UNIVERSITÉ DU QUÉBEC À MONTRÉAL

L'INNOVATION COLLECTIVE AU SEIN D'UNE COMMUNAUTÉ OPEN SOURCE : LE CAS DE LA COMMUNAUTÉ PYTHON

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN SCIENCE, TECHNOLOGIE ET SOCIÉTÉ

PAR
BRUNO SCHMITT-CORNET

FÉVRIER 2016

À mes parents,

REMERCIEMENTS

En préambule à ce mémoire de recherche, je souhaite adresser mes remerciements à toutes les personnes qui ont permis la réalisation de cette étude. À commencer par Florence Millerand et Guillaume Latzko-Toth, mes codirecteurs de recherche. Je tiens à vous remercier pour votre confiance en mes capacités et la patience dont vous avez fait preuve durant toute la rédaction de ce mémoire. Vous avez été pour moi une grande source d'inspiration et de motivation. Vos commentaires avisés m'ont grandement aidé dans ma réflexion et dans la poursuite de mes objectifs.

Je tiens également à remercier la direction de la maîtrise en science, technologie et société par l'intermédiaire de Catherine Chartré grâce à qui j'ai pu être orienté vers ma direction de recherche et dans mes démarches administratives.

Mes remerciements s'adressent également à mes parents, amis et collègues pour leur soutien et leurs encouragements tout au long de la réalisation de ce mémoire de recherche.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

BDFL	Benevolent Dictator For Life
F/OSS	Free/Open Source Software
FSF	Free Software Foundation
GNU	GNU's Not Unix
OSI	Open Source Initiative
PEP	Python Enhancement Proposal
PSF	Python Software Foundation
PyCon	Python Conference

RÉSUMÉ

Ce mémoire porte sur l'organisation sociale de la communauté Python et sur le processus d'innovation technique qui constitue le cœur de ses activités. Notre problématique s'articule autour du paradoxe apparent que représente la production d'un logiciel innovant et de haut niveau par une communauté de développeurs bénévoles, selon des modalités très éloignées des processus organisationnels qui régissent l'innovation industrielle. Nous cherchons donc à comprendre leur organisation et leurs modes de fonctionnement interne. La question qui sous-tend cette étude est celle-ci : en quoi la communauté Python constitue-t-elle une communauté innovante et quels mécanismes a-t-elle mis en place pour structurer l'action collective et soutenir le processus d'innovation collective ? Sur le plan théorique, nous mobilisons trois approches complémentaires pour caractériser la communauté Python : 1) la théorie des *communautés de pratique*, qui propose un cadre d'analyse des communautés d'individus liés ensemble par une pratique commune, ici le développement logiciel ; 2) le concept de *communauté épistémique* pour ce qui a trait à la production des connaissances au sein de la communauté Python ; 3) la théorie de *l'innovation par l'utilisateur* et la notion d'utilisateur-pionnier, pour comprendre la manière dont l'innovation est gérée par la communauté. La méthodologie retenue dans le cadre de ce mémoire est qualitative et repose sur une démarche inductive consistant en une observation non participante, suivant les principes de l'ethnographie virtuelle, de la communauté Python, et ce, à travers les outils de communication qu'elle met en œuvre, à savoir des sites web et des listes de discussion. Les résultats de cette étude font état d'une diversité de rôles et de statuts au sein de la communauté et d'une organisation fortement hiérarchisée. D'autre part, la communauté est organisée en un réseau horizontal d'innovation ; l'effort d'innovation est en effet mis en œuvre par des utilisateurs-pionniers appelés « champions » qui écrivent du code et font revoir et valider collectivement leur production sur les listes de discussion de la communauté. Ce mode opératoire très structuré – et formalisé sous le nom de « proposition d'amélioration du langage » (ou PEP) – constitue une innovation organisationnelle qui marque l'originalité de la communauté Python non seulement dans le monde du logiciel libre, mais aussi plus largement dans le domaine du développement logiciel.

Mots-clés : Python, communauté, réseau, innovation, utilisateurs.

INTRODUCTION

Le projet de cette recherche est né d'une interrogation personnelle, apparue lorsque j'étais moi-même abonné aux listes de discussion de la communauté Python et alors utilisateur du langage. Existait-il des études sur le phénomène que j'observais : l'organisation en ligne d'une communauté qui menait à bien le développement collectif d'un langage de programmation ? Au-delà de l'intérêt technique, c'était un nouveau domaine que je souhaitais explorer : celui du fonctionnement d'une communauté de praticiens ayant pour objectifs la création et la maintenance d'un logiciel innovant. Poussé par la curiosité et la volonté de comprendre la communauté Python – et plus largement le monde du logiciel libre –, je me suis intéressé à la technique sous un nouveau jour, c'est-à-dire sous l'angle de la dynamique sociale de sa mise en œuvre et de son développement.

Ce mémoire porte donc sur la communauté Python en tant qu'organisation sociale et comporte cinq parties distinctes. Dans un premier chapitre, nous allons problématiser le phénomène étudié en partant de l'origine du logiciel libre et de l'*open source* et en retraçant les courants de pensée qui les caractérisent jusqu'à délimiter les contours de cette recherche. Dans un deuxième temps, nous présentons le cadre théorique de la recherche composé de deux approches principales : la théorie des communautés de pratique et la théorie de l'innovation par l'utilisateur. Dans un troisième chapitre, nous présentons la démarche méthodologique utilisée dans le cadre de cette étude, à savoir l'observation non participante en ligne de type ethnographique de la communauté Python à travers les outils de communication, sites web et listes de

discussion qu'elle met en œuvre. Le quatrième chapitre, dédié à la présentation de la communauté Python, vise à décrire la communauté et son mode de fonctionnement, notamment les modalités de prise de décision et les différents rôles et statuts qui la composent, et ce, à la lumière de la littérature mobilisée. Dans le cinquième et dernier chapitre, dédié au processus d'innovation de la communauté, nous présentons les *propositions d'amélioration du langage* comme principal outil et vecteur de l'innovation collective par les utilisateurs au sein de la communauté. Enfin, nous concluons ce mémoire par le rappel des différents résultats marquants de cette recherche, la présentation des limites de cette dernière et une ouverture sur les pistes de recherche en vue d'études futures.

CHAPITRE I

PROBLÉMATIQUE

Dans ce premier chapitre, nous présentons l'objet de cette recherche. Nous mettons en situation notre questionnement et abordons différentes composantes de la problématique. Nous procédons à une revue de la littérature et nous précisons notre question de recherche centrale, ainsi que les questions spécifiques sous-jacentes. Nous concluons ce chapitre en proposant nos objectifs de recherche et en explicitant la pertinence sociologique de notre étude.

1.1 La naissance d'un projet technologique

L'ordinateur personnel s'est progressivement installé dans les foyers américains, depuis le milieu des années 1970, avec l'apparition des premiers microprocesseurs et micro-ordinateurs. Dès lors, de jeunes férus d'électronique vont tester ces machines, travailler à faire fonctionner ces matériels (*hardware*), souvent acquis à prix d'or, avec des programmes informatiques (*software*) inédits écrits par leur soin. Se soutenant mutuellement, ils vont se regrouper en clubs et forger les premières communautés d'utilisateurs, par exemple au sein du Homebrew Computer Club en Californie (Levy, 2013 ; Broca, 2013).

Ces technophiles d'alors, appelés *hackers*[1], vont partager leurs connaissances techniques pointues, puis mettre en œuvre des pratiques radicalement innovantes : celles du partage, de l'échange et de la réutilisation libre de leurs logiciels en marge des usages des fournisseurs de matériels et de logiciels officiels. C'est dans ce contexte qu'en 1985, Richard Stallman publie le *Manifeste GNU*, un texte fondateur décrivant les règles d'usage du logiciel libre et marquant l'avènement d'un projet technologique : le système d'exploitation GNU[2] (Stallman, Williams et Masutti, 2010).

Durant les trente années qui suivirent, cette production logicielle libre s'est consolidée, notamment grâce à la miniaturisation des machines, l'accessibilité grandissante de l'ordinateur personnel, le développement de logiciels socles fondamentaux[3] (GNU C Compiler, GNU Emacs, Bash, BSD Sockets, etc.), l'avènement des réseaux informatiques et plus tard, l'accès grand public à Internet (Weber, 2004).

Ainsi, nous sommes depuis une quinzaine d'années témoins de l'émergence d'un ensemble cohérent et robuste d'outils logiciels, de systèmes d'exploitation et d'applications diverses ; un écosystème viable qui devient — par l'adhésion toujours croissante de nouveaux utilisateurs — un des standards du marché. Ces outils sont utilisés par des millions d'utilisateurs quotidiennement, parmi les plus connus, le navigateur web Firefox, le système d'exploitation GNU/Linux, le langage de programmation PHP ou encore le logiciel de traitement de texte OpenOffice (Broca, 2013).

