pour devenir contributeur. « Combler son propre besoin » est toujours une excellente motivation, mais il est difficile de construire un écosystème là-dessus.
>
> L'autre problème est le déséquilibre de plus en plus important entre producteurs et consommateurs. Avant, cela s'équilibrait à peu près. Tout le monde investissait du temps et des efforts dans les Communs et tout le monde en récoltait les bénéfices. Ces temps-ci, très peu de personnes font cet effort et la grande majorité ne fait que bénéficier du travail de ceux qui s'impliquent.
>
> Ce déséquilibre s'est tellement enraciné qu'il est presque impensable pour une entreprise de rendre (en temps ou en argent) ne serait-ce qu'une petite fraction de la valeur qu'elle tire des Communs.

Cela ne veut pas dire qu'il n'existe plus de grands projets *open source* avec des communautés de contributeurs fortes (Node.js, dont on parlera plus tard, est un exemple de projet qui est parvenu à ce statut). Cela signifie qu'à côté de ces réussites, il y a une nouvelle catégorie de projets qui est défavorisée par les normes et les attentes actuelles de l'*open source* et que

23. Noah Kantrowitz, « Funding FOSS », coderanger.net, 01/07/2015.

le comportement qui dérive de ces nouvelles normes affecte même des projets plus importants et plus anciens.

Hynek Schlawack, *fellow*[24] de la Python Software Foundation et contributeur à des projets d'infrastructure Python, exprime[25] ses craintes au sujet d'un futur où il y aurait une demande plus forte, mais seulement une poignée de contributeurs solides :

> Ce qui me frustre le plus, c'est que nous n'avons jamais eu autant de développeurs Python et aussi peu de contributions de haute qualité. Dès que des développeurs clés comme Armin Ronacher ralentissent leur travail, la communauté tout entière le ressent aussitôt. Le jour où Paul Kehrer arrêtera de travailler sur PyCA, on sera très mal. Si Hawkowl interrompt son travail de portage, Twisted ne sera jamais sur Python 3 et Git.
>
> La communauté est en train de se faire saigner par des personnes qui créent plus de travail qu'elles n'en fournissent. […] En ce moment, tout le monde bénéficie de ce qui a été construit, mais la situation se détériore à cause du manque de financements et de contributions. Ça m'inquiète, parce Python est peut-être très populaire aujourd'hui, mais une fois que les conséquences se feront sentir, les opportunistes partiront aussi vite qu'ils étaient arrivés.

Pour la plupart des développeurs, il n'y a guère que cinq ans peut-être que l'*open source* est devenue populaire. La large communauté des concepteurs de logiciel débat rarement de la pérennité à long terme de l'*open source*, et n'a parfois même pas conscience du problème. Avec l'explosion du nombre de nouveaux développeurs qui utilisent du code partagé sans contribuer en retour, nous construisons des palaces sur une infrastructure en ruines.

24. Le terme *fellow* est intraduisible sans longue périphrase, pour en savoir plus, voir l'article « Fellow » sur Wikipédia.
25. Source : entretien par mail avec l'auteur.

NÉGLIGER LES INFRASTRUCTURES A UN COÛT CACHÉ

Comme nous l'avons vu, l'infrastructure numérique est un constituant essentiel du monde actuel. Notre société repose sur les logiciels et ces logiciels s'appuient de plus en plus sur une infrastructure qui utilise des méthodologies *open source*. Dans la mesure où nous prenons peu d'initiatives pour comprendre et pérenniser notre infrastructure numérique, que mettons-nous en péril ?

Ne pas réinvestir dans l'infrastructure numérique présente des dangers que l'on peut classer en deux catégories :

- **les coûts directs :** les coûts directs sont les bugs non détectés et les vulnérabilités de sécurité qui peuvent être exploitées à des fins malveillantes ou qui mènent à des interruptions imprévues dans le fonctionnement des logiciels. Ces coûts sont fortement ressentis et causent des problèmes qui doivent être résolus immédiatement.
- **les coûts indirects :** les coûts indirects se traduisent par exemple par la perte de main-d'œuvre qualifiée, ainsi que par une croissance faible et peu d'innovation. Même si ces coûts ne sont pas immédiatement perceptibles, ils représentent une valeur sociale difficile à évaluer.

Bugs, failles de sécurité et interruptions du service

L'introduction de ce rapport relatait les détails de la faille de sécurité Heartbleed, qui a été découverte en avril 2014 dans une bibliothèque logicielle appelée OpenSSL. Du fait de son usage généralisé, notamment pour le fonctionnement de nombreux sites web majeurs, Heartbleed a largement attiré

l'attention du public sur le problème des failles de sécurité des logiciels.

En septembre 2014, une autre faille majeure a été découverte dans un autre outil essentiel appelé Bash. Bash est inclus dans des systèmes d'exploitation populaires tels que Linux et Mac OS, ce qui fait qu'il est installé sur 70 % des machines connectées à Internet[1].

L'ensemble des bugs de sécurité, surnommé « ShellShock », peut être exploité pour permettre un accès non autorisé à un système informatique. Les vulnérabilités étaient restées non détectées pendant au moins une décennie. Bash a été créé en 1987 par un développeur appelé Brian Fox, mais depuis 1992, il est maintenu par un seul développeur, Chet Ramey, qui travaille comme architecte technique senior à la Case Western University dans l'Ohio.

Un autre projet, OpenSSH, fournit une suite d'outils de sécurité dont l'usage est largement répandu sur Internet. Des développeurs ont trouvé de multiples failles dans son code qui ont pu être prises en charge et corrigées, y compris celle de juillet 2015, qui permettait aux attaquants de dépasser le nombre maximal d'essais sur les mots de passe[2], et celle de janvier 2016, qui laissait fuiter les clés de sécurité privées[3].

L'un des aspects du problème est que beaucoup de projets sont des outils anciens, conçus au départ par un ou plusieurs développeurs passionnés, qui ont par la suite manqué de ressources pour gérer le succès de leur projet. Avec le temps, les contributions diminuent et les acteurs restants se lassent et s'en vont, mais pour autant le projet est toujours activement utilisé, avec seulement une ou deux personnes qui tâchent de le maintenir en vie.

Un autre problème croissant dans le paysage des logiciels actuel, où l'on voit tant de jeunes développeurs inexpérimentés, c'est que les concepts de sécurisation ne sont pas enseignés ou pas considérés comme prioritaires. Les nouveaux développeurs

1. Nicole Perlroth, « Security experts expect "Shellshock" software bug to be significant », *Gadgets Now*, 28/09/2014.
2. Doug Drinkwater, « OpenSSH flaw opens the door to brute force attackers », *SC Media*, 24/07/2015.
3. Dan Goodin, « Bug that can leak crypto keys just fixed in widely used OpenSSH », *Ars Technica*, 01/04/2016.

veulent simplement écrire du code qui marche. Ils ne savent pas faire un logiciel sécurisé ou pensent à tort que le code public qu'ils utilisent dans la sécurité de leurs programmes a été vérifié. Même les bonnes pratiques de divulgation sécurisée ou de gestion des failles ne sont généralement pas enseignées ni comprises. La sécurité ne devient un enjeu que lorsque le code d'un développeur a été compromis.

Christopher Allen a coécrit la première version du protocole de transfert sécurisé TLS (Transport Layer Security), dont les versions successives sont devenues un standard utilisé quasiment universellement en ligne, y compris sur des sites comme Google, Facebook ou YouTube. Bien que le protocole soit devenu un standard, Christopher parle ainsi de ses origines[4] :

> En tant que coauteur de TLS, je n'aurais pas prédit que quinze ans plus tard la moitié d'Internet utiliserait une implémentation de TLS maintenue par un ingénieur à quart-temps. C'est ce manque de maintenance qui a conduit au bug tristement célèbre de Heartbleed. Je raconte aujourd'hui cette anecdote à mes collègues qui travaillent sur les crypto-monnaies pour les avertir que leur chiffrage, ultra moderne aujourd'hui, pourrait être « dépassé » dans dix ans et subir le même sort, le projet n'étant plus aussi passionnant, et leur travail acharné risquerait d'être compromis.

En définitive, la stabilité de nos logiciels repose sur la bonne volonté et la coopération de centaines de développeurs, ce qui représente un risque significatif. La fragilité de notre infrastructure numérique a récemment été démontrée par un développeur nommé Azer Koçulu.

Azer, un développeur Node.js, hébergeait un certain nombre de bibliothèques dans un gestionnaire de paquets nommé npm[5]. Après un conflit avec npm sur la propriété intellectuelle d'un de ses projets, Azer, mécontent du dénouement, décida de supprimer toutes les publications qu'il avait pu faire sur npm.

4. Christopher Allen, article sur Medium.com, 16/01/2016.
5. Azer Koçulu, « I've Just Liberated My Modules », Medium.com, 22/03/2016.

L'une de ces bibliothèques, left-pad, avait été réutilisée dans des centaines d'autres projets. Même s'il ne s'agissait que de quelques lignes de code, en supprimant le projet left-pad, Azer a « cassé » les algorithmes d'innombrables protocoles logiciels développés par d'autres. La décision d'Azer a provoqué tant de problèmes que npm a pris la décision sans précédent de republier sa bibliothèque, contre la volonté d'Azer, afin de restaurer les fonctionnalités offertes par le reste de l'écosystème[6].

Npm a aussi revu sa politique pour qu'il soit plus difficile pour les développeurs de retirer leurs bibliothèques sans avertissement, reconnaissant ainsi que les actions d'un individu peuvent en affecter négativement beaucoup d'autres[7].

Les logiciels ne reçoivent pas la maintenance nécessaire dont ils ont besoin

Construire une infrastructure numérique de façon désorganisée implique que tout logiciel sera construit plus lentement et moins efficacement. L'histoire de l'infrastructure Python en fournit un bon exemple.

L'un des importants projets d'infrastructure pour les développeurs Python se nomme Setuptools. Setuptools fournit un ensemble d'outils qui rendent le développement en Python plus simple et plus standardisé.

Setuptools a été écrit en 2004, par le développeur P.J. Eby. Pendant les quatre années qui ont suivi, l'outil a été largement adopté. Néanmoins, Setuptools était difficile à installer et à utiliser, et Eby était très peu réceptif aux contributions et aux corrections apportées par d'autres, car il voulait, en tant que concepteur, avoir le dernier mot sur Setuptools. En 2008, un groupe de développeurs conduits par Tarek Ziade a décidé de *forker* le projet pour obliger Eby à faire des améliorations. Ils ont appelé le nouveau projet Distribute. En 2013, les deux projets ont fusionné dans Setuptools.

6. Chris Williams, « How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript », *The Register*, 23/03/2016.
7. Npm, « kik, left-pad, and npm », *The Npm Blog*, 23/03/2016.

Ce long désaccord a néanmoins souligné à la fois l'état douteux des outils de l'infrastructure de Python, et la difficulté de mettre en œuvre des améliorations, notamment parce que personne ne se consacrait aux problèmes de la communauté ni ne désirait s'en occuper.

Les outils de Python ont commencé à s'améliorer quand le groupe de travail Python Packaging Authority (PyPA) s'est formé pour se consacrer spécifiquement à définir de meilleurs standards pour le paquetage. Un développeur, Donald Stufft, fit des outils de paquetage Python le cœur de son travail et fut engagé par HP (devenu HPE) en mai 2015 pour poursuivre son travail (son parcours sera évoqué plus tard dans cet ouvrage[8]).

Un autre exemple intéressant est celui de RubyGems.org, un site web utilisé par la plupart des développeurs Ruby pour héberger leurs bibliothèques Ruby. Ruby a été utilisé pour bâtir des sites web majeurs comme Twitter, Airbnb, YellowPages et GitHub lui-même. En 2013, une faille de sécurité de RubyGems.org a été découverte, mais elle ne fut pas réparée avant plusieurs jours, parce que RubyGems.org était entièrement maintenue par des bénévoles. Les bénévoles pensaient régler le problème le week-end, mais pendant ce temps, quelqu'un d'autre a découvert la faille et a piraté le serveur de RubyGems.org. Après l'attaque, les serveurs ont dû être entièrement reconfigurés. Plusieurs bénévoles ont pris sur leur temps de travail, et certains ont même pris des jours de congé, afin de remettre RubyGems.org en état de marche le plus vite possible.

Comme RubyGems.org est un élément d'infrastructure critique, la faille de sécurité affectait par rebond beaucoup de développeurs et d'entreprises. L'incident a mis en lumière le fait qu'un travail fondé uniquement sur la base du volontariat limitait les garanties de sécurité et de fiabilité que l'on pouvait offrir sur une infrastructure logicielle importante. Des dizaines de développeurs se mobilisèrent de façon « bénévole » pendant l'incident parce que le problème affectait directement leur emploi salarié.

8. Source: Entretiens par courriel avec Russell Keith-Magee et Hynek Schlawac.

Malheureusement, aucun d'entre eux n'avait l'expérience requise pour être utile, et aucun d'entre eux n'a continué à offrir son aide une fois les serveurs réparés. En 2015, une organisation nommée Ruby Together a été formée pour aider à financer la maintenance et le développement de l'infrastructure Ruby, entre autres RubyGems.org, en sollicitant des entreprises comme sponsors[9].

La perte de main-d'œuvre qualifiée

Comme dans beaucoup de communautés de bénévoles, le *burn-out* est courant parmi les contributeurs *open source*, qui se retrouvent à répondre aux requêtes d'utilisateurs individuels ou d'entreprises, pour un travail sans compensation. Beaucoup de développeurs ont des anecdotes où des entreprises les sollicitaient pour du travail gratuit. Daniel Roy Greenfield, développeur Django et Python, a écrit[10] :

> J'ai personnellement eu des demandes pour du travail non payé (les discussions pour payer le travail n'aboutissent jamais) par des entreprises aux profits considérables, grandes ou petites, pour mes projets. Si je ne réponds pas dans les temps convenus, si je n'accepte pas une *pull request* merdique, on va me coller une étiquette de connard. Il n'y a rien de pire que d'être face à des développeurs du noyau Python/PyPA travaillant pour Red Hat (sic), qui exigent de toi un travail non payé tout en critiquant ce qu'ils considèrent comme les insuffisances de ton projet, pour te pourrir ta journée et plomber ta foi en l'*open source*.

(Red Hat est une multinationale du logiciel avec un revenu annuel excédant les 2 milliards d'euros[11], qui vend des logiciels *open source* à des clients d'entreprise. Du fait de la nature de leur

9. Source : Entretiens par courriel et au téléphone avec André Arko. Merci à lui.
10. Commentaire de pydanny sur *Paying the piper* (projet GitHub), 16/10/2015.
11. Barb Darrow, « Red Hat Is Now a $2 Billion Open-Source Baby », *Fortune*, 23/03/2016.

entreprise, les employés de Red Hat utilisent et contribuent à de nombreux projets *open source* : en un sens, Red Hat est devenu la tête d'affiche de l'*open source* dans le monde de l'entreprise. Nous reparlerons de son succès financier plus tard dans cet ouvrage.)

Read the Docs, service d'hébergement de documentation technique précédemment mentionné[12], annonce explicitement sur son site qu'il ne s'occupe pas de l'installation personnalisée dans les entreprises ou chez les particuliers[13].

L'un des mainteneurs, Eric Holscher, va jusqu'à faire ce commentaire[14] : « Je suis à peu près sûr que Read the Docs n'a aucun intérêt à être *open source*, vu que les utilisateurs privés ne contribuent jamais en retour, et se contentent de demander une assistance gratuite. »

Marquess, le contributeur OpenSSL, a tenu un discours acerbe à propos des requêtes récurrentes sur ses posts qui parlent de financement[15] :

> C'est à vous que je pense, entreprises du Fortune 1000. Vous qui incluez OpenSSL dans vos *firewall*/dispositifs/*cloud*/produits financiers ou de sécurité, tous ces produits que vous vendez à profit, ou que vous utilisez pour vos infrastructures et communications internes. Vous qui n'avez pas besoin de financer une équipe interne de programmeurs pour se débattre avec du code crypté, puis qui nous harcelez pour obtenir une assistance gratuite quand vous réalisez que vous êtes incapables de l'utiliser. Vous qui n'avez jamais levé le petit doigt pour contribuer à la communauté *open source* qui vous a fait ce cadeau. Les concernés se reconnaîtront. Certains développeurs choisissent d'arrêter de maintenir leurs projets parce qu'ils n'ont plus assez de temps à y consacrer, et espèrent que quelqu'un d'autre prendra le relais. Pendant ce temps,

12. Voir partie 2, chapitre 3.
13. Read the Docs, *Read the Docs Open Source Philosophy*.
14. Eric Holscher, Tweet, sur Twitter.com, datée du 19/01/2016.
15. Stevem, « Of Money, Responsibility, and Pride », *Speeds and Feeds*, 12/04/2014.

> les entreprises, les gouvernements et les individus dépendent de ces bibliothèques pour leur bon fonctionnement, inconscients des enjeux sous-jacents.

David Michael Ross, ingénieur manager dans une agence web, a écrit au sujet de son expérience[16] :

> Pour moi, c'est là que le bât blesse. […] On sait qu'on a créé quelque chose gratuitement, par passion, et on voit ce flux infini de personnes qui crient « plus ! encore plus ! » et qui se mettent en colère quand on ne traite pas leur cas particulier. Il y avait mon numéro de téléphone sur l'un de mes sites personnels pour que mes amis puissent me joindre. Je l'ai enlevé au bout d'une semaine parce que des gens m'appelaient à toute heure de la journée pour de l'assistance sur les *plugins*, alors qu'il y a un forum consacré à ça. Il n'y a rien de fondamentalement méchant là-dedans, c'est juste que c'est usant. On se met à avoir peur de vérifier ses mails ou de répondre au téléphone.

Ryan Bigg, qui écrit de la documentation technique pour le *framework* Ruby on Rails, a annoncé en novembre 2015 qu'il renonçait à tout travail *open source*[17] :

> Je n'ai plus le temps ni l'énergie de m'investir dans l'*open source*. Je ne retire strictement aucun revenu de mon travail *open source*, ce qui veut dire que le travail que je fais là, c'est du temps que je pourrais consacrer à des activités perso, ou à écrire. Ce n'est pas justifié d'attendre de moi que je travaille encore, en dehors de mon emploi salarié, sans que je sois honnêtement rétribué pour ça (en temps ou en argent). C'est aussi une recette qui a de bonnes chances de me conduire au *burn-out* ou de me rendre juste complètement aigri.

16. Commentaire de csixty4 sur Hacker News, sur le fil « The reason people burn out on open source ».
17. Ryan Bigg, « Open Source Work », site personnel Ryanbigg.com, 16/11/2015.

Par ailleurs, la perte de main-d'œuvre qualifiée dans l'*open source*, ce n'est pas seulement les contributeurs qui démissionnent, c'est aussi ceux qui n'ont jamais contribué du tout.

Il existe très peu de statistiques sur la démographie des contributeurs *open source*, ce qui est déjà révélateur en soi. Une analyse récente de GitHub a montré que seulement 5,4 % des contributeurs *open source* étaient des femmes[18], qui occupent pourtant environ 15 à 20 % des postes techniques dans l'ensemble des entreprises de logiciels.

L'une des raisons qui fait que les contributeurs *open source* sont un groupe remarquablement plus homogène que le secteur de la technologie dans son ensemble, c'est qu'ils ont besoin de temps et d'argent pour apporter tout d'abord des contributions significatives. Ces contraintes empêchent des contributeurs par ailleurs qualifiés d'entrer dans cet espace.

David MacIver, créateur de Hypothésis, une bibliothèque Python qui sert à tester des applications logicielles, explique[19] pourquoi il a pu passer autant de temps sur le projet :

> J'ai pu le faire seulement parce que j'avais le temps et l'argent pour le faire. J'avais le temps parce que j'étais obsessionnel, je n'avais personne à charge et je n'avais pas d'emploi. Je pouvais me permettre de ne pas avoir d'emploi parce que j'avais de l'argent. J'avais de l'argent parce que pendant la dernière moitié de l'année passée, je touchais un salaire deux fois plus élevé que d'habitude, en dépensant deux fois moins que d'habitude et je traversais une dépression qui me rendait trop « *borderline* » pour avoir envie de dépenser mon argent dans quoi que ce soit d'intéressant. Ce ne sont pas des conditions qu'on peut raisonnablement exiger de quelqu'un. […] Est-ce qu'on pourrait produire un logiciel de qualité en moins de temps que ça, en ne travaillant que sur du temps libre ? J'en doute.

18. 5,4 %, c'est le pourcentage de contributrices ayant soumis plus de 10 contributions. Voir Breanden Beneschott, « Is Open Source Open to Women? », *Toptal*, septembre 2015.
19. David R. MacIver, « It's OK for your open source library to be a bit shitty », 08/04/2015.

Cory Benfield, un développeur pour les fonctions de base de Python, écrit[20] :

> De manière générale, les personnes qui ne sont pas des hommes cisgenres[21], hétérosexuels, blancs, de classe moyenne et anglophones sont moins susceptibles de pouvoir assumer les risques financiers accrus associés à l'absence d'emploi stable. Cela signifie que ces personnes ont vraiment besoin d'un salaire régulier pour pouvoir contribuer le plus efficacement possible. Et nous avons *besoin* de leur contribution : des équipes diversifiées font un meilleur travail que des équipes homogènes.

Charlotte Spencer, qui contribue au *framework* logiciel Hoodie et au système de bases de données PouchDB, fait écho à cette opinion[22] :

> Toutes mes contributions sont purement bénévoles. Je n'en retire pas d'argent, même si j'aimerais beaucoup pouvoir le faire. J'ai demandé à des vétérans de l'*open source* s'ils étaient payés et ce n'est pas le cas, ce qui m'a découragé d'essayer quoi que ce soit (si ces personnes-là ne sont pas payées, pourquoi le serais-je ?). J'y consacre la plus grande partie de mon temps libre, mais j'essaie d'en faire moins parce que ça envahissait trop ma vie.

Jessica Lord, développeuse, a contribué activement à l'*open source* tout en travaillant à Code for America, une organisation à but non lucratif qui soutient la technologie dans le secteur public. Urbaniste de formation, elle insiste sur le fait[23] qu'elle n'avait « pas de diplôme en informatique, pas d'expérience formelle en programmation, mais un portfolio GitHub ».

20. Source : entretien par courriel avec Cory Benfield.
21. Dans les études de genre, cisgenre décrit un type d'identité de genre où le genre ressenti d'une personne correspond au genre qui lui a été assigné à la naissance. Voir « Cisgenre », sur Wikipédia.
22. Source : entretien par courriel avec Charlotte Spencer.
23. Jessica Lord, « Privilege, Community and Open Source », 18/05/2015.

Ses contributions régulières attirèrent l'attention de la plateforme GitHub elle-même, pour qui elle travaille désormais. Cependant, Jessica note qu'elle a pu contribuer à l'*open source* grâce à un concours de circonstances « particulier » : elle a accepté une baisse de salaire pour travailler à Code for America, utilisé toutes ses économies, travaillé « presque non-stop » sur des projets *open source*, et bénéficié d'une communauté de soutiens.

À propos du manque de diversité dans l'*open source*, Jessica écrit :

> La valeur des savoirs communs ne peut être surestimée. Nous devons faire mieux. Nous avons besoin des idées de tout le monde. C'est le but que nous devrions chercher à atteindre. Il est nécessaire que l'*open source* soit ouvert à tous. Pas seulement aux privilégiés ou même aux seuls développeurs.

Ce dernier point soulevé par Jessica Lord est révélateur : permettre à des profils plus divers de participer à l'*open source* peut aider à pérenniser l'*open source* en elle-même. D'un point de vue fonctionnel, la grande majorité des contributeurs *open source* sont des développeurs, mais beaucoup d'autres rôles sont nécessaires pour gérer les projets d'ampleur, comme la rédaction, la gestion de projet ou la communication. Les projets *open source* ne sont pas différents des autres types d'organisations, y compris les *startups* où l'on a besoin de personnes se chargeant de l'administration, du marketing, du design, etc. qui sont autant de strates nécessaires au fonctionnement de base d'une structure. C'est en partie parce que la culture *open source* repose principalement sur les développeurs que la pérennité financière est si rarement l'objet de discussions et d'actions concrètes.

Enfin, l'homogénéité des contributeurs *open source* impacte les efforts en faveur de la diversité dans le monde de la technologie au sens large, puisque l'*open source* est étroitement lié à l'embauche. En effet, comme nous l'avons remarqué plus haut, beaucoup d'employeurs utilisent les contributions *open source*, notamment les profils GitHub, pour découvrir leurs futurs employés potentiels ou pour vérifier les qualifications d'un

candidat. Les employeurs qui se fient ainsi essentiellement aux preuves de contributions *open source* ne recrutent que parmi un vivier de candidats extrêmement restreint.

Ashe Dryden, dans un essai important intitulé *L'éthique du travail non payé et la communauté OSS*, expliquait[24] :

> Juger que quelqu'un est un bon programmeur en s'appuyant uniquement sur le code qu'il rend disponible publiquement, c'est exclure bien plus que les marginaux. C'est aussi exclure quiconque n'est pas autorisé à publier son code publiquement pour des raisons de licence ou de sécurité. Cela concerne également un grand nombre de travailleurs *freelance* ou de contractuels qui, pour des raisons légales, n'ont pas le droit de revendiquer publiquement leur participation à un projet (par exemple, s'ils ont signé un accord de confidentialité). Dans une industrie où on lutte pour dénicher assez de talents, pourquoi limitons-nous artificiellement le spectre des candidats ?

Comment atténuer ou éviter certains des coûts qui s'imposent aux personnes qui participent à l'élaboration d'infrastructures numériques aujourd'hui ? Pour commencer, nous analyserons comment les projets d'infrastructure sont actuellement financés.

24. Ashe Dryden, « The Ethics of Unpaid Labor and the OSS Community », 13/11/2013.

4

SOUTENIR LES INFRASTRUCTURES NUMÉRIQUES

« Même s'il est vrai que le projet OpenSSL "appartient au peuple", il ne serait ni réaliste ni correct d'attendre de quelques centaines, ou même de quelques milliers d'individus seulement, qu'ils le financent à eux seuls. »

Steve Marquess, OpenSSL

DES MODÈLES ÉCONOMIQUES POUR LES INFRASTRUCTURES NUMÉRIQUES

Certains aspects des infrastructures numériques peuvent fonctionner dans un contexte concurrentiel. Les bases de données et les services d'hébergement, par exemple, sont souvent des affaires profitables, bien financées, parce qu'elles peuvent faire payer l'accès. Tout comme l'accès à l'eau ou à l'électricité, l'accès à un serveur ou à une base de données peut être mesuré, facturé, et fermé si les honoraires ne sont pas réglés.

Heroku (mentionné au début de cet ouvrage[1]) et Amazon Web Services sont deux exemples notables de plateformes qui vendent des services d'infrastructure numérique à des développeurs logiciels contre une redevance (à noter qu'aucun des deux n'est un projet *open source*). Des projets *open source* similaires, à ce niveau d'infrastructure, tels que OpenStack (une plateforme concurrente d'Amazon Web Services) ou MySQL (une base de données), ont trouvé leurs assises dans des entreprises. OpenStack est financé par un consortium d'entreprises, et MySQL a été racheté par Oracle.

C'est en partie l'absence de « bruit » qui rend ces services financièrement attractifs. Pour un seul logiciel, un développeur utilise parfois 20 bibliothèques distinctes, avec chacune des fonctions différentes, mais il n'a besoin que d'une seule base de données. En conséquence, les projets à succès ont plus de chances d'obtenir l'attention et le soin dont ils ont besoin.

Il existe une autre façon utile de cerner les infrastructures que l'on peut facturer : s'il y a un risque immédiat de défaillance, alors il y a probablement un modèle économique. En d'autres termes, un serveur peut subir des interruptions de

1. Voir partie 1, chapitre 1.

service inattendues, tout comme l'électricité peut sauter à l'improviste, mais un langage de programmation ne « casse » ni n'a de telles périodes d'indisponibilité, parce qu'il s'agit d'un système d'information[2].

Pour ce genre de projets *open source*, le modèle économique a tendance à se focaliser sur la recherche de services ou d'assistance facturables. Cela fonctionne pour les projets qui bénéficient d'un usage significatif par les entreprises, en particulier quand il s'agit d'un problème techniquement complexe, ou lorsqu'une entreprise a besoin qu'une fonction soit développée.

Récompenses

À petite échelle, des personnes ou des entreprises promettent parfois des « récompenses » d'ordre pécuniaire lorsque certains objectifs de développement ont été atteints.

Par exemple, IBM demande régulièrement de nouvelles fonctionnalités pour divers projets par le biais d'un site web appelé Bountysource[3], offrant jusqu'à 5 000 dollars par tâche. Bountysource est une plateforme populaire pour trouver et proposer des récompenses ; elle compte plus de 26 000 membres.

Les récompenses aident à régler les problèmes précédemment mentionnés en faisant un don pour un projet précis. Comme les récompenses sont clairement liées à un résultat, l'argent va être utilisé. En revanche, les récompenses peuvent avoir des effets pervers pour l'incitation à contribuer à un projet.

Les récompenses peuvent dicter quel travail sera ou ne sera pas effectué, et parfois ce travail n'est pas en phase avec les priorités d'un projet. Il peut aussi introduire du bruit dans le système : par exemple, une entreprise peut offrir une forte récompense pour une fonctionnalité que les propriétaires du projet ne considèrent pas comme importante.

Du côté des contributeurs, des personnes extérieures sans connaissance sur un projet peuvent y participer seulement pour

2. Merci à Sam Gerstenzang d'avoir fait le point sur cette différence. Voir son Tweet, daté du 13/01/2016.
3. Voir le site de Bountysource (Bountysource.com).

obtenir la récompense, puis le quitter. Ou bien elles peuvent bâcler le travail requis, parce qu'elles essaient d'obtenir des récompenses. Enfin, les récompenses peuvent être une façon appropriée de financer de nouvelles fonctionnalités ou des problèmes importants, mais sont moins pratiques lorsqu'il s'agit de financer des opérations continues, comme le service client ou la maintenance.

Jeff Atwood, le créateur de Stack Overflow, a remarqué les problèmes suivants avec les programmes de récompenses, en particulier en ce qui concerne la sécurité[4] :

> L'un des effets pervers de cette tendance à attribuer des récompenses pour les rapports de bugs est que cela n'attire pas seulement de véritables programmeurs intéressés par la sécurité, mais aussi toutes les personnes intéressées par l'argent facile. Nous avons reçu trop de rapports de bugs de sécurité « sérieux » qui n'avaient qu'une importance très faible. Et nous devons les traiter, parce qu'ils sont « sérieux », n'est-ce pas ? Malheureusement, beaucoup d'entre eux ne représentent qu'un gaspillage de temps… Ce genre d'incitation me semble vraiment néfaste. Même si je sais que la sécurité est extrêmement importante, je suis de plus en plus inquiet face à ces interactions parce qu'elles me demandent beaucoup de travail et que le retour sur investissement est très faible.

Services

À une plus vaste échelle, un des exemples bien connus et le plus souvent cités de modèle économique *open source*, c'est Red Hat, l'entreprise dont nous avons déjà parlé, qui propose une assistance, des sessions de formation et autres services à des entreprises qui utilisent Linux. Red Hat a été fondée en 1993, il

4. « Given Enough Money, All Bugs Are Shallow », *Coding Horror*, 03/04/2015.

s'agit d'une entreprise cotée en bourse avec un chiffre d'affaires déclaré de 2 milliards de dollars par an.