---

1 Le terme *hacker* dans ce contexte ne fait en aucun cas référence à la piraterie informatique.

2 Le projet GNU vise à créer un système d'exploitation libre et à code source « ouvert » complet.

3 Ces logiciels sont les premiers logiciels qui doivent être développés et qui servent d'outils et de base aux logiciels de plus hauts niveaux. C'est le cas par exemple du compilateur qui traduit le code source en langage machine.

1.2 De l'initiative individuelle à la communauté innovante

Les laboratoires de recherche en informatique, universitaires dans le cas du Massachusetts Institute of Technology (MIT AI Lab), de Berkeley (Berkeley Software Distribution) et industriels chez Bell et AT&T (Unix), ont joué un rôle important dans le développement de ce mouvement naissant. Ils ont notamment permis la rencontre de ces programmeurs pionniers autour de machines d'avant-garde, et vu naître les prémices du développement collaboratif, soutenant ainsi l'innovation en un lieu de pratique et de création des savoirs. D'autres, plus tard — nous pensons aux compagnies Netscape (1998) et Sun Microsystems (1999) — œuvreront également pour le mouvement du logiciel libre en libérant le code source de leurs logiciels dans un contexte concurrentiel, technologique et économique propice et concourant ainsi à la diffusion et à la promotion des valeurs du mouvement du « libre » (Stallman, Williams et Masutti, 2010 ; Levy, 2013).

Nous constatons que ni les États, ni les institutions de la recherche scientifique et technologique, ni les grandes entreprises commerciales, dominant alors le marché, n'ont été initiateurs, producteurs ou promoteurs de ces logiciels aujourd'hui en vogue et à la pointe de la technologie. En effet, ces millions de lignes de code informatique sont le fruit du travail bénévole de ces programmeurs créatifs (développeurs), des *hackers* — d'ailleurs souvent employés dans des entreprises de secteur de l'informatique — militants et regroupés en communautés (Raymond, 1999 ; Himanen, 2001 ; Lerner et Tirole, 2002). Cette vision du développement logiciel diffère de celle de l'industrie (Williams, 2002). Elle est innovante, car elle prend en compte le caractère immatériel et décentralisé de l'activité de production des codes sources, qui ne nécessite pas de lourds investissements financiers ou fonciers et qui repose principalement sur le travail cognitif de ses collaborateurs, dans une économie

du savoir — ou de la connaissance — dans laquelle le capital est immatériel (Foray, 2000).

Ce fourmillement d'idées et de travaux, multiples et dérivés, de la part d'inventeurs précurseurs, n'est pas un fait inédit dans l'histoire de la technologie ; l'invention de la bicyclette nous le rappelle (Pinch et Bijker, 1989). Cependant, les motivations de ce mouvement et les raisons sociales du progrès technologique qui en émanent sont, elles, bien plus novatrices (Lazaro, 2008 ; Oliveri, 2011). En effet, ces inventeurs ne sont pas motivés par la création d'entreprises privées, mais par la création d'un bien commun disponible à tous (Bergquist et Ljungberg, 2001 ; O'Mahony, 2003) dans une culture du don technologique (Coris, 2007). Ces communautés dont l'objectif est d'innover sont des organisations aux contours plus flous que ceux de l'entreprise ou de l'institution et dont les membres, entre amateur et expert, ne sont ni rémunérés, ni encadrés par le droit du travail, ni soumis à l'autorité d'une hiérarchie formelle, directe et contractualisée (Flichy, 2010). Ce paradoxe apparent a motivé et motive toujours les recherches dans les domaines des sciences de la gestion ou de la sociologie des techniques ou des organisations, afin de comprendre le monde du logiciel libre et son objet sociotechnique (Demazières, Horn et Zune, 2006 ; 2007a ; 2007b ; 2009 ; Meyer et Montagne, 2007 ; Muller, 2004).

1.3 De la Free Software Foundation à l'Open Source Initiative

Le logiciel en tant qu'artefact technologique est vecteur de valeurs, celles de ses auteurs, transférées à l'objet technique par leurs licences d'utilisation et leurs codes source. Les concepteurs de logiciels libres proposent de doter leurs utilisateurs de droits et de libertés (Williams, Stallman et Masutti, 2010), c'est la vision « idéologique », proposée par la Free Software Foundation (FSF), qui garantit les quatre libertés suivantes :

> (1) la liberté d'exécuter le programme, pour tous les usages ; (2) la liberté d'étudier le fonctionnement du programme, et de le modifier pour qu'il effectue vos tâches informatiques comme vous le souhaitez ; l'accès au code source est une condition nécessaire ; (3) la liberté de redistribuer des copies, donc d'aider votre voisin ; (4) la liberté de distribuer aux autres des copies de vos versions modifiées, 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. (Free Software Foundation, 2002)

Cependant, une partie des utilisateurs et des membres de la communauté, au sens large, ont souhaité s'émanciper de ces quatre libertés originelles dans une optique non plus politique, mais pragmatique. C'est l'origine du mouvement *open source*. Par *open source*, on entend un logiciel dont la licence d'utilisation respecte dix critères définis par l'Open Source Initiative [4] :

> (1) redistribution libre, (2) distribution du code source, (3) autorisation de produits dérivés, donc de modifications du code source, (4) respect de l'intégrité du code source de l'auteur, c'est-à-dire qu'il est possible pour l'utilisateur de modifier le code source, à la condition sine qua non que les changements puissent être et soient identifiés comme tel. (5) Absence de discrimination envers les individus ou les groupes, c'est-à-dire ne pas restreindre l'utilisation, la modification ou la distribution du logiciel au nom de principes discriminatoires. (6) Absence de discrimination envers les sujets d'utilisations, c'est-à-dire qu'un usage médical dans un domaine controversé, par exemple, ne saurait être interdit ; (7) Distribution de la licence. (8) La licence ne doit pas être spécifique à un produit, c'est-à-dire qu'un programme donné ne peut dépendre de sa distribution au sein d'un ensemble. (9) La licence ne doit pas restreindre d'autres logiciels en ce qui a trait à leur utilisation, modification ou distribution. (10) La licence doit être neutre en ce qui a trait à la technologie avec laquelle elle sera employée, c'est-à-dire qu'elle ne peut être prévue pour l'utilisation d'une technologie unique et encore moins en restreindre l'usage à cette technologie. (Open Source Initiative, 2014)

La définition de l'Open Source Initiative (OSI) initiée par Bruce Perens et Eric Raymond en 1998 (Von Hippel, 2001) semble évincer la dimension politique du logiciel libre ; pourtant, la dimension sociale et les valeurs du logiciel libre restent

---

4 Texte disponible en version originale à l'adresse suivante : http://opensource.org/osd.

entières. Le logiciel *open source* demeure librement distribuable, avec son code source, ce qui permet son étude et surtout ses modifications pour toutes utilisations sans discrimination (Crémer et Gaudeul, 2004).

1.4 Des communautés d'intérêts aux communautés de pratique

Le développement de logiciels libres, autrefois qualifié de hobby, est issu de pratiques techniques hautement codifiées. Ces logiciels sont d'abord le résultat de l'activité de programmation, c'est-à-dire l'écriture formelle et rigoureuse d'instructions, porteuse de sens, d'un langage informatique. En outre, ils s'inscrivent dans le droit par la licence d'utilisation qui leur est octroyée (Pellegrini et Cavenet, 2013). Enfin, ils sont le produit de pratiques de collaboration mises en œuvre pour maintenir une cohésion tant sociale que technique (la méritocratie et la revue par les pairs, notamment) au sein d'une équipe de développement (Broca, 2013).

Dans la désignation même du logiciel, l'adjectif « libre » (*free*[5]) est fort de sens, il assure un caractère distinctif face aux logiciels propriétaires — auxquels ne sont pas associés de libertés fondamentales, mais une licence d'utilisation généralement restrictive — alors qualifiés de logiciels privateurs. D'autre part, ils répondent aux critères d'une production collaborative nécessitant (1) coordination compte tenu de la complexité grandissante des projets (2) vérification et contrôle qualité ce qui inclut un ensemble de tests cohérents. Le système de production du logiciel libre est ainsi soutenu dans son fonctionnement par un ensemble de règles explicites et tacites qui forment le cadre normatif de sa production (Coris et Lung, 2005).