Bien que Red Hat ait connu un succès fantastique d'un point de vue financier, nombreux sont ceux qui soulignent qu'il s'agit d'une anomalie sans lendemain. Red Hat a bénéficié de l'avantage du premier arrivé dans son domaine technologique. Matt Asay, un journaliste spécialisé en *open source*, a remarqué que Red Hat utilise un ensemble unique de licences et brevets pour protéger ses parts de marché. Asay, qui auparavant était un fervent défenseur des entreprises *open source*, est maintenant persuadé que certaines licences propriétaires sont nécessaires pour faire sérieusement des affaires[5]. Matthew Aslet du 451 Group, un organisme de recherche, a découvert lui aussi que la plupart des entreprises *open source* qui réussissent utilisent en fait indifféremment un type de licence commerciale[6].

Docker, déjà mentionné plus haut[7], est un projet *open source* qui aide les applications à fonctionner efficacement. C'est l'exemple le plus récent d'entreprise qui s'inspire de ce modèle. Docker a levé 180 millions de dollars en capital-risque auprès d'investisseurs, avec une valorisation d'un milliard de dollars de la part d'investisseurs privés[8]. Comme sa part de marché s'est accrue, Docker a commencé à proposer des services d'assistance au niveau des entreprises. Or sans revenus solides, Docker pourrait n'être qu'un exemple de plus de capital-risque qui fait un investissement dans une entreprise d'infrastructure *leader* sur son marché, mais qui réalise des pertes.

À petite échelle, beaucoup de développeurs proposent des services de consultant pour pouvoir financer leur travail. Hoodie[9] est un *framework* poids plume qui repose sur Node et qui a réussi dans les services de consultant.

Hoodie lui-même est un projet *open source*. Plusieurs mainteneurs gagnent leur vie grâce à la boutique de l'entreprise, Neighbourhoodie, qui propose des services de développement

5. Matt Asay, «Beyond $1bn: Why Red Hat is a one off», *The Register*, 29/03/2011.
6. Matthew Aslett, « Open source is not a business model », *The 451 Group*, 13/10/2008.
7. Voir partie 2, chapitre 2.
8. Page Docker sur *Crunchbase*.
9. À l'origine, un *hoodie* est un sweat-shirt à capuche. Voir le site Hoodie (hood.ie).

logiciel[10]. Bien que Neighbourhoodie se spécialise dans le *framework* de Hoodie, ce dernier est encore un projet plutôt jeune, de sorte que certaines parties de son travail proviennent de projets qui ne sont pas liés à Hoodie[11]. Dans le cas de Hoodie, le modèle de services choisi est censé payer le salaire de plusieurs mainteneurs, plutôt que de viser une stratégie d'entreprise à l'échelle de Red Hat.

Le conseil est une option viable pour les développeurs indépendants, si suffisamment d'utilisateurs du projet sont d'accord et ont l'argent nécessaire pour payer de l'aide supplémentaire. Mais à petite échelle, cela peut aussi les empêcher d'améliorer le projet lui-même, puisque les deux personnes au plus qui le maintiennent passent désormais leur temps à développer leur affaire et à fournir des services qui peuvent ou non être en accord avec les besoins du projet en termes de maintenance.

Aspirer à une activité de consultant peut aussi entrer en contradiction avec l'objectif de rendre le produit facile à utiliser et à appréhender, ce qui est bien dans l'esprit de l'*open source*. Twisted, la bibliothèque Python déjà citée[12], a mentionné un témoignage plein d'humour de l'un de ses utilisateurs, une entreprise nommée Mailman[13] : « Les gars, vous avez un gros problème, parce que c'était vraiment trop facile ! Comment vous comptez vous faire un paquet d'argent juste avec du conseil ? »

En fin de compte, le « modèle économique » pour un projet *open source* n'est pas très différent du simple travail indépendant.

Licences payantes

Certains développeurs ont l'impression que mettre les projets sous licence serait une solution au moins partielle aux problèmes de financement de l'*open source*. Si les projets *open source* sont fortement utilisés, pourquoi ne pas les facturer ? Ces

10. Voir le site Neighbourhood.ie/, littéralement « les voisins de Hoodie ».
11. Source : entretien téléphonique avec Jan Lehnardt, le PDG de Neighbourhoodie.
12. Voir partie 2, chapitre 2.
13. Voir la page Success stories, sur Twistedmatrix.com.

« licences payantes » ne sont techniquement pas des licences *open source*, selon la définition de l'Open Source Initiative[14]. Il s'agit plutôt d'initiatives qui tentent d'apporter un équilibre entre le besoin très concret de travail rémunéré et le désir de rendre le code accessible au public. Ce type de code peut être appelé « à source visible » ou « à source disponible ». Fair Source, par exemple, se décrit lui-même comme « (offrant) certains des avantages de l'*open source* tout en préservant la possibilité de faire payer pour le logiciel ».

La licence Fair Source[15] fut créée en novembre 2015 par une entreprise appelée Sourcegraph pour répondre au besoin de licence payante. Les termes de la licence ont été rédigés par Heather Meeker, une juriste qui a également travaillé dans l'équipe principale de la Mozilla Public License v2.0. Avec la licence Fair Source, on peut librement consulter, télécharger, exécuter et modifier le code, jusqu'à un certain nombre d'utilisateurs par organisation. Une fois cette limite dépassée, l'organisation doit payer un forfait de licence, dont le montant est déterminé par l'éditeur. En d'autres termes, le code Fair Source est gratuit pour un usage personnel et pour les PME, mais fournit une base légale pour facturer les cas de plus gros usages commerciaux.

L'annonce par Sourcegraph de la création de la licence Fair Source, qu'ils utilisent maintenant eux-mêmes, a provoqué un débat animé sur la monétisation de l'*open source*. (Il est à noter qu'un mouvement similaire autour du *shareware*, logiciel propriétaire gratuit, avait émergé avec un certain succès populaire dans les années 1980.)

Mike Perham, l'un des mainteneurs de Sidekiq, un outil populaire pour le développement en Ruby, a aussi récemment suggéré aux contributeurs et contributrices *open source* d'utiliser une « licence duale » pour monétiser leur travail, faisant payer aux entreprises l'accès à une licence MIT permissive plutôt qu'une licence AGPL plus restrictive qui impose l'attribution. Sa théorie est qu'en faisant d'AGPL la licence par défaut, « les entreprises vont payer pour l'éviter ».

14. Voir le site Opensource.org.
15. Voir le site fair.io.

Pour justifier cette idée, Perham a rappelé à son public[16] :

> Souvenez-vous : logiciel *open source* ne signifie pas logiciel gratuit. Ce n'est pas parce que l'on peut consulter la source sur GitHub que tout le monde peut l'utiliser et en faire n'importe quoi[17]. Il n'y a aucune raison pour laquelle vous ne pourriez pas donner l'accès à votre code mais aussi faire payer pour son utilisation. Tant que vous possédez le code, vous avez le droit d'y attribuer la licence que vous voulez.
>
> […] la réalité, c'est que la plupart des petits projets *open source* dépendent d'une seule personne qui fait 95 % du travail. Si c'est votre cas, soyez reconnaissants envers les personnes qui vous aident gratuitement, mais ne vous sentez pas coupable de garder 100 % du revenu.

Faire payer les entreprises offre une autre possibilité aux développeurs et développeuses qui souhaitent poursuivre leur travail, en particulier s'il n'y a qu'une ou deux personnes pour maintenir un projet actif. Cependant, tous les projets ne peuvent pas faire payer pour le travail fourni, en particulier les projets plus vieux, ou les projets d'infrastructure qui ressemblent plus à des biens publics qu'à des produits de consommation, comme les langages de programmation.

Même si les licences payantes peuvent fonctionner pour certains scénarios, ce modèle peut aussi être considéré comme en porte-à-faux avec l'énorme valeur sociale offerte par l'*open source*, qui suggère que lorsque le logiciel est libre, l'innovation suit.

L'objectif ne devrait pas être le retour à une société qui repose sur les logiciels fermés, où le progrès et la créativité sont limités, mais de soutenir de façon durable un écosystème public dans lequel le logiciel peut être créé et distribué librement.

16. Mike Perham, « How to Charge for your Open Source », site personnel Mikeperham.com, 23/11/2015.

17. Ça vaut le coup de noter que Mike a raison à propos du code hébergé sur GitHub sans licence spécifique, mais une licence *open source* telle que définie par l'OSI doit inclure le droit de redistribution. Cette citation souligne à quel point la définition actuelle de l'*open source* est floue, avec un usage habituel qui s'éloigne de la définition historique.

TROUVER DES MÉCÈNES OU DES DONATEURS POUR FINANCER UN PROJET D'INFRASTRUCTURE

La deuxième option pour financer des projets d'infrastructure numérique consiste à trouver des mécènes ou des donateurs. Il s'agit d'une pratique courante dans les cas de figure suivants :

- il n'existe pas de demande client facturable pour les services proposés par le projet ;
- rendre l'accès payant empêcherait l'adoption (on ne pourrait pas, par exemple, faire payer l'utilisation d'un langage de programmation comme Python, car personne ne l'utiliserait ; ce serait comme si parler anglais était payant) ;
- le projet n'a pas les moyens de financer des emplois rémunérés ou bien il n'y a pas de volonté de la part du développeur de s'occuper des questions commerciales ;
- la neutralité et le refus de la commercialisation sont considérés comme des principes importants en termes de gouvernance.

Dans ce type de situation, un porteur de projet cherchera des mécènes qui croient en la valeur de son travail et qui sont disposés à le soutenir financièrement. À l'heure actuelle, il existe deux sources principales de financement : les entreprises de logiciel et les autres développeurs.

Le financement participatif

Certains travaux de développement obtiennent des fonds grâce à des campagnes de financement participatif (*crowdfunding*) *via* des plateformes telles que Kickstarter ou Indiegogo. Bountysource, le site de récompenses dont nous parlions dans

un chapitre précédent, possède également une plateforme appelée Salt dédiée au financement participatif de projets *open source*.

Andrew Godwin, un développeur du noyau Django résidant à Londres, a ainsi réussi à récolter sur Kickstarter 17,952 livres sterling (environ 21 000 euros) de la part de 507 contributeurs, afin de financer des travaux de base de données pour Django. Le projet a été entièrement financé en moins de quatre heures.

Pour expliquer sa décision de lever des fonds pour un projet *open source*, Godwin écrit[1] :

> Une quantité importante de code *open source* est écrite gratuitement. Cependant, mon temps libre est limité. Je dispose actuellement d'une seule journée libre par semaine pour travailler, et j'adorerais la consacrer à l'amélioration de Django, plutôt qu'à du conseil ou à de la sous-traitance.
>
> L'objectif est double : d'une part, garantir au projet un temps de travail important et au moins quatre-vingts heures environ de temps de codage ; et d'autre part prouver au monde que les logiciels *open source* peuvent réellement rémunérer le temps de travail des développeurs.

À l'instar des récompenses, le financement participatif s'avère utile pour financer de nouvelles fonctionnalités, ou des développements aboutissant à un résultat clair et concret. Par ailleurs, le financement participatif a moins d'effets pervers, notamment parce qu'organiser une campagne de financement demande plus d'efforts que de poster une offre de récompense, et parce que le succès du financement repose en grande partie sur la confiance qu'a le public dans la capacité du porteur de projet à réaliser le travail annoncé. Dans le cas de Godwin, il était l'un des principaux contributeurs au projet Django depuis six ans et était largement reconnu dans la communauté.

1. Voir sur le site de *crowdfunding* Kickstarter, Andrew Godwin, « Schemas Migration for Django » de juillet 2013.

Toutefois, le financement participatif ne répond pas à la nécessité de financer les frais de fonctionnement et les frais généraux. Ce n'est pas une source régulière de capital. En outre, planifier et mettre en œuvre une campagne de financement participatif demande à chaque fois un investissement important en temps et en énergie. Enfin, les donateurs pour ces projets sont souvent eux-mêmes des développeurs ou des petites entreprises – et un porteur de projet ne peut pas éternellement aller frapper à la même porte pour financer ses projets.

Avec le recul, Godwin a commenté sa propre expérience[2] :

> Je ne suis pas sûr que le financement participatif soit totalement compatible avec le développement *open source* en général ; non seulement c'est un apport ponctuel, mais en plus l'idée de rétribution est souvent inadéquate car elle nécessite de promettre quelque chose que l'on puisse *a priori* garantir et décrire.
>
> S'en remettre uniquement à la bonne volonté du public, cela ne fonctionnera pas. On risque de finir par s'appuyer de manière disproportionnée sur des développeurs, indépendants ou non, à un niveau personnel – et je ne pense pas que ce soit viable.

À côté des campagnes de financement participatif, plusieurs plateformes ont émergé pour encourager la pratique du « pourboire » (*tipping* en anglais) pour les contributeurs *open source* : cela consiste à verser une petite somme de revenu régulier à un contributeur, en signe de soutien de son travail. Deux plateformes populaires se distinguent : Patreon (qui ne se limite pas exclusivement aux contributeurs *open source*) et Gratipay (qui tend à fédérer une communauté plus technique).

L'idée d'un revenu régulier est alléchante, mais souffre de certains problèmes communs avec le financement participatif. On remarque notamment que les parrains (*patrons* ou

2. Commentaire de andrewgodwin sur *Paying the piper* (projet GitHub), 13/10/2015.

tippers en anglais) sont souvent eux-mêmes des développeurs, dotés de capitaux limités à se promettre les uns aux autres. Les dons ont généralement la réputation de pouvoir financer une bière, mais pas un loyer. Gratipay rassemble 122 équipes sur sa plateforme, qui reçoivent collectivement 1 000 dollars par semaine, ce qui signifie qu'un projet touche en moyenne moins de 40 dollars par mois[3].

Même les très gros projets tels que OpenSSL ne généraient que 2 000 dollars de dons annuels avant la faille Heartbleed. Comme expliqué précédemment[4], après Heartbleed, Steve Marquess, membre de l'équipe, a remarqué « un déferlement de soutien de la part de la base de la communauté OpenSSL » : la première vague de dons a rassemblé environ 200 donateurs pour un total de 9 000 dollars.

Marquess a remercié la communauté pour son soutien mais a également ajouté[5] :

> Même si ces donations continuent à arriver indéfiniment au même rythme (ce ne sera pas le cas), et même si chaque centime de ces dons allait directement aux membres de l'équipe OpenSSL, nous serions encore loin de ce qu'il faudrait pour financer correctement le niveau de main-d'œuvre humaine nécessaire à la maintenance d'un projet aussi complexe et aussi crucial. Même s'il est vrai que le projet « appartient au peuple », il ne serait ni réaliste ni correct d'attendre de quelques centaines, ou même de quelques milliers d'individus seulement, qu'ils le financent à eux seuls. Ceux qui devraient apporter les vraies ressources, ce sont les entreprises lucratives et les gouvernements qui utilisent OpenSSL massivement et qui le considèrent comme un acquis.

3. Ces chiffres relevés sur la page Stats-About-Gratipay, sur gratipay.com, datent de novembre 2015. En janvier 2017, 284 projets se partageaient 900 dollars par semaine.
4. Voir l'introduction de l'ouvrage.
5. Voir Steeve Marquess, « Of money responsability and pride », *Speeds and Feeds*, blog personnel, 12/04/2014.

(À l'appui de l'argument de Marquess, les dons de la part des entreprises furent par la suite plus importants, les sociétés ayant davantage à donner que les particuliers. La plus grosse donation provint d'un fabricant de téléphones chinois, Smartisan, pour un montant de 160 000 dollars[6]. Depuis, Smartisan a continué de faire des dons substantiels au projet OpenSSL[7].)

Au bout du compte, la réalité est la suivante : il y a trop de projets, tous qualitatifs ou cruciaux à leur manière, et pas assez de donateurs, pour que la communauté technique (entreprises ou individus) soit en mesure de prêter attention et de contribuer significativement à chacun d'eux.

Le mécénat d'entreprises pour les projets d'infrastructure

À plus grande échelle, dans certains cas, la valeur d'un projet devient si largement reconnue qu'une entreprise finit par recruter un contributeur pour travailler à plein temps à son développement.

John Resig est l'auteur de jQuery, une bibliothèque de programmation JavaScript qui est utilisée par près des deux tiers du million de sites web les plus visités au monde[8]. John Resig a développé et publié jQuery en 2006, sous la forme d'un projet personnel. Il a rejoint Mozilla en 2007 en tant que développeur évangéliste, se spécialisant notamment dans les bibliothèques JavaScript[9].

La popularité de jQuery allant croissant, il est devenu clair qu'en plus des aspects liés au développement technique, il allait falloir formaliser ceux liés à la gouvernance du projet. Mozilla a alors proposé à John de travailler à plein temps sur jQuery entre 2009 et 2011, ce qu'il a fait.

6. Voir Steven Norton, « OpenSSL Seeing More Support Post-Heartbleed », *Wall Street Journal*, 20/08/2014.
7. Source : entretien par courriel avec Steve Marquess.
8. Libscore.com collecte des données sur l'utilisation des bibliothèques Javascript par les sites web. jQuery est de loin la plus utilisée.
9. Voir John Resig, « Starting with Mozilla » sur sa page personnelle, 02/01/2007.

À propos de cette expérience, John Resig a écrit[10] :

> Pendant l'année et demi qui vient de s'écouler, Mozilla m'a permis de travailler à plein temps sur jQuery. Cela a abouti à la publication de 9 versions de jQuery… et à une amélioration drastique de l'organisation du projet (nous appartenons désormais à l'organisation à but non lucratif Software Freedom Conservancy, nous avons des réunions d'équipe régulières, des votes publics, fournissons des états des lieux publics et encourageons activement la participation au projet). Heureusement, le projet jQuery se poursuit sans encombre à l'heure actuelle, ce qui me permet de réduire mon implication à un niveau plus raisonnable et de participer à d'autres travaux de développement.

Après avoir passé du temps chez Mozilla pour donner à jQuery le support organisationnel dont il avait besoin, John a annoncé qu'il rejoindrait la Khan Academy afin de se concentrer sur de nouveaux projets.

Cory Benfield, développeur Python, a suivi un chemin similaire. Après avoir contribué à plusieurs projets *open source* sur son temps libre, il est devenu un développeur clé pour une bibliothèque essentielle de Python intitulée Requests. Cory Benfield note que[11] :

> Cette bibliothèque a une importance comparable à celle de Django, dans la mesure où les deux sont des « infrastructures critiques » pour les développeurs Python. Et pourtant, avant que j'arrive sur le projet, elle était essentiellement maintenue par une seule personne.

Benfield estime qu'il a travaillé bénévolement sur le projet environ douze heures par semaine pendant presque quatre ans,

10. Voir John Resig, « Next Steps in 2011 » sur sa page personnelle, 03/05/2011.
11. Source : entretien par courriel avec Cory Benfield.

en plus de son travail à plein temps. Personne n'était payé pour travailler sur Requests.

Pendant ce temps, HP embauchait un employé, Donald Stufft, pour se consacrer spécifiquement aux projets en rapport avec Python, un langage qu'il considère comme indispensable à ses logiciels. (Donald est le développeur cité précédemment[12] qui est payé à plein temps pour travailler sur le packaging Python.) Donald a alors convaincu son supérieur d'embaucher Cory pour qu'il travaille à temps plein sur des projets Python. C'est toujours le cas.

Les entreprises sont des acteurs tout désignés pour soutenir financièrement les projets bénévoles qu'elles considèrent comme indispensables à leurs activités et quand des cas comme ceux de John Resig ou de Cory Benfield surviennent, ils sont chaleureusement accueillis. Cependant, il y a des complications.

Premièrement, aucune entreprise n'est obligée d'embaucher quelqu'un pour travailler sur des projets en demande de soutien ; ces embauches ont tendance à advenir par hasard de la part de mécènes bienveillants. Et même une fois qu'un employé est embauché, il y a toujours la possibilité de perdre ce financement, notamment parce que l'employé ne contribue pas directement au résultat net de l'entreprise. Une telle situation est particulièrement périlleuse si la viabilité d'un projet dépend entièrement d'un seul contributeur employé à plein temps. Dans le cas de Requests, Cory est le seul contributeur à plein temps (on compte deux autres contributeurs à temps partiel, Ian Cordasco et Kenneth Reitz).

Une telle situation s'est déjà produite dans le cas de rvm, un composant critique de l'infrastructure Ruby. Michal Papis, son auteur principal, a été engagé par Engine Yard entre 2011 et 2013 pour soutenir le développement de rvm. Mais quand ce parrainage s'est terminé, Papis a dû lancer une campagne de financement participatif pour continuer de financer son développement[13].

12. Voir partie 3, chapitre 3.
13. Voir sur le site de *crowdfunding* Bountysource, Michal Papis, « RVM », janvier 2014.

Le problème, c'est que cela ne concernait pas seulement rvm. Engine Yard avait embauché plusieurs mainteneurs de projets d'infrastructure Ruby, qui travaillaient notamment sur JRuby, Ruby on Rails 3 et bundler. Quand les responsables d'Engine Yard ont été contraints de prendre la bonne décision pour assurer la viabilité de leur entreprise, à savoir réduire leur soutien financier, tous ces projets ont perdu leurs mainteneurs à temps plein, et presque tous simultanément.

L'une des autres craintes est qu'une entreprise unique finisse par avoir une influence disproportionnée sur un projet, puisqu'elle en est *de facto* le seul mécène. Cory Benfield note[14] également que le contributeur/la contributrice lui/elle-même peut avoir une influence disproportionnée sur le projet, puisqu'il/elle dispose de beaucoup plus de temps que les autres pour faire des contributions. De fait, une telle décision peut même être prise par une entreprise et un mainteneur, sans consulter le reste de la communauté du projet.

On peut en voir un exemple dans le cas d'Express.js, un *framework* important pour l'écosystème Node.js. Quand l'auteur du projet a décidé de passer à autre chose, il en a transféré les actifs (en particulier le dépôt du code source et le nom de domaine) à une société appelée StrongLoop dont les employés avaient accepté de continuer de maintenir le projet[15]. Cependant StrongLoop n'a pas fourni le soutien qu'attendait la communauté, et comme les employés de StrongLoop étaient les seuls à avoir un accès administrateur, il est devenu difficile pour la communauté de faire des contributions. Doug Wilson, l'un des principaux mainteneurs (non affilié à StrongLoop), disposait encore d'un accès commit et a continué de traiter la charge de travail du projet, essayant tant bien que mal de tout gérer à lui seul.

Après l'acquisition de StrongLoop par IBM, Doug déclara que StrongLoop avait bel et bien tué la communauté des contributeurs[16].

14. Source : entretien par courriel avec Cory Benfield.
15. Voir Al Tsang, sur le site de la société Strongloop, « TJ Holowaychuk Passes Sponsorship of Express to StrongLoop », 29/07/2014.
16. Commentaire de Dougwilson sur *is express dying ?* (projet GitHub), 18/01/2016.

> Au moment où on est passé à StrongLoop, il y avait des membres actifs comme @Fishrock123 qui travaillaient à créer… de la documentation. Et puis tout à coup, je me suis retrouvé tout seul à faire ça sur mon temps libre alors que les demandes de support ne faisaient que se multiplier… et pendant tout ce temps, je me suis tué à la tâche, je me suis engagé pour le compte de StrongLoop. Quoi qu'il arrive, jamais plus je ne contribuerai à aucun dépôt logiciel appartenant à StrongLoop.

Finalement, le projet Express.js a été transféré de StrongLoop à la Fondation Node.js, qui aide à piloter des projets appartenant à l'écosystème technologique Node.js.

En revanche, pour les projets *open source* qui ont davantage d'ampleur et de notoriété, il n'est pas rare d'embaucher des développeurs. La Fondation Linux a fait savoir, par exemple, que 80 % du développement du noyau Linux est effectué par des développeurs rémunérés pour leur travail[17]. La Fondation Linux emploie également des *fellows*[18] payés pour travailler à plein temps sur les projets d'infrastructure, notamment Greg Kroah-Hartman, un développeur du noyau Linux, et Linus Torvalds lui-même, le créateur de Linux.

17. Voir le « Linux Development Report » publié par la Fondation Linux, 18/02/2015.
18. « Compagnons » selon un titre consacré (NdT). Le terme *fellow* est intraduisible sans longue périphrase, pour en savoir plus, voir l'article « Fellow » sur Wikipédia.

POURQUOI CES PROJETS SONT-ILS SI DIFFICILES À FINANCER ?

Aujourd'hui, le travail sur les infrastructures numériques est effectué par des développeurs *freelance* ou ayant un « job alimentaire ». Leur temps libre est consacré aux projets *open source*, et ils exercent par ailleurs un travail rémunéré sans rapport avec eux. C'est une solution viable pour financer son quotidien, mais qui ne permet pas d'apprécier à sa juste valeur l'apport social de ces projets.

Étonnamment, bien que tout le monde soit d'accord pour reconnaître qu'il y a un problème (qu'on le qualifie de « *burn-out* du bénévole », de mauvaise gestion de la communauté ou de financement insuffisant), la discussion ne dépasse pas le stade de maigres solutions à court terme comme les « pourboires » ou le *crowdfunding*.

Discutez avec des développeurs qui ont trouvé un moyen d'être financièrement autonomes et vous entendrez à tout bout de champ le mot « chanceux » : chanceux d'avoir été embauché par une entreprise, chanceux d'avoir eu de la notoriété et des dons, chanceux d'être tombé sur un modèle économique viable, chanceux de ne pas avoir une famille ou un prêt dont s'inquiéter. Tout le monde peut être chanceux. Mais la chance dure quelques mois, peut-être un an ou deux, et puis elle s'épuise.

Pourquoi est-il si difficile de financer les infrastructures numériques ?

Fondamentalement, l'infrastructure numérique a un problème de passagers clandestins. Les ressources sont disponibles gratuitement et tout le monde les utilise (qu'il s'agisse de développeurs individuels ou de grandes entreprises de logiciels), mais personne n'est encouragé à contribuer en retour,

chacun s'imaginant qu'un autre finira par le faire. S'il est laissé à l'abandon, ce problème mènera à une tragédie des Communs.

À leur enjeu macroéconomique s'ajoutent plusieurs raisons pour lesquelles le financement des infrastructures numériques est particulièrement compliqué. Ces raisons ont déjà été abordées au cours de cette étude, mais sont toutes résumées ici :

- **On croit à tort qu'il s'agit d'un « problème résolu » :** même parmi les acteurs du secteur comme les entreprises de logiciels, la croyance est très répandue que l'*open source* est déjà correctement financée, ce qui rend d'autant plus difficile la levée de fonds. Certains projets d'infrastructure fonctionnent durablement, soit parce qu'ils disposent d'un modèle économique viable ou de mécènes, soit parce que les coûts de maintenance sont limités. Un public novice pourra également faire le lien entre l'*open source* et des entreprises telles que Red Hat ou Docker et penser que le problème a été résolu. Mais il faut garder à l'esprit que ces cas sont l'exception et non la règle.
- **Il manque une prise de conscience et une compréhension culturelle de ce problème :** en dehors de la communauté *open source*, tout le monde, ou presque, ignore les problèmes de financement de ces projets d'infrastructure et le sujet est perçu comme plutôt aride et technique. Les développeurs qui ont besoin de soutien ont tendance à se concentrer principalement sur la technique et sont mal à l'aise lorsqu'il s'agit de défendre l'aspect financier de leur travail. Et, au bout du compte, ils ne trouvent pas l'élan pour changer cette situation sclérosée.
- **Les infrastructures numériques sont enracinées dans l'*open source*, dont la culture du bénévolat n'encourage pas à parler d'argent :** même si cette attitude a fait de l'*open source* ce qu'elle est aujourd'hui, elle crée également un tabou qui rend difficile pour les développeurs l'évocation de leurs besoins, car ils se sentent coupables ou ont peur de passer pour des personnes qui n'auraient pas l'esprit d'équipe. La nature hautement décentralisée et démocratique de l'*open source* rend également difficiles la coordination et le financement d'acteurs institutionnels qui pourraient défendre leurs intérêts.

- **Les infrastructures numériques sont hautement décentralisées, contrairement aux infrastructures physiques :** contrairement à un projet de construction de pont, il n'est pas toujours évident de savoir quels projets seront utiles avant qu'ils n'aient déjà décollé. Ils ne peuvent pas être planifiés à l'avance par un organisme centralisé. Et à l'autre bout du cycle de vie, certains projets sont destinés à tomber en désuétude à mesure que d'autres solutions, meilleures, prendront leur place. L'infrastructure numérique est constituée de centaines de projets, grands ou petits, réalisés par des individus, des groupes ou des entreprises ; en faire l'inventaire serait titanesque.

Kyle Kemp, développeur *freelance* et contributeur *open source*, conclut ainsi[1] :

> Il est difficile de trouver des financements… pour le développeur moyen (comme moi), certains sont totalement hors de portée. Kickstarter ne marche que si tu deviens viral ou si tu embauches quelqu'un pour faire tout ce qui est marketing/design/promotion… Transformer un projet en entreprise, c'est génial aussi mais… ce sont des choses qui t'éloignent du développement (qui est la partie qui m'intéresse). Si je voulais obtenir une subvention, je ne saurais même pas par où commencer.

1. Commentaire de seiyria sur *Paying the piper* (projet GitHub), 28/10/2015 (en anglais).

LES EFFORTS INSTITUTIONNELS POUR FINANCER LES INFRASTRUCTURES NUMÉRIQUES

Il existe des institutions qui s'efforcent d'organiser collectivement les projets *open source* et d'aider à leur financement. Il peut s'agir de fondations indépendantes liées aux logiciels ou d'entreprises de logiciels elles-mêmes qui apportent leur soutien.

Soutien administratif et mécénat financier

Plusieurs fondations fournissent un soutien organisationnel, comme le mécénat financier, aux projets *open source* : plus précisément, la prise en charge des tâches autres que le code, dont beaucoup de développeurs se passent volontiers.

L'Apache Software Foundation, constituée en 1999, a été créée en partie pour soutenir le développement du serveur Apache HTTP, qui dessert environ 55 % de la totalité des sites internet dans le monde[1]. Depuis lors, la Fondation Apache est devenue un foyer d'ancrage pour plus de 350 projets[2] *open source*. Elle se structure comme une communauté décentralisée de développeurs, sans aucun employé à plein temps et avec presque 3 000 bénévoles. Elle propose de multiples services aux projets qu'elle héberge, consistant principalement en un soutien organisationnel, juridique et de développement. En 2011, Apache avait un budget

1. Données statistiques sur W3techs.com, données en continu. En janvier 2017, la part d'Apache est de 50,8 %.
2. Voir le site Apache.org.

annuel de plus de 500 000 dollars, issu essentiellement de subventions et de donations[3].

Le Software Freedom Conservancy, fondée en 2006, fournit également des services administratifs non lucratifs à plus de 30 projets libres et *open source*[4]. Parmi les projets que cette fondation soutient, on retrouve notamment Git, le système de contrôle de versions dont nous avons parlé plus haut et sur lequel GitHub a bâti sa plateforme, et Twisted, une librairie Python déjà citée précédemment[5].

On trouve encore d'autres fondations fournissant un soutien organisationnel, par exemple The Eclipse Foundation et Software in the Public Interest. La Fondation Linux et la Fondation Mozilla soutiennent également des projets *open source* externes de diverses façons (dont nous parlerons plus loin dans ce chapitre), bien que ce ne soit pas le but principal de leur mission.