On notera que l'auteur d'un logiciel libre est clairement identifié, représenté et que son droit de propriété intellectuelle est protégé par un cadre légal (Pellegrini et

---

5 La phrase célèbre de Richard Stallman explicite ce terme : « *The term "free" is used in the sense of "free speech," not of "free beer."* »

Cavenet, 2013). C'est l'utilisation détournée du droit d'auteur — le *copyleft* — proposé par Richard Stallman et Elben Molgen qui permet à la fois de garantir la non-appropriation, la libre circulation des produits et les droits d'usage. Cette utilisation propose à ses auteurs et à ses utilisateurs des garanties que les logiciels dans le domaine public ou n'ayant pas de licence spécifique n'ont pas.

Ainsi, l'utilisation d'une licence libre et *open source* est d'abord issue d'une volonté de partage : c'est un choix éthique (Williams, Stallman et Masutti, 2010). Au mouvement du logiciel libre s'associe un discours idéologique, porté par la Free Software Foundation. Cette idéologie s'étend désormais à d'autres sphères, de la photographie (Flickr) à la musique (SoundCoud, Jamendo) en passant par l'édition ou l'art. On doit notamment cela aux licences Creative Commons, développées par Lawrence Lessig, et à l'apparition d'une « culture du libre » (Latrive, 2004).

En quoi ces communautés — issues de cette culture du libre, du développement logiciel et marquées par un engagement bénévole et ancrées dans une éthique de partage — innovent-elles en mettant en œuvre des méthodes de gestion, de communication, d'organisation et de régulation de leurs actions ?

Les communautés *open source* sont d'abord des regroupements d'individus, mais comment peut-on qualifier ces communautés ? Henri et Pudelko définissent la communauté d'intérêts comme « un regroupement de personnes qui se rassemblent autour d'un sujet d'intérêt commun. » (2006, p. 111). Cependant, dans le cas des communautés *open source*, cet intérêt n'est pas uniquement informatif ; il ne s'agit pas simplement de « mieux comprendre un sujet » ou « d'obtenir des réponses à des questions ou à des problèmes », car un objectif commun est à atteindre : réaliser un logiciel, un projet commun. Parallèlement, Etienne Wenger (1998) introduit la notion de communautés de pratique pour parler des groupes d'individus engagés dans la même pratique, et communiquant régulièrement entre eux à propos de leurs activités (Amin et Cohendet, 2004). Ce concept sera développé dans le chapitre suivant.

1.5 Des communautés étudiées par les sociologues

Dans leurs travaux, Crowston et Howison (2005) analysent les communications de nombreuses équipes du logiciel libre sur leurs plateformes de gestion de *bugs* et de rapports d'anomalies, utilisées lors de la phase de développement et de validation des logiciels, afin de caractériser les interactions entre les différents membres d'une même équipe. Dans la présentation de ces cas, ils observent que les échanges sont centralisés autour de quelques individus et que quatre niveaux de pratique apparaissent. Il y a les *core developers*, dont le rôle est l'écriture du code, en charge de l'architecture du projet ; les *co-developers* dont le rôle ponctuel est la résolution d'anomalies ; les *active users* qui ne participent pas à l'écriture du code, mais réalisent des tâches simples et enfin, les *passive users*, soit la majeure partie des utilisateurs périphériques au projet, qui ont un rôle d'observation. On remarque ainsi l'émergence de rôles différenciés et d'une hiérarchie au sein de l'équipe de développement de logiciel libre. Cette structure est modélisée de la manière suivante :

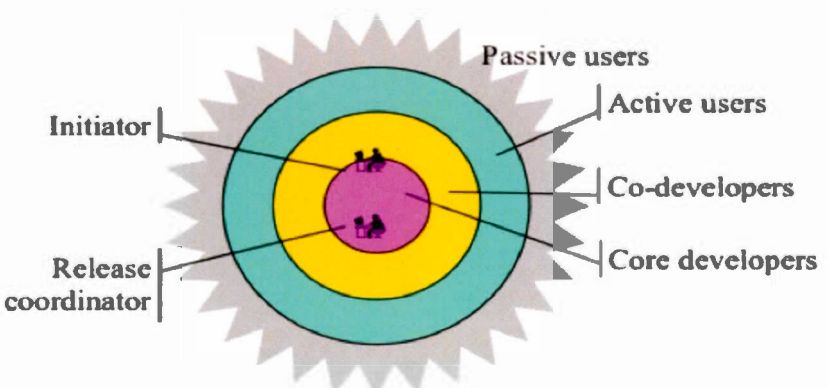

Figure 1.1 : Les membres du projet

Demazieres, Horn et Zune (2007a) exposent, par le recueil des témoignages des collaborateurs du projet Spip (un système de gestion de contenu sur le web), la

création de rôles spécifiques d'administrateurs-régulateurs lors de la complexification et de l'augmentation de la taille du projet. L'existence de ces agents — disposant d'un pouvoir de décision et de gestion des autorisations techniques des individus au sein de l'infrastructure du projet — démontre une certaine hiérarchie, plus fine, entre les membres d'une même équipe. De même, on assiste à une spécialisation des tâches, caractéristique d'une différenciation par la compétence. On peut ainsi assimiler ces aspects aux fonctions du *middle management*, c'est-à-dire des fonctions de coordination des agents développeurs ou traducteurs ; c'est un premier niveau d'organisation.

Meyer et Montagne (2007), pour leur part, attestent de l'existence de normes sociales au sein des communautés du logiciel libre. En effet, elles s'auto-régulent sous l'impulsion de leaders légitimes disposant d'un « capital social ». La réputation est alors le mécanisme identifié autorisant la prise de décision au sein de la communauté. Selon Coris et Lung (2005), ces leaders disposent d'un « droit de propriété morale » et peuvent nommer des individus disposant de responsabilité au sein de la communauté. D'ailleurs, si le collectif peut décider de la reconnaissance du leader — un chef dûment qualifié — il peut également le révoquer.

En ce qui concerne l'assignation des tâches dans le cadre du processus de résolution de *bug* et de production de correctifs, Crowston *et al* (2007) rendent compte de mécanismes d'auto-assignation existant au sein de la communauté des développeurs. De façon plus marginale, dans les cas où l'auto-assignation n'est pas effectuée, les développeurs assignent les tâches à un membre spécifique. La pratique la plus courante demeure toutefois dans ce cas d'assigner la tâche à une personne non spécifiée ou de ne simplement pas l'assigner, cette dernière restant ainsi disponible pour assignation.

Nicolas Auray (2007) détaille l'utilisation d'un outil de prise de décision : le vote. Au sein des communautés du libre, il permet de légitimer le choix collectif. Cela va à

l'encontre de la pensée selon laquelle une communauté serait dirigée par un leader unique, charismatique et omnipotent, celui que l'on nomme le « dictateur bienveillant », prenant toutes formes de décisions et ne laissant place à aucune forme de démocratie. La prise de décision semble repartie et distribuée sur un ensemble d'acteurs différenciés. D'ailleurs, « le rôle d'un leader dans la communauté reste ouvert à interprétation » (O'Mahony et Ferraro, 2007).

En étudiant la communauté Debian sur une période de treize ans, O'Mahony et Ferraro (2007) s'interrogent sur la notion de gouvernance appliquée aux communautés de développement *open source* et comment ces collectifs sociaux gouvernent, organisent et coordonnent leurs actions afin d'atteindre leurs buts collectifs. Ils montrent que même dans une communauté de développeurs où la contribution technique est très fortement valorisée, la conception de l'autorité par les membres évolue à travers le temps avec une augmentation de la valeur attribuée aux contributions organisationnelles (*organization-building contributions*). Les mécanismes démocratiques permettent au système de gouvernance de s'adapter, à mesure que les membres intègrent et interprètent le leadership et l'autorité dans la communauté (Muller, 2004). Ainsi, les communautés ont pour caractéristique intrinsèque d'être pilotées par un coordinateur, ou un groupe de coordinateurs, et témoignent ainsi d'un certain niveau d'organisation ; ces mécanismes de coordination « ne peuvent être envisagés qu'en étroite complémentarité avec les autres mécanismes classiques de coordination » (Cohendet et Diani, 2003). O'Mahony (2007) identifie lors de son étude sur les communautés Apache, Debian, GNOME et Linux Standards Base, cinq caractéristiques d'un modèle de gouvernance communautaire (*community management model*) : (1) l'indépendance de tout sponsor dans le processus de décision, le matériel supporté, et aucune relation d'autorité liée à l'emploi des membres de la communauté ; (2) la pluralité des points de vue de ses membres, de leurs usages, des applications et des environnements d'exécution des logiciels ; (3) la représentativité des membres de la communauté par un processus

démocratique ; (4) la prise de décision décentralisée, c'est-à-dire la distribution directe des pouvoirs de décision à certains de ses membres et ce dans un cadre de prise de décision public et transparent et (5) la participation autonome, c'est-à-dire la libre participation et la contribution selon les propres intérêts, motivations et capacités de chacun.