Il est important de souligner que ces fondations fournissent une aide juridique et administrative, mais rarement financière. Ainsi, être sponsorisé par Apache ou par le Software Freedom Conservancy ne suffit pas en soi à financer un projet ; les fondations ne font que faciliter le traitement des dons et la gestion du projet.

Un autre point important à noter est que ces initiatives soutiennent le logiciel libre et *open source* d'un point de vue philosophique, mais ne se concentrent pas spécifiquement sur ses infrastructures. Par exemple, OpenTripPlanner, projet soutenu par le Software Freedom Conservancy, est un logiciel pour planifier les voyages : même si son code est *open source*, il s'agit d'une application destinée aux consommateurs, pas d'une infrastructure.

3. Voir « Apache Software Foundation », sur Wikipédia.
4. Voir la page Projets sur le site de la Software Freedom Conservancy (sfconservancy.org).
5. Voir partie 2, chapitre 2.

Créer une fondation pour aider un projet

Certains projets sont suffisamment importants pour être gérés à travers leurs propres fondations. Python, Node.js, Django et jQuery sont tous adossés à des fondations.

Il y a deux conditions fondamentales à remplir pour qu'une fondation fonctionne : être éligible à l'exonération d'impôts et trouver des financements.

Réussir à accéder au statut 501(c3)[6], la loi américaine qui définit les organismes sans but lucratif, peut s'avérer difficile pour ces projets, à cause du manque de sensibilisation autour de la technologie *open source* et de la tendance à voir l'*open source* comme une activité non caritative. En 2013, une controverse a révélé que l'IRS (Internal Revenue Service, service des impôts américain) avait dressé une liste de groupes postulant au statut d'exemption fiscale qui nécessiteraient davantage de surveillance : l'*open source* en faisait partie[7] [8]. Malheureusement, ces contraintes ne facilitent pas l'institutionnalisation de ces projets.

Par exemple, Russell Keith-Magee, qui était jusqu'à une époque récente président de la Django Software Foundation, a expliqué que la fondation ne pouvait pas directement financer le développement logiciel de Django, sans prendre le risque de perdre son statut 501(c3). La fondation soutient plutôt le développement *via* des activités communautaires.

En juin 2014, la Fondation Yorba, créatrice des logiciels de productivité qui tournent sous Linux, s'est vu refuser le statut 501(c3) après avoir attendu la décision pendant presque quatre ans et demi. Jim Nelson, son directeur exécutif, a été particulièrement inquiété par le raisonnement de l'IRS : parce que leur logiciel pouvait potentiellement être utilisé par des entités commerciales, le travail de Yorba ne pouvait pas être considéré comme caritatif. Une lettre de l'IRS explique[9] :

6. Pour en savoir plus, voir sur le statut 501(c) des États-Unis. Article « 501c » sur Wikipédia. Le statut 501(c3) est assez proche du statut français des associations à but non lucratif déclarées d'utilité publique.
7. Voir l'article « IRS targeting controversy » sur Wikipédia.
8. Kelly Phillips Erb, « Not Just The Tea Party: IRS Targeted and Turned Down Tax Exempt Status Tied To Open Source Software », *Forbes*, 17/07/2014.
9. Jim Nelson, « The new 501(c)(3) and the future of free software in the United States », *Jim Nelson and Yorba Foundation Archives* (blog), 30/06/2014.

> Se contenter de publier sous une licence *open source* tous usages ne signifie pas que les pauvres et les moins privilégiés utiliseront effectivement les outils. […] On ne peut pas savoir qui utilise les outils, et encore moins quel genre de contenus sont créés avec ces outils.

Nelson a pointé les failles de ce raisonnement dans un billet de blog, comparant la situation à celle d'autres biens publics[10] :

> Il y a une organisation caritative ici à San Francisco qui plante des arbres pour le bénéfice de tous. Si l'un de leurs arbres rafraîchit les clients d'un café pendant qu'ils profitent de leur expresso, cela signifie-t-il que l'organisation qui plante des arbres n'est plus caritative ?

Les projets qui accèdent au statut 501(c3) ont tendance à insister sur l'importance de la communauté, comme la Python Software Foundation, dont l'objet est de « promouvoir, protéger et faire progresser le langage de programmation Python, ainsi que de soutenir et faciliter la croissance d'une communauté diversifiée et internationale de programmeurs Python »[11].

En parallèle, certains projets candidatent pour devenir une association de commerce au sens du statut 501(c6)[12]. La Fondation jQuery en est un exemple, se décrivant comme « une association de commerce à but non lucratif pour développeurs web, financée par ses membres »[13]. La Fondation Linux est également une association de commerce.

Le deuxième aspect de la formalisation de la gouvernance d'un projet à travers une fondation est la recherche de la source de financement adéquate. Certaines fondations sont financées par des donations individuelles, mais ont proportionnellement de petits budgets.

La Django Software Foundation, par exemple, gère Django, le plus populaire des *frameworks web* écrits en Python, utilisé

10. Idem.
11. Présentation de la Python Software Foundation, sur le site Python.org.
12. Voir l'article « 501c » sur Wikipédia.
13. Voir le site jquery.org/.

par des entreprises comme Instagram et Pinterest. La fondation est dirigée par des bénévoles, et reçoit moins de 60 000 dollars de donations par an[14]. L'année dernière, la Django Software Foundation a reçu une subvention ponctuelle de la part de la Fondation Mozilla[15].

Parmi les autres sources habituelles de financement, on trouve les entreprises mécènes. En effet, les entreprises privées sont bien placées pour financer ces projets logiciels, puisqu'elles les utilisent elles-mêmes. La Fondation Linux est l'un de ces cas particuliers qui rencontrent le succès, et ce grâce à la valeur fondamentale du noyau Linux pour les activités de la quasi-totalité des entreprises. La Fondation Linux dispose de 30 millions de dollars d'un capital géré sur une base annuelle, alimenté par des entreprises privées comme IBM, Intel, Oracle et Samsung – et ce chiffre continue d'augmenter[16].

Créer une fondation pour soutenir un projet est une bonne idée pour les projets d'infrastructure très importants. Mais cette solution est moins appropriée pour de plus petits projets, en raison de la quantité de travail, des ressources et du soutien constant des entreprises, nécessaires pour créer une organisation durable.

Node.js est un exemple récent d'utilisation réussie d'une fondation pour soutenir un gros projet. Node.js est un *framework* JavaScript, développé en 2009 par Ryan Dahl et autres développeurs employés par Joyent, une entreprise privée du secteur logiciel. Ce *framework* est devenu extrêmement populaire, mais a commencé à souffrir de contraintes de gouvernance liées à l'encadrement par Joyent, que certaines personnes estimaient incapable de représenter pleinement la communauté enthousiaste et en pleine croissance de Node.js.

En 2014, un groupe de contributeurs de Node.js menaça de *forker* le projet. Joyent essaya de gérer ces problèmes de gouvernance en créant un conseil d'administration pour le projet, mais la scission eut finalement lieu, le nouveau *fork* prenant

14. Django Software Foundation, « 2013 Annual Report », données de 2013.
15. Andrew Godwin, « Django awarded MOSS Grant », DjangoProject.com, 11/12/2015.
16. Source sur collabprojects.linuxfoundation.org (01/03/2017 : lien rompu et renvoi sur la page d'accueil des projets de la Fondation Linux).

le nom d'io.js[17]. En février 2015 fut annoncée l'intention de créer une organisation 501(c6) en vue d'extraire Node.js de la mainmise de Joyent. Les communautés Node.js et io.js votèrent pour travailler ensemble sous l'égide de cette nouvelle entité, appelée la Fondation Node.js. Structurée suivant les conseils de la Fondation Linux, elle dispose d'un certain nombre d'entreprises mécènes qui contribuent financièrement à son budget, notamment IBM, Microsoft et PayPal[18]. Ces sponsors pensent retirer une certaine influence de leur soutien au développement d'un projet logiciel populaire qui fait avancer le Web, et ils ont des ressources à mettre à disposition.

Un autre exemple prometteur est Ruby Together, une organisation initiée par plusieurs développeurs Ruby pour soutenir des projets d'infrastructure Ruby. Ruby Together est structurée en tant qu'association commerciale, dans laquelle chaque donateur, entreprise ou individu investit de l'argent pour financer le travail à temps plein de développeurs chargés d'améliorer le cœur de l'infrastructure Ruby. Les donateurs élisent un comité de direction bénévole, qui aide les membres de Ruby Together à décider chaque mois sur quels projets travailler.

Ruby Together fut conçue par deux développeurs et finance leur travail : André Arko et David Radcliffe. Aujourd'hui, en avril 2016, est également rémunéré le travail de quatre autres mainteneurs d'infrastructure. Le budget mensuel en mars 2016 était d'un peu plus de 18 000 dollars par mois, couvert entièrement par des dons. La création de Ruby Together fut annoncée en mars 2015 et reste un projet récent, mais pourrait bien servir de base à un modèle davantage orienté vers la communauté pour financer la création d'autres projets d'infrastructure[19].

Programmes d'entreprises

Les éditeurs de logiciels soutiennent les projets d'infrastructure de différentes manières.

17. Paul Krill, « Joyent staves off talk of a Node.js fork », *InfoWorld*, 17/10/2014.
18. Scott Hammond, « Introducing the Node.js Foundation », *Joyent*, 10/02/2015.
19. Voir le site Rubytogether.org.

En tant que bénéficiaires des projets d'infrastructures, ils contribuent en faisant remonter des dysfonctionnements et des bugs, en proposant ou soumettant de nouvelles fonctionnalités et en transmettant d'autres types de remarques. Certaines entreprises encouragent leurs employés à participer à des projets cruciaux sur leur temps de travail. De nombreux salariés contribuent ainsi de manière significative à des projets *open source* extérieurs à leur entreprise. Pour certains d'entre eux, travailler sur de l'*open source* fait clairement partie de leur fonction. Pour les entreprises, allouer du temps de travail de leurs salariés à l'*open source* est une des plus importantes façons d'y contribuer.

Les grandes entreprises comme Google ou Facebook adhèrent avec enthousiasme à l'*open source*, de façon à inspirer confiance et renforcer leur influence ; elles sont de fait les seuls acteurs institutionnels assez importants qui peuvent assumer son coût sans avoir besoin d'un retour financier sur investissement. Les projets *open source* aident à renforcer l'influence d'une entreprise, que ce soit en publiant son propre projet *open source* ou en embauchant des développeurs de premier plan pour qu'ils travaillent à plein temps sur un projet *open source*.

Ces pratiques ne sont pas limitées aux entreprises purement logicielles. Walmart, par exemple, qui est un soutien majeur de l'*open source*, a investi plus de deux millions de dollars dans un projet *open source* nommé hapi[20]. Eran Hammer, développeur senior à Walmart Labs, s'est empressé de préciser que « l'*open source*, ce n'est pas du caritatif »[21] et que les ressources d'ingénierie gratuites sont proportionnelles à la taille des entreprises qui utilisent hapi. Dion Almaer, l'ancien vice-président en ingénierie de Walmart Labs, a remarqué que leur engagement envers l'*open source* les aidait à recruter, à construire une solide culture d'entreprise, et à gagner « une série d'effets de levier »[22].

20. Simon Phipps, « Walmart's investment in open source isn't cheap », *InfoWorld*, 22/08/2014.
21. Eran Hammer, « Open Source ain't Charity », *Hueniverse*, 15/08/2014.
22. Dion Almaer, « Why we run an open source program - Walmart Labs », *ToDo* (todogroup.org), 24/02/2015.

En termes de soutien direct au maintien du projet, il arrive que des entreprises embauchent une personne pour travailler à plein temps à la maintenance d'un projet *open source*. Les entreprises contribuent aussi occasionnellement à des campagnes de financement participatif pour un projet particulier. Par exemple, récemment, une campagne sur Kickstarter pour financer un travail essentiel sur Django a reçu 32 650 livres sterling (environ 40 000 euros) ; Tom Christie, l'organisateur de la campagne, a déclaré[23] que 80 % du total venait d'entreprises. Cependant, ces efforts sont toujours consacrés à des projets spécifiques et les infrastructures numériques ne sont pas encore vues communément comme une question de responsabilité sociale par les entreprises de logiciel à but lucratif. Cela laisse encore beaucoup de marge aux actions de défense et de promotion.

L'un des programmes d'entreprise les plus connus est le Summer of Code de Google (été de programmation, souvent nommé GSoC), déjà mentionné dans ce livre[24], qui offre de l'argent à des étudiants pour travailler sur des projets *open source* pendant un été. Ils sont associés à des mentors qui vont les aider à se familiariser avec le projet. Le Summer of Code est maintenu par le bureau des programmes *open source* de Google, et il a financé des milliers d'étudiants[25] [26].

Le but du Summer of Code est de donner à des étudiants la possibilité d'écrire du code pour des projets *open source*, non de financer les projets eux-mêmes.

L'an dernier, Stripe, une entreprise de traitement des paiements, a annoncé une « retraite *open source* », offrant un salaire mensuel d'un maximum de 7 500 dollars pour une session de trois mois dans les locaux de Stripe[27]. À l'origine, l'entreprise voulait uniquement offrir deux bourses, mais après avoir reçu 120 candidatures, le programme a été ouvert à quatre bénéficiaires.

23. Issue de tomchristie, « What level to pitch pricing tiers? », sur *Paying the Piper* (projet GitHub), 28/10/2015.
24. Voir partie 2, chapitre 3.
25. Voir page Gsoc, sur Google Summer of Code.
26. Voir l'article « Summer of Code » sur Wikipédia.
27. Greg Brockman, « Stripe Open-Source Retreat », Stripe.com, 24/04/2014.

Ces derniers ont été enchantés par cette expérience. L'un d'entre eux, Andrey Petrov, continue de maintenir la bibliothèque Python urllib3 dont nous avons déjà parlé[28], et qui est largement utilisée dans l'écosystème Python.

À propos de cette expérience, Andrey a écrit[29] :

> La publication et la contribution au code *open source* vont continuer, que je sois payé pour ou non, mais le processus sera lent et non ciblé. Ce qui n'est pas un problème, car c'est ainsi que l'*open source* a toujours fonctionné. Mais on n'est pas obligé d'en rester là. […]
>
> Si vous êtes une entreprise liée à la technologie, allouez s'il vous plaît un budget à du financement et des bourses pour l'*open source*. Distribuez-le sur Gittip si vous voulez (Gittip est maintenant dénommé Gratipay. Le produit a été quelque peu modifié depuis la publication originelle du billet d'Andrew), ou faites ce qu'a fait Stripe et financez des sprints ambitieux pour atteindre de grands objectifs.
>
> Considérez cela comme une demande solennelle de parrainage : s'il vous plaît, aidez au financement du développement d'urllib3.

La retraite *open source* de Stripe peut servir de modèle aux programmes de soutien. Stripe a décidé de reconduire le programme pour une deuxième année consécutive en 2015. Malgré sa popularité et l'accueil chaleureux reçu des développeurs et développeuses, cette pratique ne s'étend pas toujours aux autres sociétés.

Les entreprises montrent un intérêt croissant pour l'*open source*, et personne ne peut prédire au juste ce que cela donnera sur le long terme. Les entreprises pourraient régler le problème du manque de support à long terme en consacrant des ressources humaines et un budget aux projets *open source*. Des

28. Voir partie 2, chapitre 3.
29. Andrey Petrov, « Urllib3, Stripe, and Open Source Grants », Medium.com, 07/07/2014.

programmes de bourse formalisés pourraient faciliter la mise en contact des entreprises avec des développeurs *open source* ayant besoin d'un soutien à plein temps. Alors que les équipes de contributeurs à un projet étaient souvent composées d'une diversité de développeurs venant de partout, peut-être seront-elles bientôt composées d'un groupe d'employés d'une même entreprise. Les infrastructures numériques deviendront probablement une série de « jardins clos », chacun d'entre eux étant techniquement ouvert et bénéficiant d'un solide soutien, mais en réalité, grâce à ses ressources illimitées, une seule entreprise et ses employés en assureront le soutien.

Cependant, si on pousse la logique jusqu'au bout, ce n'est pas de très bon augure pour l'innovation. Jeff Lindsay, un architecte logiciel qui a contribué à mettre en place l'équipe de Twilio, une entreprise performante de solutions de communication dans le *cloud*, livrait l'an dernier ses réflexions dans une émission[30] :

> À Twilio, on est incité à améliorer le fonctionnement de Twilio, à Amazon on est incité à améliorer le fonctionnement d'Amazon. Mais qui est incité à mieux les faire fonctionner ensemble et à offrir plus de possibilités aux usagers en combinant les deux ? Il n'y a personne qui soit vraiment incité à le faire.

Timothy Fuzz, un ingénieur système, ajoute[31] :

> Pour Bruce Schneier, cette situation tient du servage. Nous vivons dans un monde où Google est une cité-État, où Apple est une cité-État et… si je me contente de continuer à utiliser les produits Google, si je reste confiné dans l'environnement Google, tout me paraît bénéfique. Mais il est quasi impossible de vivre dans un monde où je change d'environnement : c'est très pénible, vous tombez sur des bugs, et aucune de ces entreprises ne cherche vraiment à vous aider. Nous

30. Écouter le podcast de SystemsLive, sur systemslive.org, épisode 51.
31. Bruce Schneier, « When it comes to security, we're back to feudalism », *Wired*, 26/11/2012.

> sommes dans ce monde bizarre, mais si vous regardez du côté des cités-États, l'un des problèmes majeurs c'est le commerce inter-étatique : si on doit payer des droits de douane parce qu'on cherche à exporter quelque chose d'Austin pour le vendre à Dallas, ce n'est pas un bon modèle économique. On pâtit de l'absence d'innovation et de partage des idées. On en est là, aujourd'hui.

Bien que l'argument du « servage » se réfère généralement aux produits d'une entreprise, comme l'addiction à l'iPhone ou à Android, il pourrait être tout aussi pertinent pour les projets *open source* parrainés. Les améliorations prioritaires seront toujours celles qui bénéficient directement à l'entreprise qui paie le développeur. Cette remarque ne relève pas de la malveillance ou de la conspiration : simplement, être payé par une entreprise pour travailler à un projet qui ne fait pas directement partie de ses affaires est une contrainte à prendre en compte.

Mais personne, pas plus Google que la Fondation Linux ou qu'un groupe de développeurs indépendants, ne peut contrôler l'origine d'un bon projet *open source*. Les nouveaux projets de valeur peuvent germer n'importe où, et quand ils rendent un service de qualité aux autres développeurs, ils sont largement adoptés. C'est une bonne chose et cela alimente l'innovation.

Aide spécifique de fondation

Deux fondations ont récemment fait part de leur décision de financer plus spécifiquement l'infrastructure numérique : la Fondation Linux et la Fondation Mozilla.

Après la découverte de la faille Heartbleed, la Fondation Linux a annoncé qu'elle mettait en place l'Initiative pour les infrastructures essentielles (Core Infrastructure Initiative, CII) pour éviter que ce genre de problème ne se reproduise. Jim Zemlin, le directeur général de la Fondation Linux, a réuni près de 4 millions de dollars en promesses de dons

provenant de treize entreprises privées, dont Amazon Web Services, IBM et Microsoft, pour financer des projets liés à la sécurité des infrastructures pour les trois ans à venir[32]. La Fondation Linux s'occupe également d'obtenir des financements gouvernementaux, y compris de la Maison-Blanche[33].

La CII est officiellement un projet de la Fondation Linux. Depuis sa création en avril 2014, la CII a sponsorisé du travail de développement d'un certain nombre de projets, dont OpenSSL, NTP, GnuPG (un système de chiffrement des communications) et OpenSSH (un ensemble de protocoles relatifs à la sécurité). La CII se concentre en priorité sur une partie de l'infrastructure numérique : les projets relatifs à la sécurité.

Au mois d'octobre 2015, Mitchell Baker, la présidente de la Fondation Mozilla, a annoncé la création du Programme de soutien à l'*open source* de Mozilla (Mozilla Open Source Support Program, MOSS) et a promis de consacrer un million de dollars au financement de logiciels libres et *open source*. Selon Baker, ce programme aura deux volets : une « rétribution » pour les projets qu'utilise Mozilla et une « contribution » pour les projets libres et *open source* en général. Grâce aux suggestions de la communauté, Mozilla a sélectionné neuf projets pour la première série de bourses[34]. Ils se disent également prêts à financer des audits de sécurité pour les projets *open source* importants[35].

Enfin, certaines fondations contribuent ponctuellement à des projets de développement logiciel. Par exemple, la Python Software Foundation propose aux individus et aux associations des bourses modestes destinées pour la plupart aux actions pédagogiques et de sensibilisation[36].

32. Seth Rosenblatt, « Tech titans join forces to stop the next Heartbleed », *Cnet*, 24/04/2014.
33. Jim Zemlin, The Linux Foundation's Core Infrastructure Initiative Working with White House on Cybersecurity National Action Plan, Linux.com, 09/02/2016.
34. Mitchell Baker, « Mozilla Launches Open Source Support Program », *The Mozilla Blog*, 23/10/2015.
35. Page MOSS/Secure Open Source, sur *Mozilla Wiki*.
36. The Python Software Foundation, *PSF Board Resolutions*, sur python.org.

Autres acteurs institutionnels

Il existe plusieurs autres acteurs qui apportent diverses formes de soutien aux infrastructures numériques : GitHub, le capital-risque et le monde universitaire.

Si Facebook est un « utilitaire social »[37] et Google un « utilitaire de recherche », tous deux régulant *de facto* les corps dans leur domaine respectif – alors GitHub a une chance de devenir « l'utilitaire *open source* ». Son modèle économique l'empêche de devenir un mastodonte financier (contrairement à Facebook ou Google dont le modèle est fondé sur la publicité, alors que GitHub se monétise par l'hébergement de code pour les clients professionnels, et par l'hébergement individuel de code privé), mais GitHub est toujours un endroit où aujourd'hui encore l'*open source* est créé et maintenu.

GitHub s'est doté de grandes aspirations avec une levée de fonds de capital-risque de 350 millions de dollars, même si l'entreprise était déjà rentable. Si GitHub assume pleinement son rôle d'administrateur du code *open source*, l'organisation peut avoir une énorme influence sur le soutien apporté à ces projets. Par exemple, elle peut créer de meilleurs outils de gestion de projets *open source*, défendre certaines catégories de licences, ou aider les gestionnaires de projets à gérer efficacement leurs communautés.

GitHub a subi de grosses pressions venant des développeurs qui gèrent certains projets, dont une lettre ouverte collective[38] intitulée « Cher GitHub », principalement issue de la communauté JavaScript. Cette lettre explique : « Beaucoup sont frustrés. Certains parmi nous qui déploient des projets très populaires sur GitHub se sentent totalement ignorés par vous. » La lettre inclut une liste de requêtes pour l'amélioration de produits, qui pourrait les aider à gérer plus efficacement leurs projets.

GitHub se confronte de plus en plus à des difficultés largement documentées dans les médias. Auparavant, l'entreprise était connue pour sa hiérarchie horizontale, sans aucun

37. Josh Constine, Gregory Ferenstein, « Facebook Doesn't Want To Be Cool, It Wants To Be Electricity », *Techcrunch*, 18/09/2013.
38. *Dear GitHub*, projet GitHub, 12/02/2016.

manager ni directive venant d'en haut. Les employés de GitHub avaient aussi la liberté de choisir de travailler sur les projets qu'ils souhaitaient[39]. Ces dernières années, tandis que GitHub s'est développée pour atteindre presque 500 employés, l'entreprise a axé sa stratégie vers une orientation plus commerciale en recrutant des équipes de vente et des dirigeants insérés dans un système hiérarchique plus traditionnel. Cette transition d'une culture horizontale vers une culture plus verticale s'est faite dans la douleur chez GitHub : au moins 10 dirigeants ont quitté l'organisation durant les quelques mois de l'hiver 2015-2016, ces départs incluant l'ingénieur en chef, le directeur des affaires financières, le directeur stratégique et le directeur des ressources humaines[40]. En raison de ces conflits internes, GitHub n'a toujours pas pris position publiquement pour jouer un rôle de promoteur de l'*open source* et assumer un *leadership* à même de résoudre les questions pressantes autour de l'*open source*, mais le potentiel est bel et bien là.

Pour le capital-risque, abordé précédemment, il y a un enjeu particulier dans l'avenir des infrastructures numériques. Comme les outils des développeurs aident les entreprises du secteur technologique à créer plus rapidement et plus efficacement, meilleurs sont les outils, meilleures sont les *startups*, meilleure sera la rentabilité du capital-risque. Néanmoins, l'infrastructure, d'un point de vue capitaliste, n'est en rien limitée à l'*open source* mais plus largement focalisée sur les plateformes qui aident d'autres personnes à créer. C'est pour cela que les investissements dans GitHub ou npm, qui sont des plateformes qui aident à diffuser du code source, ont un sens, mais tout aussi bien les investissements dans Slack, une plateforme de travail collaboratif que les développeurs peuvent utiliser pour construire des applications en ligne de commande connectées à la plateforme (à ce propos, le capital-risque a constitué un fonds de 80 millions dédié au

39. Chris Dannen, « Inside GitHub's Super-Lean Management Strategy-And How It Drives Innovation », *Fastcompany*, 18/10/2013.
40. Matt Weinberger, « GitHub is undergoing a full-blown overhaul as execs and employees depart – and we have the full inside story », *Business Insider*, 06/02/2016.

support de projets de développement qui utilisent Slack[41]). Même si le capital-risque apprécie les mécaniques sous-jacentes de l'infrastructure, il est limité dans ses catégories d'actifs : un capitaliste ne peut pas investir dans un projet sans modèle économique.

Enfin, les institutions universitaires ont joué un rôle historique éminent dans le soutien aux infrastructures numériques, tout particulièrement le développement de nouveaux projets. Par exemple, LLVM, un projet de compilateur pour les langages C et C++, a démarré en tant que projet de recherche au sein de l'université de l'Illinois, à Urbana-Champaign. Il est maintenant utilisé par les outils de développement de Mac OS X et iOS d'Apple, mais aussi dans le kit de développement de la Playstation 4 de Sony.

Un autre exemple, R, un langage de programmation répandu dans la statistique assistée par ordinateur et l'analyse de données, a été d'abord écrit par Robert Gentleman et Ross Ihaka à l'université d'Auckland[42]. R n'est pas uniquement utilisé par des entreprises logicielles comme Facebook ou Google, mais aussi par la Bank of America, l'Agence américaine des produits alimentaires et médicamenteux et le Service météorologique national américain, entre autres[43].

Quelques universités emploient également des programmeurs qui ont alors la liberté de travailler à des projets *open source*. Par exemple, le protocole d'heure réseau ou NTP (Network Time Protocol) utilisé pour synchroniser le temps *via* Internet, fut d'abord développé par David Mills, aujourd'hui professeur émérite de l'université du Delaware – le projet continuant à être maintenu par un groupe de volontaires conduit par Harlan Stenn. Bash, l'outil de développement dont nous parlions dans un chapitre précédent[44], est actuellement maintenu par Chet Ramsay, qui est employé par le département des technologies de l'information de l'université Case Western.

41. Heather Clancy, « Slack Starts $80 Million App Investment Fund », *Fortune*, 18/12/2015.
42. R project, page *Contributors*, sur r-project.org.
43. Voir *Companies using R in 2014*, sur revolutionanalytics.com.
44. Voir partie 3, chapitre 3.

Les institutions universitaires ont le potentiel pour jouer un rôle important dans le soutien de nouveaux projets, parce que cela coïncide avec leurs missions et types de donation, mais elles peuvent aussi manquer de la réactivité nécessaire pour attirer les nouveaux programmeurs *open source*. NumFOCUS[45] est un exemple d'une fondation 501(c3) qui soutient les logiciels scientifiques *open source* à travers des donations et parrainages financiers.

Le modèle de la fondation externe peut aider à fournir le soutien dont les logiciels scientifiques ont besoin dans le contexte de l'environnement universitaire. Les fondations Alfred P. Sloan et Gordon & Betty Moore expérimentent aussi des manières de connecter les institutions universitaires avec les mainteneurs de logiciels d'analyse des données, dans le but de soutenir un écosystème ouvert et durable[46].

45. Voir la page *Projects*, Numfocus.org.
46. Voir section « Tools and Software », sur msdse.org.

5

PERSPECTIVES

« Il y a un besoin de soutenir l'infrastructure publique libre. Mais il n'y a pas de source de revenu disponible à l'heure actuelle. Les individus se plaignent lorsque leurs horloges sont décalées d'une seconde. Ils disent, "oui nous avons besoin de vous, mais nous ne pouvons pas vous donner d'argent". »

Harlan Stenn, Network Time Protocol

ÉLABORER DES STRATÉGIES D'ASSISTANCE EFFICACES

Même si l'intérêt porté aux efforts pour soutenir les infrastructures numériques va croissant, les initiatives actuelles sont encore récentes, faites pour des cas particuliers, ou fournissent seulement un support partiel (comme le partage d'avantages fiscaux par des organisations à but non lucratif avec des groupes extérieurs à celles-ci).

Le développement de stratégies de soutien efficaces demande une compréhension fine de la culture *open source* qui caractérise une très grande partie de notre infrastructure numérique, mais aussi de reconnaître que beaucoup de choses ont changé dans les cinq dernières années, y compris la définition même de l'*open source*.

L'argent seul ne suffira pas à répondre aux problèmes d'un projet d'infrastructure en difficulté, parce que l'*open source* s'épanouit grâce aux ressources humaines et non financières. Il existe de multiples façons d'accroître les ressources humaines, comme répartir la charge entre plus de contributeurs ou encourager les entreprises à faire publier en *open source* une partie du travail de leurs employés. Une stratégie de soutien efficace doit inclure plusieurs façons de générer du temps et des ressources au-delà du financement direct du développement. Elle doit partir du principe que l'approche *open source* n'est pas défectueuse en elle-même, mais manque simplement de ressources.

Soutenir les infrastructures nécessite d'intégrer le concept d'intendance en lieu et place du concept de contrôle. Comme nous l'avons vu, les infrastructures numériques ne ressemblent pas aux infrastructures physiques. Elles sont réparties entre de multiples acteurs et organisations, avec des projets de toute forme et de toute taille, et il est difficile de prédire quels projets deviendront un succès ou qui y contribuera sur le long terme.