La littérature est ainsi relativement riche au sujet de la gestion des développeurs, soit la gestion de tâches récurrentes, le plus souvent techniques, dans leurs activités de développement. Cependant, les études s'intéressent peu au fonctionnement *interne* des communautés de grande envergure et aux processus de prise de décisions internes liées à l'activité de conception, et notamment en ce qui concerne la direction technologique. En outre, d'autres actions — parfois non techniques — comme la publicisation du projet, la formation, l'engagement dans la communauté (*advocacy*), la documentation ou l'organisation d'évènements sociaux ne sont que peu documentés (Guiri, Rullani et Torrisi, 2008).

Ainsi, certains éléments ne sont pas identifiés. Certes, les développeurs s'entraident et produisent du code, ils sont organisés et encadrés par des développeurs *leaders* (qui possèdent de l'expérience, de l'expertise et du capital social). Ce premier niveau de management est documenté (Ljungberg, 2000 ; Lakhani et Von Hippel, 2003) ; cependant notre hypothèse est qu'une communauté ne peut se développer, à un niveau international, dans des pratiques variées, dans sa taille et dans le nombre de projets qu'elle gère ; ni même proposer des conférences internationales, sans l'appui de mécanismes de gouvernance, d'outils d'évaluation et de prise de décision, ainsi que de dispositifs institutionnels. Il reste à déterminer la nature et les caractéristiques de ces dispositifs, de même que d'identifier les acteurs et les rôles qui leur sont associés.

1.6 Le cas de Python comme langage et communauté

Python est un langage de programmation relativement récent dans l'histoire de l'informatique et, déjà massivement utilisé à travers le monde. Le site officiel de Python[6] rapporte que le langage est employé dans des contextes aussi divers que ceux du développement logiciel, du stockage de fichiers en ligne (Dropbox), du web social (Facebook, Twitter), des infrastructures informatiques (Google, Red Hat), de la finance, de la recherche scientifique (SciPy), du cinéma (Walt Disney, Lucas Film) ou encore celui de l'éducation (Codecademy).

Créé au début des années 1990 par Guido van Rossum, un ingénieur logiciel hollandais, à l'Institut national de recherches mathématiques et informatiques des Pays-Bas (CWI pour *Centrum Wiskunde & Informatica*). Il a été conçu dans l'optique d'être le successeur du langage ABC. Démarrer comme un hobby, il se voit ensuite intégré dans le système distribué Amoeba du CWI. En 1995, Guido continua ses travaux sur Python dans l'organisme appelé CNRI (pour Corporation for National Research Initiative) aux États-Unis où il a développé plusieurs versions successives du logiciel, financé à la fois par des fonds de recherches et des fonds privés. En mai 2000, Guido et l'équipe de développement de Python, composée de trois personnes (*core team*), ont déménagé pour rejoindre la *startup* BeOpen.com et former l'équipe BeOpen PythonLabs. En octobre de la même année, l'équipe déménagea une nouvelle fois chez Digital Creations (maintenant Zope Corporation), société qui a énormément contribué au développement du langage. Enfin en 2001, la Python Software Foundation fût créée et annoncée à la neuvième conférence Python (PSF, Histoire de Python).

Le langage a la particularité d'être « multiparadigme », c'est-à-dire qu'il autorise plusieurs styles d'écriture du code et offre plusieurs manières de conceptualiser

---

6 http://www.python.org

l'information. Mais surtout, sa philosophie de conception force à la lisibilité du code et à une écriture fluide et facilitée, notamment par les mots clés en anglais, les signes qui structurent le langage et l'usage important des espaces pour clarifier et espacer les blocs de code. Python est un langage de haut niveau : il propose la programmation orientée objet, procédurale, impérative, ou encore fonctionnelle. C'est un langage dynamique qui permet l'écriture de programmes sous forme de scripts, d'applications autonomes ou encore d'applications web. Eric Raymond le recommande pour débuter en programmation : « Il est clairement conçu, bien documenté, et relativement facile pour les débutants. Bien qu'il s'agisse d'un bon langage de départ, ce n'est pas un jouet ; il est très puissant, adaptable à toute sorte de situations, et on peut l'utiliser pour de grands projets. » (Raymond, 2000)

L'implémentation standard[7] de Python, nommée CPython, est un logiciel sous la licence Python Software Foundation (PSF) compatible avec la licence GNU *General Public Licence* (GPL). Python est donc un langage de programmation libre et *open source* ayant un modèle de développement communautaire. Autour du langage, et de son fondateur Guido van Rossum, s'est développée une communauté de développeurs informatique tant utilisateurs que concepteurs du logiciel.

L'encadrement légal de Python est assuré par la fondation Python (Python Software Foundation ou PSF), une organisation à but non lucratif (enregistrée aux États-Unis sous la forme 503(c)(3) *non-profit corporation*) qui chapeaute le développement du langage de programmation et la communauté associée. La PSF gère la licence de Python, elle détient et protège toutes les marques commerciales associées à Python, notamment les logos.

Il est intéressant de constater que ce logiciel est développé par une communauté qui dispose d'une instance de représentation légale la Python Software Foundation, au

7 Python possède plusieurs implémentations, c'est-à-dire plusieurs compilateurs permettant de transformer le langage compréhensible par l'humain en langage machine.

même titre que les logiciels GNU sont sous l'égide de la Free Software Foundation ou les logiciels Apache sous l'Apache Software Foundation. Ceci constitue un schéma récurrent dans le modèle de développement communautaire de logiciels. La forme de la fondation devenant un organe institutionnel. La structuration en fondation reflète ainsi les défis de chaque logiciel dans son écosystème sur le marché (O'Mahony, 2005).

1.7 Questions de recherche

Question centrale de recherche :

En quoi la communauté Python constitue-t-elle une communauté innovante et quels mécanismes a-t-elle mis en place pour structurer l'action collective et soutenir le processus d'innovation collective ?

Questions spécifiques de recherche :

(a) Comment la communauté est-elle organisée et quels sont les différents rôles et statuts au sein de la communauté ?

(b) En quoi le mécanisme de proposition en ligne permet-il une expérience de développement collectif ?

(c) Comment le processus d'innovation est-il soutenu par les discussions en ligne ?

1.8 Pertinence de la recherche et objectifs

Cette recherche porte sur l'organisation d'une communauté produisant un logiciel libre et *open source*, le langage de programmation Python, et sur le caractère innovant de cette organisation.

L'intérêt de cette recherche est de documenter l'organisation du travail de production logicielle d'une communauté et ainsi comprendre comment, du point de vue de la gestion d'ensemble, cette grande communauté est pilotée à l'interne par les développeurs. Nous souhaitons également mieux appréhender le phénomène d'institutionnalisation de la communauté : comment des procédures explicites peuvent-elles émerger au sein de ce collectif, pour ainsi former un ensemble de règles souples, mais toutefois codifiées ?

Nous nous proposons de (1) dresser un portrait de la communauté en tant qu'organisation ; (2) décrire et étudier les mécanismes d'innovation, en particulier en ce qui concerne les mécanismes de proposition et d'évaluation des évolutions techniques au sein de la communauté ; (3) identifier les différents rôles qu'occupent les membres dans l'organisation et (4) mettre à jour et analyser un ensemble de discussions internes, au sein de la communauté, visant à faire évoluer le logiciel. D'une part, alors que la question du processus d'innovation reste en suspens dans ce type d'organisation à structure ouverte, sans brevets ni contrats et apparemment sans contraintes sur ses agents et que, d'autre part, l'entièreté des activités d'une communauté *open source* demeure peu documentée, cette recherche permettra de saisir au mieux les dimensions organisationnelles et innovantes de cette communauté. Nous nous proposons d'ouvrir la « boîte noire » de la technique et du travail des innovateurs et ainsi dévoiler des pratiques innovantes.

En ce qui concerne la pertinence sociale de ce projet, notons que ces développements et projets technologiques communautaires prennent de plus en plus d'importance dans nos sociétés et sont utilisés par un nombre d'utilisateurs en constante croissance. Ils deviennent ainsi des standards de fait et c'est pourquoi il importe de mieux les comprendre. Le cas de la communauté Python nous semble particulièrement intéressant parce que contrairement à d'autres, comme la communauté Apache, il n'a pas encore été beaucoup étudié (Squire, 2012). Ce projet contribuera à un ensemble

de travaux traversant plusieurs corps de littérature, de la sociologie des organisations au management. Il rejoint ainsi les problématiques centrales des *Science & Technologie Studies* (STS), en particulier le développement de l'innovation technologique et les rôles sociaux des collectifs de ce processus. Enfin, sur le plan méthodologique, ce projet contribue à une littérature naissante sur l'ethnographie virtuelle.

CHAPITRE II

CADRE THÉORIQUE

Dans ce second chapitre, nous présentons le cadre théorique qui est utilisé dans notre recherche. Mobilisant d'abord, la communauté de pratique afin de caractériser l'organisation sociale de la communauté Python ; nous engageons ensuite le concept de communauté épistémique pour comprendre la circulation et la création de nouvelles connaissances au sein du groupe. Enfin, nous présentons les travaux sur l'innovation par l'utilisateur pour comprendre le processus d'innovation collectif.

2.1 Théorie de la communauté de pratique

2.1.1 La communauté de pratique comme structure sociale

Le cadre conceptuel mobilisé dans ce projet pour analyser la communauté *open source* Python est celui de la *communauté de pratique*. Les communautés de pratique sont définies par Wenger, McDermott et Snyder (2002) comme :

> [D]es groupes de personnes qui partagent un intérêt, un ensemble de problèmes ou une passion à propos d'un sujet et qui vont accroître leur savoir et leur expertise dans ce domaine en interagissant sur une base continue. (p. 4)

Le lien entre pratique et communauté s'articule autour de deux points : (1) un modèle particulier de communauté : la communauté de pratique où (2) la notion de pratique s'interprète de manière souple : la pratique, c'est l'action de « faire », avec la capacité de structurer et de donner une signification aux actions. Les recherches sur les communautés au sein des organisations ont permis « de découvrir que c'est la construction collective d'une pratique interne qui permet de satisfaire les attentes institutionnelles ». Les communautés de pratique réalisent ensemble des tâches sur une base de connaissance commune et partagée. « Leur pratique sert d'appui à la mémoire collective, laquelle leur permet d'accomplir le travail sans pour autant avoir besoin de tout savoir » (Wenger, McDermott et Snyder, 2002).

Les communautés de pratique ne sont pas un fait nouveau ; elles sont notre première structure sociale fondée sur la connaissance. Les communautés de pratique sont partout. Nous appartenons tous à plusieurs d'entre elles : au travail, à l'école, à la maison et dans nos loisirs. Certaines ont des noms, d'autres pas ; certaines peuvent facilement être reconnues, d'autres restent invisibles. Nous sommes des membres actifs dans certaines et des participants occasionnels dans d'autres, mais, quelle que soit notre participation, la plupart d'entre nous sommes familiers avec l'expérience d'appartenir à une communauté de pratique.

Les membres d'une communauté de pratique ne travaillent pas nécessairement tous les jours ensemble, mais ils se rencontrent régulièrement, car ils tirent de la valeur de leurs interactions. Pendant les moments qu'ils partagent, ils échangent des informations, des idées ou des avis, ils s'entraident aussi à résoudre des problèmes.

L'activité principale d'une communauté de pratique est de favoriser l'apprentissage. Les membres de la communauté n'ont alors pas pour objectif premier d'y appartenir, mais plutôt d'apprendre continuellement : apprendre des autres, apprendre sur l'objet qui les rassemble, mais également apprendre dans l'accomplissement d'un projet commun. Les communautés de pratique prennent naturellement part à la vie de

l'organisation. Elles se développent d'elles-mêmes et sont plus ou moins reconnues dans l'organisation. Leur vigueur dépend principalement de l'engagement volontaire de ses membres et de l'émergence d'un leadership interne.

2.1.2 Participation et rôles dans une communauté de pratique

Les gens prennent part aux communautés de pratique pour différentes raisons : certains voient leur travail bonifié par une communauté qui lui donne de la valeur, d'autres peuvent y établir des contacts personnels et d'autres enfin saisissent l'opportunité d'améliorer leurs compétences. Au-delà de ces motivations variables, la participation de chacun des membres au sein de la communauté n'est pas nécessairement égale et peut évoluer dans le temps.

Les communautés de pratique peuvent être spontanées ou pilotées ; ces dernières étant développées par la direction de l'organisation. Qu'elles soient spontanées ou pilotées, une personne en charge de la coordination organise les évènements et présente ou connecte les membres entre eux. Ce ne sont pas les seuls à avoir des rôles de leadership, d'autres individus au sein de la communauté peuvent aussi développer ce rôle. Les communautés de pratique sont ainsi auto-organisées et créent leurs propres règles. Enfin, on distingue généralement trois principaux niveaux de participation dans la communauté : le niveau des membres appartenant au noyau dur (*core group)*, celui des membres actifs et celui des membres périphériques (Wenger, McDermott, Snyder, 2002).

2.1.3 Les dimensions de la pratique dans une communauté de pratique

Les communautés de pratique se caractérisent par trois propriétés ou facteurs de cohérence : un engagement mutuel, une entreprise commune et un répertoire partagé (Wenger, 1998).

L'engagement mutuel constitue le « faire ensemble », c'est-à-dire l'action de maintenir un sens commun dans les interactions, entre individus, auxquelles chacun est partie prenante. La notion de « compétences complémentaires » émerge alors dans le groupe, puisque chacun, ne pouvant tout savoir, doit s'appuyer sur les contributions des autres.

L'entreprise commune est le résultat du « processus collectif de négociation » du groupe et représente sa cohésion. Elle a pour objectif de créer chez le membre une relation de responsabilité par rapport au groupe et devient une partie de la pratique même. L'entreprise commune est négociée, en ce sens qu'elle n'est pas « déterminée par un contrôle, une décision extérieure ou un seul participant », et ce, même lorsque la communauté de pratique nait sous un mandat extérieur.

Le répertoire partagé est un ensemble de ressources à la disposition de la communauté de pratique et constitué par ses membres. Cela comprend des routines organisationnelles, un vocabulaire, des outils, des procédures, des concepts et des histoires partagées qui deviennent ainsi partie intégrante de la pratique commune.

Ces trois dimensions sont basées sur la négociation de sens au sein de la communauté, c'est-à-dire le processus par lequel l'ensemble des tâches, des activités (actions, langages, pensées) — ou encore des relations sociales — acquièrent une signification construite dans un contexte donné. C'est (1) « une démarche active de production de sens qui est à la fois dynamique et historique » ; (2) « une perspective à la fois souple et flexible » ; (3) « la capacité mutuelle d'influencer et d'être influencé » ; (4) « l'inclusion de plusieurs points de vue et facteurs » ; (5) « la

conception d'une nouvelle résolution alignée sur la convergence de ces facteurs et points de vue » ; (6) « le caractère incomplet de cette résolution, qui peut être partielle, éphémère et propre à une situation ».

En outre, les communautés de pratique ont trois dimensions structurantes selon Snyder et Wenger (2010) : le domaine de la connaissance, qui identifie un ensemble de problèmes ; la communauté des personnes qui s'intéressent à ce domaine, et la pratique commune qu'elles développent pour être efficientes dans leur domaine.

Le domaine

Une communauté de pratique s'intéresse à un domaine spécifique, ce qui définit son identité et ce à quoi elle accorde de l'importance. Quel que soit le domaine, la passion est cruciale. La passion des membres de la communauté de pratique pour le domaine n'est pas une chose abstraite ni désintéressée. Cela représente souvent une part profonde de l'identité personnelle de chacun des membres et une manière d'exprimer une partie du travail d'une vie. Le domaine crée un terrain et un sens commun auxquels les membres de la communauté de pratique peuvent s'identifier. Un domaine défini donne de la légitimité à la communauté en affirmant son objet et en accordant de la valeur aux membres. Le domaine inspire les membres, les encourage à participer et à contribuer, en donnant un sens à leurs actions, de même qu'il guide leur apprentissage. Connaître les frontières du domaine permet aux membres de décider ce qui vaut la peine d'être partagé, de déterminer comment présenter leurs idées et quelles activités poursuivre. Cela permet également aux membres de reconnaître le potentiel de tentatives d'idées ou d'idées semi-développées.

La communauté

Le second élément est la communauté elle-même et la qualité des relations qui unit ses membres. De manière optimale, la communauté des membres reflète la diversité des perspectives et des approches nécessaires pour mener à bien l'effort d'innovation dans le domaine. La communauté crée la structure sociale d'apprentissage. Une

communauté forte est basée sur des relations interpersonnelles faites de respect mutuel et de confiance. Cela encourage la volonté de partager des idées, d'exposer parfois son ignorance, de poser les questions difficiles et écouter attentivement. La communauté est un élément important, car l'apprentissage est à la fois un processus intellectuel, mais également une volonté d'engagement.