Cela en tête, voici quelques clés pour élaborer une stratégie d'assistance efficace :

- **Adopter la décentralisation, plutôt que s'y opposer :** les ressources de l'*open source* sont destinées à être partagées, c'est en partie ce qui leur donne autant d'impact. Utiliser la force que donne l'aspect communautaire comme un levier, plutôt que de recentraliser l'autorité.
- **Travailler étroitement avec les communautés informatiques existantes :** les communautés informatiques sont actives, soudées et savent se faire entendre. Faites appel à elles plutôt que de prendre une décision en aparté. Les voix les plus sonores des communautés agissent comme un signal de danger quand un problème nécessite d'être soulevé.
- **Envisager une approche globale du soutien aux projets :** les projets ont besoin de bien plus que du code ou de l'argent, parfois même ils n'ont besoin ni de l'un ni de l'autre. Le soutien sur le long terme est davantage une question de temps accordé que d'argent. La revue de code, la documentation technique, les tests de code, le soutien de la communauté, et la promotion du projet constituent un ensemble de ressources importantes.
- **Aider les mainteneurs de projets à anticiper :** aujourd'hui, les efforts pour soutenir l'infrastructure numérique ont surtout tendance à parer au plus pressé ou à demeurer ponctuels. En plus des projets existants, il existe sûrement de nouveaux projets qui ont besoin d'être lancés et accompagnés. Pour les projets existants, les mainteneurs trouveront un grand avantage à pouvoir planifier en vue des trois à cinq ans à venir, et pas seulement pour six mois ou un an.
- **Voir les opportunités, pas seulement les risques :** soutenir l'*open source* de nos jours, cela ne consiste pas uniquement à éviter les scénarios catastrophes (par exemple les failles de sécurité), mais plutôt à donner les moyens à plus de personnes de réaliser davantage de choses. Ce concept est une caractéristique essentielle de la culture *open source* actuelle et permet aussi de mettre en place un soutien pérenne. Tenez compte dans votre stratégie de la

façon dont vous pourriez accueillir davantage de personnes d'horizons, de compétences et de talents différents, plutôt que de limiter l'activité pour favoriser les personnes qui participent déjà.

David Heinemeier Hansson, le créateur de Ruby on Rails, compare l'*open source* à un récif de corail[1] :

> C'est un milieu plus fragile que vous ne le pensez, et il est difficile de sous-estimer la beauté qui est involontairement en jeu. Marchez avec précaution.

1. David Heinemeier Hansson, « The perils of mixing open source and money », blog personnel, 12/11/2013.

UNE ESQUISSE DU TABLEAU

Il est trop tôt pour dire à quoi devrait ressembler le soutien institutionnel à long terme d'un point de vue prospectif, mais il y a plusieurs domaines de travail critiques qui peuvent nous aider à le déterminer. Les propositions suivantes se rattachent à trois domaines :

- **traiter les infrastructures numériques comme un bien commun essentiel** et les élever au rang d'acteur intersectoriel clé ;
- **travailler avec des projets** pour améliorer les standards, la sécurité et les flux de production ;
- **augmenter la taille du groupe de contributeurs** de manière à ce que davantage de personnes, et encore plus de personnes de types différents, puissent élaborer et soutenir ensemble les logiciels publics.

Conscientiser et éduquer les acteurs clés

Comme nous l'avons relevé dans cette étude, beaucoup d'acteurs clés – dont les *startups*, les gouvernements, et les sociétés de capital-risque – pensent à tort que les logiciels publics « fonctionnent, tout simplement » et ne requièrent pas de maintenance supplémentaire. Pour entretenir correctement l'écosystème de nos infrastructures numériques, ces populations devraient être les premières à être informées du problème. Les infrastructures numériques ont besoin de porte-parole qui soient affranchis de toute contrainte politique ou commerciale et qui puissent comprendre et communiquer les besoins de l'écosystème.

Traiter les infrastructures numériques comme des biens communs essentiels pourrait également motiver l'investissement direct dans la construction de meilleurs systèmes en

partant de zéro. Par exemple, aux États-Unis, les autoroutes inter-États et le réseau de bibliothèques publiques furent dès l'origine conçus comme des ressources collectives. Les unes et les autres ont eu leur champion (respectivement le président Dwight D. Eisenhower et le philanthrope Andrew Carnegie) qui ont clairement argumenté en faveur du bénéfice social et financier qui résulterait de tels projets.

Un réseau national d'autoroutes ne sert pas uniquement à nous relier en tant qu'individus en facilitant les déplacements d'un endroit à un autre, mais il apporte aussi la prospérité financière dans tous les coins du pays, grâce à l'usage commercial des voies rapides pour acheminer les marchandises. Dans les bibliothèques Andrew Carnegie, publiques et gratuites, les livres étaient accessibles et non stockés en magasin, pour permettre aux personnes de les feuilleter et d'y trouver elles-mêmes l'information sans avoir recours à un(e) bibliothécaire. Cette pratique a aidé à démocratiser l'information et à donner aux gens les moyens de s'éduquer eux-mêmes.

Une meilleure éducation et une meilleure prise de conscience pourraient s'étendre jusqu'aux gouvernements, dont certains ont rendu, par la loi, les infrastructures numériques difficiles à soutenir et qui ne sont peut-être pas familiers des normes culturelles et de l'histoire de l'*open source*. Aux États-Unis, l'IRS (Internal Revenue Service, service des impôts américain) a une définition très restrictive des activités caritatives, et comme l'*open source* est mal comprise, son impact positif sur la société demeure ignoré[1]. Cela complique l'institutionnalisation de plus gros projets à travers une fondation ou une association professionnelle.

Mesurer l'utilisation et l'impact de l'infrastructure numérique

L'impact de l'infrastructure numérique est encore très difficile à mesurer. Les indicateurs habituels sont soit très imprécis, soit simplement indisponibles. Ce n'est pas un problème facile à

1. Voir partie 4, chapitre 4.

résoudre. Mais sans données relatives aux outils utilisés et à notre dépendance vis-à-vis d'eux, il est difficile de se faire une idée précise de l'état des sous-financements.

Avec de meilleurs indicateurs, nous pourrions décrire l'impact économique de l'infrastructure numérique, identifier les projets essentiels qui manquent de financement et comprendre les dépendances entre les projets et les personnes. Pour le moment, il est impossible de dire qui utilise un projet *open source* à moins que l'utilisateur, individu ou entreprise, ne le révèle. Pour déterminer quel projet a besoin de plus de soutien, nous ne disposons que d'informations anecdotiques.

De meilleures statistiques pourraient également nous aider à identifier les « contributeurs clés de voûte ». En biologie environnementale, une « espèce clé » est une espèce animale qui a un impact disproportionné sur son environnement au regard de ses effectifs[2]. Dans la même idée, un « contributeur clé » pourrait être un développeur qui contribue à plusieurs projets essentiels, le seul responsable d'un projet crucial, ou généralement perçu comme influent et digne de confiance. Les « contributeurs clés » sont des défenseurs essentiels, les valoriser en leur fournissant les ressources dont ils ont besoin pourrait améliorer le système dans son ensemble. Comprendre les relations entre les communautés *open source* et les « contributeurs clés » pourrait aider à identifier rapidement les secteurs qui auront besoin de soutien supplémentaire.

On dispose également de peu de données sur les contributeurs eux-mêmes : qui contribue à l'*open source*, quelles conditions leur permettent de le faire, et quelles sont les contributions effectuées. Les femmes, les non-anglophones et les nouveaux contributeurs à l'*open source* sont des exemples de population qui devraient être suivis dans le temps, en particulier pour mesurer l'impact des programmes de soutien.

Les seules statistiques disponibles sur les dépôts GitHub sont le nombre de personnes ayant *étoilé* (action semblable à *liker*), vu (c'est-à-dire qu'elles reçoivent des nouvelles du projet) ou « *forké* » un projet. Ces chiffres permettent de fournir

2. Pour en savoir plus sur les espèces clés, voir l'article « Espèce clé de voûte » sur Wikipédia.

des indicateurs concernant la popularité, mais ils peuvent être trompeurs. Beaucoup de personnes peuvent étoiler un projet, parce qu'il a une conception intéressante par exemple, sans toutefois l'intégrer à leur propre code.

Certains gestionnaires de paquets tel npm (qui est celui de Node.js) suivent les téléchargements. Le « *popularity contest* » de Debian piste les téléchargements du système d'exploitation libre Debian. Néanmoins, chaque gestionnaire de paquets est limité à un écosystème particulier, et aucun de ces gestionnaires ne peut donner une image du système dans son ensemble. Plusieurs projets ne sont pas inclus dans un gestionnaire de paquets et ne sont pas suivis. Libraries.io, un site web créé par Andrew Nesbitt, est une tentative pour agréger des données des projets *open source* en fonction de leur usage, il piste environ 1,3 million de bibliothèques *open source* sur 32 gestionnaires de paquets[3].

Travailler avec les projets pour moderniser l'organisation de travail

Beaucoup de projets peinent et pas seulement à cause d'un manque de financement, mais aussi parce qu'il est difficile d'y contribuer, ou encore parce qu'il existe un goulot d'étranglement au niveau des mainteneurs qui traitent les demandes de modification (*pull requests*) de la communauté. C'est vrai, en particulier, pour les plus anciens projets qui ont été bâtis avec des outils de développement, des langages et des processus qui ne sont plus aussi répandus (ceux qui par exemple utilisent un système de contrôle de version autre que Git, dont la popularité croît chez les développeurs).

On peut faire beaucoup de choses pour faciliter la contribution à un projet, depuis la migration vers un flux de travail (*workflow*) plus moderne, le nettoyage du code, la fermeture des *pull requests* délaissées, jusqu'à la mise en place d'une politique claire pour les contributions. Certains projets ont essayé

3. Ces chiffres relevés sur le site Libraries.io, sont constamment mis à jour. En janvier 2017, on compte environ 2,1 millions de bibliothèques *open source* recensées sur 33 gestionnaires de paquets.

de rendre les contributions plus simples. Le développeur Felix Geisendörfer, par exemple, a suggéré que chaque personne qui soumet une modification du code devrait avoir une permission de *commit* afin de réduire l'engorgement au niveau de l'unique mainteneur vérifiant et approuvant ces changements. Felix a estimé que « cette approche est un moyen fantastique d'éviter que le projet ne se ratatine en transformant le projet d'un seul homme en celui d'une communauté »[4].

Le règlement de contribution de Node.js, qui peut être adopté par les autres projets Node, met l'accent sur l'augmentation du nombre de contributeurs et sur leur autonomisation dans la prise de décision, plutôt que de désigner les mainteneurs comme seule autorité approbatrice. Leurs règles de contribution expliquent comment soumettre et valider des *pull requests*, comment consigner des bugs, etc. Les mainteneurs Node.js ont constaté qu'adopter de meilleures règles les avait aidés à gérer leur charge de travail et à faire évoluer leur communauté vers un projet plus sain et actif[5]. Des études sont à réaliser pour déterminer quels projets doivent avancer en priorité. Autrement dit, définir à quoi ressemble un « projet à succès », aussi bien en termes de financement et de modèles de gouvernance, que dans l'équilibre à trouver entre mainteneurs, contributeurs et usagers ! La réponse peut varier en fonction des différents types de projets et de leur ampleur.

Encourager les standards communs dans les projets *open source*

Bien que GitHub soit en train de devenir une plateforme standard pour la collaboration sur le code, de nombreux aspects des projets *open source* ne sont pas encore standardisés, notamment l'ampleur et la richesse de la documentation, des licences et des guides de contribution, ainsi que le style de code et le formatage.

4. Voir Felix Geisendörfer, « The Pull Request Hack », sur felixge.de (site personnel), 11/03/2013.
5. Voir Mikeal, « Healthy Open Source », Medium.com, 22/02/2016.

Encourager l'adoption de standards de projets pourrait faciliter, pour les mainteneurs, la gestion des contributions, tout en réduisant pour les contributeurs les obstacles à la participation.

Parmi les exemples de standardisation croissante, on trouve le code de conduite, qui est un règlement détaillant les attentes en termes d'attitude et de communication.

Ces dernières années, des codes de conduite ont été adoptés par un nombre croissant de communautés de projets, notamment Node.js, Django et Ruby. Bien que le processus d'adoption ait pu donner lieu à d'intenses débats au sein de certaines communautés, leur prolifération révèle un intérêt croissant pour la responsabilisation du comportement des communautés.

Augmenter le nombre de contributeurs et contributrices *open source*

Comme nous l'avons évoqué dans un chapitre précédent de cette étude, l'industrie du logiciel est florissante, avec un nombre croissant de nouveaux développeurs mais aussi d'autres talents variés : il y a du travail à faire pour encourager ces nouveaux arrivants à contribuer à l'*open source*. Augmenter le nombre de contributeurs permet aux projets *open source* d'être plus durables, car davantage de personnes participent à leur développement. Permettre à encore plus de personnes de contribuer à l'*open source* accroît également l'empathie et la communication entre les « utilisateurs » de l'*open source* et les projets dont ils dépendent.

Your First PR[6] est un exemple d'initiative, développée par la programmeuse Charlotte Spencer, qui aide les nouveaux venus à effectuer leur première contribution à l'*open source*. First Timers Only[7] (Réservé aux débutants) et Make a Pull Request[8] (Faites une *pull request*) sont deux autres exemples de ressources populaires qui initient les néophytes à l'*open source*. Certains projets *open source* utilisent également des étiquettes telles que *first*

6. Voir le compte Twitter Your First PR@yourfirstpr. « Votre première PR », PR pour *pull request*, (NdT).
7. Voir Firsttimersonly.com.
8. Voir Makeapullrequest.com.

bug ou *contributor friendly* pour signaler les problèmes susceptibles d'être résolus par des contributeurs moins expérimentés. Il serait également bénéfique d'encourager les contributions à l'*open source* autres que le code, comme la rédaction de documentation technique, la gestion des tâches et des flux de travail, ou la création d'un site internet pour le projet.

En plus d'accroître la proportion de techniciens talentueux contribuant à l'*open source*, se présente l'opportunité de puiser dans un vivier plus large de contributeurs. Faire en sorte que les non-anglophones se sentent bienvenus dans les communautés *open source*, par exemple, pourrait rendre la technologie plus accessible à travers le monde. Et comme beaucoup de recruteurs utilisent les travaux *open source* comme un portfolio au moment d'embaucher un développeur, une communauté *open source* plus diverse encouragerait l'apparition d'un personnel technique globalement plus intégré.

Améliorer les relations entre projets et acteurs extérieurs

Les entreprises sont un élément incontournable de l'écosystème *open source*, et leur rôle ne fait que gagner en importance à mesure qu'un plus grand nombre d'entre elles adoptent les logiciels *open source*. Faciliter la collaboration entre entreprises et projets, ainsi qu'aider les entreprises à comprendre les besoins des communautés *open source*, pourrait débloquer le soutien des entreprises susceptibles de devenir mécènes ou promoteurs de l'*open source*.

Selon l'étude annuelle des entreprises *open source* réalisée par Black Duck[9], seulement 27 % d'entre elles ont un règlement formel concernant les contributions de leurs employés à l'*open source*. Clarifier la possibilité ou non pour les employés de contribuer à l'*open source* sur leur temps de travail, et les encourager à le faire, pourrait grandement améliorer le soutien des entreprises aux projets *open source*.

9. Voir le rapport de la société BlackDuck *The Ninth Annual Future of Open Source Survey*, sur blackducksoftware.com, 2015.

En 2014, un groupement d'entreprises a fondé le TODO Group[10], pour partager les bonnes pratiques autour de la participation corporative à l'*open source*. Parmi les membres de ce groupe, on trouve Box, Facebook, Dropbox, Twitter et Stripe. En mars 2016, le TODO Group a annoncé qu'il serait hébergé par la Fondation Linux en tant que projet collaboratif[11].

Les entreprises peuvent également fournir un soutien financier aux projets, mais il est parfois difficile pour elles de trouver comment formaliser leur mécénat. Créer des budgets dédiés au sponsoring en direction des équipes d'ingénieurs ou des employés, ou encore créer des documents permettant aux projets de « facturer » plus facilement leurs services aux entreprises, sont autant d'initiatives qui pourraient augmenter les contributions financières à l'*open source*.

Poul-Henning Kamp, par exemple, travaille sur un projet *open source* nommé Varnish[12], utilisé par un dixième des sites les plus visités d'Internet, notamment Facebook, Twitter, Tumblr, *The New York Times* et *The Guardian*. Pour financer ce travail, il a créé la Varnish Moral License pour faciliter la subvention du projet par les entreprises.

Même si en pratique la relation est un mécénat, Poul-Henning utilise une terminologie familière aux entreprises, avec des termes tels que « facturation » et « licences », pour réduire les obstacles à la participation[13].

Augmenter le soutien aux compétences diverses et aux fonctions hors codage

Dans un passé pas si lointain, les *startups* de logiciels étaient fortement centrées sur les compétences techniques. Les autres rôles, comme le marketing et le design, étaient considérés comme secondaires par rapport au code. Aujourd'hui, avec la création et la consommation rapide de logiciels, cette

10. Voir Todogroup.org.
11. Voir TheTodoGroup, « TODO Becomes A Linux Foundation Collaborative Project », sur le site Todo.org, 30/03/2016.
12. Pour en savoir plus sur Varnish, voir l'article « Varnish » sur Wikipédia.
13. Voir Poul-Henning Kamp, « The Varnish Moral License », sur la page personnelle de l'auteur, 30/06/2015.

conception ne tient plus. Les *startups* sont en concurrence pour capter l'attention de leurs clients. L'identité de la marque est devenue l'un des principaux facteurs de différenciation.

Ces cinq dernières années ont été celles de l'essor du développeur *full stack* (polyvalent) : des développeurs plus généralistes que spécialisés, capables de travailler sur plusieurs domaines d'un logiciel complexe, et qui sont susceptibles d'avoir des compétences dans la conception et la production. Les équipes de développement sont plus soudées, elles utilisent les méthodes agiles avec des approches de conception d'architecture logicielle (où le livrable est élaboré en faisant des navettes entre les équipes de techniciens, designers et commerciaux), plutôt qu'une approche en cascade (où chaque équipe apporte sa pièce à l'édifice avant de la transmettre au groupe de travail suivant).

Les logiciels *open source* ont connu peu d'évolutions de ce genre, malgré notre dépendance croissante vis-à-vis d'eux. On comprend aisément que le code soit au cœur d'un projet *open source*, il est en quelque sorte le « produit final » ou le livrable. Les fonctions relatives à la gestion de la communauté, à la documentation ou à la promotion du projet, qui sont la marque d'une organisation saine et durable, sont moins valorisées. Il en découle que les projets sont déséquilibrés. Beaucoup de choses pourraient être entreprises pour financer et soutenir les contributions autres que le code, des dons en nature pour payer les serveurs par exemple, ou des avantages comme une assurance maladie. Disposer de soutiens de ce type permettrait de réduire notablement la charge des développeurs.

À LA CROISÉE DES CHEMINS

L'état actuel de notre infrastructure numérique est un des problèmes les moins bien compris de notre temps. Il est vital d'en prendre conscience.

En s'investissant bénévolement dans notre structure sous-jacente, les développeurs ont facilité la construction de logiciels pour autrui. En la fournissant gratuitement plutôt qu'en la facturant, ils ont alimenté une révolution de l'information.

Les développeurs n'ont pas fait cela par altruisme. Ils l'ont fait car c'était la meilleure manière de résoudre leurs propres problèmes. L'histoire des logiciels *open source* est l'un des grands triomphes actuels du bien commun.

Nous avons de la chance que les développeurs aient limité les coûts cachés de ces investissements. Mais leurs investissements initiaux ne nous ont amenés que là où nous sommes aujourd'hui.

Nous ne sommes qu'au commencement de l'histoire de la transformation de l'humanité par le logiciel. Marc Andreessen, cofondateur de Netscape et reconnu comme capital-risqueur derrière la société Andreessen Horowitz, remarque en 2011 que « le logiciel dévore le monde »[1]. Depuis lors, cette pensée est devenue un mantra pour l'ère moderne.

Le logiciel touche tout ce que l'on fait : non seulement les frivolités et les loisirs, mais aussi nos obligations et ce qui est crucial. OpenSSL, le projet décrit au début de cet essai[2], le démontre bien. Dans une interview téléphonique, Steve Marquess explique qu'OpenSSL n'était pas employé seulement par les utilisateurs de sites web, mais aussi par les gouvernements, les drones, les satellites et tous « les gadgets que vous entendez bipper dans les hôpitaux ».

1. Voir Marc Andreessen, « Why Software Is Eating The World », *The Wall Street Journal*, 20/08/2011.
2. Voir l'introduction de l'ouvrage.

Le Network Time Protocol[3], maintenu par Harlan Stenn, synchronise les horloges utilisées par des milliards de périphériques connectés et touche tout ce qui contient un horodatage. Pas seulement les applications de conversations ou les courriels, mais aussi les marchés financiers, les enregistrements médicaux et la production de produits chimiques.

Et pourtant, Harlan note[4] :

> Il y a un besoin de soutenir l'infrastructure publique libre. Mais il n'y a pas de source de revenu disponible à l'heure actuelle. Les individus se plaignent lorsque leurs horloges sont décalées d'une seconde. Ils disent, « oui nous avons besoin de vous, mais nous ne pouvons pas vous donner d'argent ».

Durant ces cinq dernières années, l'infrastructure *open source* est devenue une couche essentielle de notre tissu social. Mais tout comme les *startups* ou la technologie elle-même, ce qui a fonctionné pour les trente premières années de l'histoire de l'*open source* n'aidera plus à avancer. Pour maintenir notre rythme de progression, nous devons réinvestir dans les outils qui nous aident à construire des projets plus importants et de meilleure qualité.

Trouver un moyen de soutenir l'infrastructure numérique peut sembler intimidant, mais il existe de multiples raisons de voir le chemin à parcourir comme une opportunité.

Premièrement, l'infrastructure est déjà là, avec une valeur clairement démontrée. Ce rapport ne propose pas d'investir dans une idée sans plus-value. L'énorme contribution sociale de l'infrastructure numérique actuelle ne peut être ignorée ni mise de côté, comme cela est déjà arrivé dans des débats tout aussi importants sur les données, la vie privée, la neutralité du net ou l'opposition entre investissement privé et investissement public. Il est dès lors plus facile de faire basculer les débats vers des solutions.

3. Protocole de temps par le réseau (NdT).
4. Voir Charles Babcock, « NTP's Fate Hinges On "Father Time" », *InformationWeek*, 03/11/2015.

Deuxièmement, il existe déjà des communautés *open source* engagées et prospères avec lesquelles travailler. De nombreux développeurs s'identifient par le langage de programmation qu'ils utilisent (tel que Python ou JavaScript), la fonction qu'ils apportent (telle qu'analyste ou développeur opérationnel), ou un projet important (tel que Node.js ou Rails). Ce sont des communautés fortes, visibles et enthousiastes.

Les constructeurs de notre infrastructure numérique sont connectés les uns aux autres, attentifs à leurs besoins, et techniquement talentueux. Ils ont déjà construit notre ville ; nous avons seulement besoin d'aider à maintenir les lumières allumées, de telle sorte qu'ils puissent continuer à faire ce qu'ils font de mieux.

Les infrastructures, qu'elles soient physiques ou numériques, ne sont pas faciles à comprendre, et leurs effets ne sont pas toujours visibles, mais cela doit nous encourager à les suivre de plus près, et non l'inverse. Quand une communauté a parlé si ouvertement et si souvent de ses besoins, tout ce que nous devons faire est d'écouter.

GLOSSAIRE

Bibliothèque (logicielle)

Les bibliothèques sont des fonctions préfabriquées qui accélèrent l'écriture du code d'un logiciel, tout comme une entreprise du bâtiment achète des fenêtres préfabriquées au lieu de les assembler à partir des composants de base. Par exemple, les développeurs et développeuses peuvent utiliser une bibliothèque appelée OAuthau plutôt que développer leur propre système d'identification pour les utilisateurs de leur application.

Capital-risque

Un type d'investissement privé qui permet de financer de jeunes entreprises à forte croissance en échange de titres. Le capital-risque a participé à la croissance de nombreux aspects commerciaux d'Internet et permis la montée en puissance de la Silicon Valley.

Code source

Pour les besoins de cette étude, le code source correspond au code effectivement associé à un projet *open source*.

Contributeur, contributrice

Personne qui fait une contribution à un *open source* actif. Parmi les exemples de contribution, on peut mentionner : écrire du code, rédiger de la documentation, s'occuper de l'assistance aux

utilisateurs, etc. Un contributeur n'a pas forcément un droit de commit sur le projet (par exemple, quelqu'un d'autre doit approuver ou non ses contributions avant de les mettre en production).

Dépôt (logiciel)

Lieu de stockage du code source nécessaire pour utiliser un projet informatique. Par exemple, GitHub offre un espace en ligne pour héberger les dépôts de chacun et permettre à d'autres utilisateurs de les trouver et de les utiliser. L'ensemble des fichiers de code source est appelé communément une « base de code ».

Documentation (de logiciel)

Information écrite qui explique comment les personnes peuvent utiliser ou contribuer à un projet logiciel. La documentation est comme un manuel d'instruction pour le logiciel. Sans elle, un développeur ne saurait pas comment utiliser au mieux le projet.

FLOSS

Acronyme de Free, Libre, and Open Source Software, logiciel *open source* libre et gratuit.

Fork (scission)

Il existe deux types de *forks*. Historiquement, un *fork* de projet est une copie d'un projet *open source* que l'on continue de développer séparément. Son usage est presque politique, par exemple lorsqu'il y a un désaccord interne à propos de la direction du projet, ou pour imposer des modifications substantielles au projet d'origine (idéalement, en réintégrant en retour ces

modifications au projet principal). Un *fork* sur GitHub fait référence à une copie temporaire d'un projet pour réaliser des modifications, généralement avec l'intention de les réintégrer ensuite dans le projet principal. Cela n'a pas la connotation politique et sociale d'un *fork* de projet. GitHub a remanié ce terme pour encourager une culture de bidouillage léger qui prévaut parmi les contributeurs *open source* d'aujourd'hui.

FOSS

Acronyme de Free and Open Source Software, logiciel *open source* gratuit.

Framework

Les *frameworks* offrent une base, une sorte d'échafaudage, une structure. Imaginez cela comme un schéma pour toute une application. Comme un plan, un *framework* définit la manière dont l'application se comportera sur mobile, ou comment ses données seront sauvegardées dans une base de données. Rails et Django sont des exemples de *frameworks*. Selon Wikipédia, « les *frameworks* sont acquis par les informaticiens, puis incorporés dans des logiciels applicatifs mis sur le marché, ils sont par conséquent rarement achetés et installés séparément par un utilisateur final ».

GitHub

Plateforme commerciale d'hébergement de code. GitHub a été lancée en 2008 et c'est actuellement la plateforme la plus populaire pour héberger du code et collaborer à des projets *open source* (il est également possible de stocker du code privé sur GitHub). GitHub a aidé à standardiser les pratiques dans le développement *open source* et a élargi l'audience de l'*open source*. La gestion des projets sur GitHub se fait grâce au système de contrôle de version Git.

Infrastructure numérique

Pour les besoins de cet ouvrage, l'infrastructure numérique désigne l'ensemble des ressources logicielles publiques qui sont utilisées pour créer des logiciels destinés à un usage personnel ou commercial. On peut prendre l'exemple des langages de programmation ou celui des bases de données. Cette définition n'inclut pas les éléments d'infrastructure physique nécessaires à la création de logiciel (comme les serveurs physiques, les câbles…).

Langage (de programmation)

Les langages de programmation sont l'ossature de communication des logiciels. Ils permettent aux différents composants logiciels de réaliser des actions et d'interagir entre eux. On peut citer parmi les langages les plus populaires le JavaScript, le Python et le C.

Logiciel libre

Logiciel que l'on peut librement exécuter pour n'importe quel objectif (commercial ou non) aussi bien qu'étudier, modifier et distribuer. Le terme a été créé aux alentours de 1983, à partir des travaux de l'informaticien Richard Stallman, du projet GNU, et de la Free Software Foundation (fondée en 1985).

Mainteneur (*open source*)

Personne qui endosse la responsabilité d'un projet *open source*. La fonction varie selon les projets. Parfois, les mainteneurs sont formellement nommés ; et parfois, ceux qui abattent le plus gros du travail le deviennent *de facto*. Un mainteneur porte probablement la charge de la gestion du projet plus qu'aucun autre contributeur. Il peut être ou non l'auteur de la version

initiale du projet. Il y a de fortes chances qu'il ait un accès *commit* au projet, c'est-à-dire qu'il peut y effectuer directement des modifications.

Open source (logiciel)

Techniquement, un logiciel *open source* est la même chose qu'un logiciel libre (voir ci-dessus). Néanmoins, d'un point de vue culturel, l'*open source* tend à souligner les bénéfices pratiques des logiciels publics, alors que le logiciel libre est davantage un mouvement social.

Le terme *open source* a vu le jour en 1998, au cours d'une réunion entre personnes qui recherchaient une alternative au terme *free software* qui soit plus en phase avec l'entreprise.

OSS

Acronyme de Open Source Software, logiciel *open source*.

REMERCIEMENTS

Merci à tous ceux qui ont courageusement accepté d'être mentionnés dans cet ouvrage, ainsi qu'à ceux dont les réponses honnêtes et réfléchies m'ont aidée à affiner ma pensée pendant la phase de recherche : André Arko, Brian Behlendorf, Adam Benayoun, Juan Benet, Cory Benfield, Kris Borchers, John Edgar, Maciej Fijalkowski, Karl Fogel, Brian Ford, Sue Graves, Eric Holscher, Brandon Keepers, Russell Keith-Magee, Kyle Kemp, Jan Lehnardt, Jessica Lord, Steve Marquess, Karissa McKelvey, Heather Meeker, Philip Neustrom, Max Ogden, Arash Payan, Stormy Peters, Andrey Petrov, Peter Rabbitson, Mikeal Rogers, Hynek Schlawack, Boaz Sender, Quinn Slack, Chris Soghoian, Charlotte Spencer, Harlan Stenn, Diane Tate, Max Veytsman, Christopher Allan Webber, Chad Whitacre, Meredith Whittaker, Doug Wilson.

Merci à tous ceux qui ont écrit quelque chose de public référencé dans cet essai. C'était une partie importante de ma recherche et je remercie ceux dont les idées sont publiques pour que d'autres s'en inspirent.

Merci à Franz Nicolay pour la relecture et Brave UX pour le design.

Enfin, un très grand merci à Jenny Toomey et Michael Brennan pour m'avoir aidée à conduire ce projet avec patience et enthousiasme, à Lori McGlinchey et Freedman Consulting pour leurs retours et à Ethan Zuckerman pour que la magie opère.

À PROPOS DE LA TRADUCTION FRANÇAISE…

Framasoft[1] est une association œuvrant pour la promotion et la diffusion du logiciel libre et de la culture libre dans une démarche d'éducation populaire. Parmi ses projets, Framalang a pour objectif de traduire et diffuser des articles sur le blog de l'association et des ouvrages publiés dans la collection Framabook.

Cette traduction de l'ouvrage de Nadia Eghbal a été diffusée dans une première version sur le Framablog[2] entre septembre 2016 et janvier 2017. Un partenariat entre Framasoft et OpenEdition a permis la co-édition de ce livre.

Nous remercions chaleureusement les traducteurs et traductrices du groupe Framalang : Adélie, AFS, alien spoon, Anthony, Asta (Gatien Bovyn), astraia_spica, Bam92 (Abel Mbula), Bidouille, Bromind (Martin Vassor), Ced, dominix, Edgar Lori, flo, glissière de sécurité, goofy, goudron, Julien / Sphinx, jums, Laure, Luc, Lumibd, lyn, Mika, MO, Opsylac (Diane Ranville), pasquin, Penguin, peupleLà, Piup, roptat, Rozmador, salade, serici, teromene, Théo, urlgaga, woof, xi (Juliette Tibayrenc), xXx.

Nous remercions également Solenne Louis pour sa relecture attentive de la traduction initiale.

1. https://framasoft.org.
2. https://framablog.org.

TABLE DES MATIÈRES

Réalisé sur les presses de l'imprimerie Ciaco
en impression à la demande
depuis le mois de septembre 2017
à Louvain-la-Neuve, Belgique
N° d'imprimeur : 89470