La pratique

Chaque communauté développe sa pratique en partageant et en développant la connaissance de chaque participant dans son domaine. Les éléments de la pratique incluent un répertoire d'outils, un cadre de travail, des idées, des documents, des méthodes et des histoires — tandis que les activités de la communauté sont l'apprentissage et l'innovation. La communauté partage à la fois la pratique et un ensemble d'artefacts. La pratique est cette compétence, ce savoir spécifique que la communauté développe, partage et maintient. Le corpus de savoir partagé et des ressources permet ainsi à la communauté de se confronter efficacement à son domaine.

2.1.4 Une communauté de pratique virtuelle

Une communauté est dite virtuelle ou en ligne lorsque les échanges ne se font pas en présence ou en face-à-face. Cela signifie souvent que les membres de la communauté sont géographiquement distribués. Les moyens de communication sont alors ceux des technologies de l'information et de la communication (TIC) ; les échanges se font via un ordinateur et sur le réseau Internet. La communication de groupe est ainsi « médiatisée par l'informatique ». La temporalité de la communication est modifiée : les échanges peuvent se faire de manière asynchrone, c'est-à-dire différés dans le temps (par exemple par l'échange de courriels), ou de manière synchrone, c'est-à-dire instantanée (lors de vidéoconférences ou de sessions de *chat*) (Latzko-Toth, 2010). Ainsi, certains aspects de la communauté de pratique se transfèrent aisément d'une

localisation physique à une localisation virtuelle. Si les membres de la communauté exercent la même activité, alors ils partagent en ligne un même intérêt, un même langage, un même but commun et le même savoir (Hildreth, Kimble et Wright, 2000). La participation aux échanges en ligne devient alors centrale dans l'évolution de la communauté et dans la création des relations interpersonnelles ; ces dernières vont aider au développement d'un sentiment de confiance et d'identité, ce qui définit la communauté. De plus Proulx (2006), indique que « la communication de groupe médiatisée par l'informatique constitue un environnement social et symbolique dans lequel les participants peuvent développer un sentiment d'appartenance au groupe et s'y construire une identité collective ; qu'elle soit communautaire ou sociale ».

2.1.5 Le concept de communauté épistémique

Peter Haas (1992) introduit le concept de communauté épistémique, dans le domaine des relations internationales, comme un réseau de professionnels issus d'une variété de disciplines et d'expériences, dotés d'une expertise reconnue et compétents dans un domaine particulier. Ils ont (1) un ensemble partagé de principes normatifs ; (2) des croyances communes issues de leurs pratiques dans leurs domaines et l'ensemble de problèmes qu'ils traitent ; (3) des notions partagées de validité avec des critères internes définis et intersubjectifs permettant de garantir la validité de la connaissance dans leur domaine d'expertise ; (4) une entreprise politique commune, c'est-à-dire un ensemble de pratiques communes associées à un ensemble de problèmes communs vers lesquels sont dirigées leurs compétences professionnelles et leur pratique de résolution de problèmes. En outre, leur compréhension et leurs méthodes d'apprentissage (*shared way of knowing*) sont partagées; comme leurs schémas de raisonnements, leurs valeurs, leur implication dans la production de connaissance.

Cohendet *et al* (2003) définissent les communautés épistémiques comme « un groupe de représentants partageant un objectif cognitif commun de création de connaissances

et une structure commune permettant une compréhension partagée. » (p. 105). La communauté épistémique est une communauté « intensive en connaissance » c'est-à-dire « dans laquelle une part non négligeable de ses membres produit et reproduit la connaissance, dont les contours délimitent un espace public (ou semi-public) de circulation des savoirs et où l'usage des nouvelles technologies d'information et de communication a radicalement réduit les coûts de codification et de distribution de la connaissance » (David et Foray, 2002, p. 19). Les communautés épistémiques présentent certaines vertus : (1) la progression de la connaissance est renforcée ; (2) une large part de la connaissance est codifiée ; (3) le contrôle de la qualité de la connaissance est assuré par chacun ; (4) l'efficacité y est renforcée. On ne reproduit pas deux fois la même connaissance et les nouvelles connaissances bénéficient d'un effort collectif d'expérimentation et d'amélioration ; (5) la productivité de l'apprentissage s'accroît, on apprend ainsi à apprendre (David et Foray, 2002). Les notions d'identité et d'autonomie y sont plus faibles que dans une communauté de pratique (Karoui Zouaoui et Hchich Hedhli, 2014). L'autorité procédurale interne peut y être explicite (exercée par un responsable) ou implicite (issue d'un « code de conduite ») et permet de définir les objectifs de la communauté. Dans les communautés épistémiques, l'objectif commun et primaire est la création de connaissances ; dans les communautés de pratique, la création de connaissances y est un effet secondaire, mais néanmoins existant. Ainsi, la qualité des connaissances produites dépend de l'hétérogénéité des expériences des membres de la communauté. Les communautés épistémiques produisent des connaissances, les communiquent et les distribuent ; « elles créent des formes multiples de répertoires pour accumuler et stocker ces connaissances ». Aussi, elles produisent des producteurs de connaissance, c'est-à-dire que ses membres apprennent à produire de la connaissance, et ce, en même temps que la communauté se réalise par « des dispositifs ou des évènements qui visent à fabriquer de la communauté » (Meyer et Molineux, 2011).

La communauté Python serait à la fois une communauté de pratique — par son usage et sa pratique du langage de programmation Python — et une communauté épistémique — par ses regroupements d'utilisateurs permettant la conception du langage et la construction de nouvelles connaissances sur ce dernier. Ces utilisateurs particuliers forment ainsi un ensemble d'utilisateurs-innovateurs que nous nous apprêtons à caractériser.

2.2 Théorie de l'innovation par l'utilisateur

L'innovation par l'utilisateur, un concept développé par Eric Von Hippel en 1988 (Von Hippel, 1988), fait référence aux innovations développées par un utilisateur final plutôt que par une organisation. L'innovation devient alors un processus du bas vers le haut (*bottom-up*) plutôt que du haut vers le bas (*top-down*) au sein de l'organisation élargie incluant ses usagers : on parle ainsi d'innovation ascendante.

Dans son étude sur les instruments scientifiques, Von Hippel (1988 p. 19) identifie les utilisateurs innovants (*innovative users*), des utilisateurs qui développent de nouveaux instruments en amont du constructeur. Ainsi l'utilisateur innovant : (1) perçoit en avance qu'un nouvel instrument est requis ; (2) invente l'instrument ; (3) construit un prototype ; (4) prouve la valeur du prototype en l'expérimentant ; (5) diffuse des informations détaillées à la fois sur la valeur de l'invention et sur la manière de procéder pour répliquer le prototype. Le rôle du constructeur est alors (1) d'améliorer le prototype par son travail d'ingénierie tant dans sa fiabilité que dans son utilité ; (2) de produire, de mettre sur le marché et de vendre le produit innovant.

Dans le processus d'innovation, l'utilisateur innovant assume donc les rôles d'identification des besoins, de résolution des problèmes par l'invention, de construction d'un prototype et de démonstration de sa valeur par l'utilisation.

2.2.1 L'innovateur individuel : le *lead user*

L'usager expert, ou *lead user* (Von Hippel, 1986), clé dans ce processus d'innovation, est un utilisateur qui dispose d'une bonne expertise dans son domaine et qui de ce fait a des besoins spécifiques. Ces besoins se généraliseront sur le marché et deviendront dominants.

Le *lead user* est externe à l'organisation et il peut fournir de nouveaux concepts de produits, de nouvelles données et de nouveaux designs liés à ses besoins spécifiques. Von Hippel propose ainsi quatre étapes pour intégrer les besoins des *lead users* dans le processus d'innovation : (1) identifier une tendance importante (notamment technique) du marché ; (2) identifier les utilisateurs qui mènent cette tendance en termes d'expérience et d'intensité des besoins ; (3) analyser les données générées par les *lead users* : besoins identifiés explicitement ou tacitement, expériences dans des conditions réelles d'utilisation du produit ou par l'usage de produits connexes et combinés ; (4) diffuser les données des *lead users* dans le marché global.

Pour résoudre le problème de la spécialité dans le design et la conception d'un produit spécifique, Von Hippel (2001) propose que les constructeurs fournissent à leurs utilisateurs des boîtes à outils (*toolkits*) leur permettant ainsi de *participer* au design et à la conception des produits.

Ces boîtes à outils se doivent d'être propices à l'utilisation par un non-expert (*user-friendly*). Les boîtes à outils permettent ainsi de créer et de tester des concepts de produit dérivés personnalisés (*customized*). Ces boîtes à outils vont réaliser cinq objectifs : (1) permettre à l'utilisateur de prendre en charge les cycles d'apprentissage par essai-erreur ; (2) proposer à l'utilisateur un « espace de solution » (*solution space*) prédéfini qui permette ses designs et concepts utilisateur ; (3) permettre aux utilisateurs de s'exprimer dans leur « langage » et leur compétence propre sans apprentissage supplémentaire ; (4) favoriser la modularité et ainsi l'expression

spécifique d'un élément unique (*module*) proposé ; (5) s'assurer que les produits dérivés par les utilisateurs soient dotés des caractéristiques de production et qui ne doivent pas faire l'objet de modifications (*revisions*) par les ingénieurs de la firme.

L'usage des boîtes à outils et leur développement pour les utilisateurs permet aux firmes de s'assurer d'un transfert effectif, rapide et sûr des besoins des utilisateurs en leur proposant de prendre en charge les étapes (*stages*) de conception, ainsi que de recherche et développement.

Franke et Von Hippel (2003) montrent comment le logiciel Apache offre les caractéristiques d'une boîte à outils logicielle proposée aux utilisateurs-innovateurs et qui leur permet de modifier le logiciel original pour répondre à leurs besoins spécifiques en matière de sécurité. D'une part, le logiciel est *open source* et permet donc explicitement la modification du code source, son étude, son acquisition et son utilisation libre et gratuite. D'autre part, les outils pour la modification du dit logiciel, comme le compilateur du langage C et sa suite logicielle de test, sont également disponibles en *open source*. Une fois réunis, ils permettent aux utilisateurs de tester, de mettre en œuvre et de proposer leur propre version logicielle contenant l'innovation utilisateur. En pratique, « les boîtes à outils utilisées dans les projets *open source* ont été améliorées à travers la *pratique* et sont constamment améliorées par les utilisateurs-innovateurs » (Von Hippel, 2005, p. 101)

La conception participative permet d'inclure les besoins utilisateurs dans le processus de recherche et développement et de placer ces utilisateurs sur le même plan que les ingénieurs de la firme. Les frontières de différenciations entre expert et non-expert, entre amateurs et spécialistes, deviennent plus floues. Les *lead users*, parfois nombreux, partagent l'ensemble de leurs « trouvailles » au sein de communautés où ces idées et solutions techniques sont partagées à tous et par tous (Hillairet, 2012).
Ces informations sont dites « adhérentes » (*sticky*) : elles sont coûteuses à acquérir par l'entreprise, de même que difficiles à transférer et à utiliser par d'autres. Les *lead*

users innovent alors en dehors du cadre classique des organisations, mais à l'intérieur de communautés d'utilisateurs innovants.

2.2.2 Les communautés d'innovation

Les utilisateurs-innovateurs se regroupent en communautés dans lesquelles ils partagent librement leurs connaissances et le résultat de leurs innovations (Von Hippel, 2005). C'est une importante fonction des communautés d'innovations que d'induire une pratique commune de mise en commun des innovations. Pourtant, ces communautés sont hétérogènes : *lead users* et simples demandeurs d'information y sont tous deux présents — le partage de l'information constituant la motivation intrinsèque des membres de la communauté. Dans les communautés d'innovation, les individus ou les organisations sont groupés en nœuds, interconnectés par des liens de transfert d'informations forts, et qui peuvent se faire en face-à-face ou de manière électronique. Ainsi, la communauté d'innovation peut exister en dehors des limites et au-delà des frontières de l'appartenance à un groupe donné. La communauté fournit alors aux participants les aspects suivants : (1) sociabilité ; (2) soutien ; (3) information ; (4) notion d'appartenance et (5) identité sociale (Von Hippel, 2005).

Les communautés d'innovation sont souvent spécialisées et servent de répertoires d'informations liées à l'innovation, et ce, de manière physique ou virtuelle. Par exemple, userinnovation.mit.edu est un site web spécialisé grâce auquel les chercheurs peuvent déposer les articles issus de leurs recherches sur l'innovation par les utilisateurs. Les contributeurs et les non-contributeurs peuvent accéder librement au site et le parcourir pour y trouver de l'information. Les communautés d'innovations peuvent également fournir d'autres ressources aux participants. Les systèmes de discussion instantanée (*chat rooms*) ou les listes de diffusions permettent aux contributeurs de publier, d'échanger des idées et d'offrir de l'assistance mutuelle.

Les communautés d'innovation fournissent du support aux innovateurs individuels également sous la forme d'outils. Ces outils, souvent développés par les utilisateurs de la communauté eux-mêmes, permettent d'aider, de développer, d'évaluer et d'intégrer le travail des membres de la communauté. Ainsi, les utilisateurs ne se limitent pas à distribuer et à évaluer des innovations terminées ; ils se portent également volontaires pour assister les autres dans le développement et la mise en application d'innovations (Von Hippel, 2005).

Selon Von Hippel (2005), ces fonctionnalités sont tout particulièrement visibles dans les communautés développant les logiciels libres et *open source*.

2.2.3 Les réseaux horizontaux d'innovation

Von Hippel (2007) utilise une nouvelle fois l'exemple de la communauté développant le serveur web Apache, logiciel libre et *open source*, pour présenter une extension de son concept de communauté d'innovation. Le *réseau horizontal* d'innovation (*horizontal innovation network*) est constitué uniquement d'utilisateurs *innovants* ou plus précisément « auto-producteur » (*user/self-manufacturer*). Les utilisateurs participants au réseau conçoivent et construisent des produits innovants pour leurs propres besoins et ils mettent les informations de conception à disposition des autres membres du réseau. Ces derniers peuvent alors (1) répliquer et améliorer l'innovation dévoilée ; (2) révéler gratuitement leurs propres améliorations à leur tour ; ou (3) adopter simplement le concept produit pour leur propre besoin. Le concept de communauté d'innovation se prolonge dans celui des réseaux horizontaux d'innovation : (1) l'utilisateur assume seul les fonctions de recherche et développement, production et distribution ; (2) cette auto-production peut faire concurrence à la production et à la distribution commerciale d'un autre produit. Ceci peut être fait, particulièrement pour les biens informationnels, grâce à de faibles coûts

de reproduction et de diffusion (par Internet par exemple). L'utilisateur peut ainsi se passer d'un producteur. Dans ce cadre de fonctionnement, tous les investissements spécifiques à l'innovation et activités liées sont ainsi portées par l'utilisateur-innovateur.

2.3 Synthèse

Nous mobiliserons la notion de communauté de pratique pour comprendre la communauté Python en tant que structure sociale organisée autour d'une pratique, soit la conception du langage de programmation Python, et de valeurs comme la méritocratie ou l'évaluation par les pairs. La question des connaissances étant centrale dans cette organisation, la notion de communauté épistémique nous permettra de comprendre comment se font la production et la circulation de nouvelles connaissances au sein de la communauté. Enfin, dans la mesure où nous nous intéressons à la dimension innovante de cette communauté et à ses acteurs individuels, la perspective de Von Hippel sur l'utilisateur-innovateur est des plus éclairante. Elle va nous permettre de mettre au jour les acteurs innovants et leurs activités au sein de la communauté. De plus, les notions étendues de communauté d'innovation et de réseaux horizontaux d'innovation nous permettront, d'une part, de décrire l'ensemble du processus d'innovation et, d'autre part, d'analyser ces démarches créatives.

CHAPITRE III

MÉTHODOLOGIE

Dans ce troisième chapitre, nous présentons la méthodologie de recherche choisie afin de réaliser cette étude. Nous décrivons d'abord notre démarche de nature inductive et qualitative basée sur l'ethnographie virtuelle. Puis nous abordons les techniques utilisées pour mener à bien cette recherche : l'observation exploratoire, l'analyse d'un corpus documentaire, l'analyse de fils de discussion de la communauté Python. Enfin, nous faisons état des problématiques éthiques liées à cette recherche.

3.1 Une ethnographie virtuelle

Dans le cadre de cette recherche, nous adoptons une démarche de recherche qualitative basée sur l'ethnographie virtuelle, telle que définie par Hine (2000). Cette méthodologie nécessite d'abord la présence soutenue de l'ethnographe sur le terrain étudié afin de produire un savoir ethnographique — les médias numériques fournissant de nouvelles opportunités pour l'ethnographe en amenant la notion de site d'interaction.

Le cyberespace ne doit pas être envisagé comme un espace détaché de toutes connexions à la « vie réelle » et aux interactions en face-à-face. Dans cet ordre d'idées, l'objet de la recherche ethnographique repose sur le flux d'informations et la

connectivité, plutôt que sur les localisations et les frontières comme principes organisant. Autrement dit, il s'agira d'étudier une activité à travers les échanges qu'elle suscite, même si ceux-ci dépassent le cadre du dispositif à l'étude (un forum de discussion par exemple) pour s'intéresser aussi à ses manifestations dans d'autres contextes (dans un autre dispositif ou hors-ligne). Ainsi l'ethnographie virtuelle peut et doit s'appuyer sur des interactions médiatisées mobiles plutôt que multilocalisées. Les limites de ses interactions ne sont pas définies à priori, mais plutôt explorées au cours de l'étude. Concrètement, la décision de cesser l'étude ethnographique devient une décision pragmatique. Par ailleurs, toutes les formes d'interactions sont valides d'un point de vue ethnographique, et non pas uniquement celles qui ont lieu en situation de face-à-face. Dans le cadre de notre recherche, nous nous intéresserons donc aux interactions médiatisées, en l'occurrence celles qui se déroulent en ligne dans le cadre d'échanges par courrier électronique au sein de listes de discussion.

3.2 L'observation exploratoire

L'observation exploratoire vise à sélectionner les sites les plus pertinents pour l'analyse. Dans le cas de notre recherche, il s'agit d'explorer l'ensemble des sites en ligne, développés ou utilisés par la communauté Python pour diffuser de l'information ou communiquer. Il s'agit également d'identifier les outils techniques, dont se dote la communauté en ligne pour fonctionner, de façon à pouvoir les circonscrire et les explorer plus en détail. Pour Guimarães, l'objectif de ce travail exploratoire préliminaire est de délimiter le terrain en procédant à une première délimitation du groupe social sur lequel portera l'étude, par exemple à travers les interactions sociales de ses participants, ou encore de délimiter les groupes sociaux fonctionnant à travers un même support technique (Guimarães, 2005).

Dans le cas de la communauté Python, les sites et outils techniques sont variés : ils vont du site web à vocation de communication institutionnelle, au site de gestion de contenu très technique, qui inclut par exemple une plateforme de gestion de *bugs*, un outil de gestion des codes sources et des *wikis* (dispositif de contribution éditoriale en ligne). Il existe également des documents de travail, de type bureautique, partagés en ligne comme des tableaux ou des comptes rendus, et enfin et surtout, des échanges en ligne sur des listes de discussion.

L'exploration de ces masses d'informations permettra de saisir la dynamique de la communauté à travers ses activités de communication, de diffusion, de publicisation des projets et surtout d'échanges sur les listes de discussion. Certains intervenants au sein de la communauté pourront ainsi faire l'objet d'un suivi de leurs interactions sur une période donnée. Cette première analyse exploratoire permettra la création d'un corpus de documents et de traces d'interaction à analyser.

3.3 L'analyse d'un corpus documentaire

La communauté Python s'articule et s'organise principalement sur Internet. Pour ce faire, elle publie de nombreuses informations sur différents sites, par l'échange de courriels ou des fichiers de travail partagés.

Le site de la communauté Python (https://www.python.org) qui présente le langage et qui fait office de portail sur le web pour la communauté constitue la première source d'information. On y trouve en effet une section consacrée à la Fondation Python (https://www.python.org/psf) qui, combinée au blog *Python History* de Guido van Rossum (http://python-history.blogspot.com), constitue un vaste ensemble de documents pertinents sur la communauté, son histoire et son fonctionnement. En outre, le *wiki* de Python (https://wiki.python.org) et le site web de la conférence

annuelle PyCon (https://us.pycon.org) représentent de précieuses sources d'information.

On y trouve un nombre important de documents de travail provenant autant de la Fondation que de la communauté Python « en action ». À titre d'exemple, le portail de Python présente le *Guide du Développeur de Python* avec la liste des tâches qui incombe au *core developers*. On trouve y également les statuts officiels publiés par la Fondation, la description des groupes d'intérêts, les propositions d'améliorations du langage (les *Python Enhancement Proposals,* ou PEP, des documents calqués sur les *Request For Comments* — RFC — de l'IETF), de même que les procès-verbaux de réunions. Ces documents seront triés et sélectionnés pour constituer un corpus documentaire à analyser.

3.4 L'analyse de fils de discussion

Au-delà de cette présence sous forme de textes, la communauté Python existe d'abord et avant tout par une intense activité en ligne, qui se déroule au sein de plusieurs listes de discussion. Nous procéderons à « l'exploitation des traces numériques des activités du groupe » (Demazières, Horn et Zune, 2011, p. 167). Nous nous intéressons donc à deux listes qui jouent un rôle déterminant au sein de la communauté Python et dans l'élaboration du langage : la liste *python-dev*, une liste dédiée à la communication publique des *core developers* et la liste *python-ideas*, une liste dédiée à la communication et la promotion de nouvelles idées sur le langage au sein de la communauté.

Nous procéderons alors à la sélection des fils de discussion les plus pertinents, sur les listes de diffusion *python-dev* et *python-ideas*, pour notre étude de l'innovation par la communauté.

Cette sélection suivra une démarche inductive, elle s'apparentera à une démarche « conduite par les données, [où] le choix des situations dépend d'hypothèses préalables générales sur ce qu'on cherche et sur les situations susceptibles de le procurer » (Traverso, 1999, p. 22).

Nous souhaitons ainsi mettre au jour l'organisation qui se construit en ligne à travers le processus de proposition d'amélioration du langage Python (PEP). Pour ce faire, nous décrirons et analyserons les *mécanismes* de proposition et d'évaluation des évolutions techniques qui sont discutées au sein de différents fils de discussion. Nous sélectionnerons des fils de discussion relatifs à des évolutions ciblées du langage (PEP).

Ces fils de discussion formeront un corpus de conversations en ligne sur lequel nous conduirons une analyse de contenu : « l'analyse de contenu a pour but de connaître la vie sociale à partir de [la] dimension symbolique des comportements humains » (Sabourin, 2003, p. 358). Faire une analyse de contenu implique de produire du langage — le discours de l'analyse des documents — à partir du langage : les documents à analyser. L'analyse de contenu constitue « un ensemble de démarches méthodologiques recourant à des méthodes et des techniques » ayant pour fonction l'interprétation de documents dans le but de connaître la vie sociale. Cette vie sociale peut être appréhendée à l'échelle individuelle ou collective.

L'identification des situations de langage et des contextes sera fondée sur des *catégories*, construites à partir des données, et ayant pour but de saisir le sens manifeste du texte. Cette démarche implique un va-et-vient nécessaire entre observations des données et hypothèses élaborées en cours de route (Traverso, 1999). « Le chercheur élabore par induction les catégories et procède au codage » c'est-à-dire « au découpage des informations contenues dans les documents en fonction des unités d'analyse préalablement définies. » (Bonneville, Grosjean et Lagacé, 2007, p. 195). Nous étudierons ainsi les processus de construction de l'innovation par des

membres de la communauté organisés autour des propositions d'amélioration du langage Python.

3.5 Dimensions éthiques

Les données[8] utilisées pour cette recherche étant publiques et librement accessibles, il n'est à priori pas nécessaire de procéder à une demande d'approbation éthique pour réaliser notre recherche. L'énoncé des politiques des trois Conseils (2010, p. 18) sur l'éthique de la recherche portant sur les êtres humains stipule que « l'exemption de l'évaluation par un CÉR repose sur le fait que l'information se trouve dans le domaine public et qu'on peut y accéder, et que les personnes visées par l'information n'ont pas d'attente raisonnable quant à la protection de leur vie privée ». Toutefois, bien que ces discussions aient lieu dans un espace public, les participants pourraient avoir l'impression d'échanger dans un espace privé. À ce titre, nous veillerons à respecter l'intégrité et à protéger la vie privée des participants en anonymisant les échanges, c'est-à-dire en remplaçant les émetteurs et les destinataires des courriels que nous allons étudier. Dans le cas du fondateur de Python, Guido van Rossum, nous avons choisi de conserver son nom et prénom puisque c'est une personnalité de notoriété publique.

8 Les données utilisées, y compris les messages courriels des listes de discussion archivés sur le web.

CHAPITRE IV

PORTRAIT DE LA COMMUNAUTÉ PYTHON

Dans ce quatrième chapitre, nous dressons le portrait de la communauté Python à la lumière du concept de communauté de pratique et passons en revue le domaine, la participation, les rôles et les différentes ressources construites par la communauté. En mettant l'accent sur l'action de « faire », nous montrons la capacité de la communauté de se structurer grâce à sa pratique et donnons du sens à ses actions.

4.1 La communauté Python comme communauté de pratique

4.1.1 Un domaine

La communauté Python se forme autour d'un domaine spécifique, la programmation informatique ; la totalité de ses membres sont des développeurs employant le langage de programmation Python dans le cadre de leurs activités. Ces programmeurs proviennent de divers horizons, de l'employé salarié par une firme privée qui utilise Python dans le cadre de son travail au féru d'informatique qui développe une pratique durant son temps libre. Ce sont tous des individus passionnés par Python et chez qui ses caractéristiques intrinsèques trouvent résonance : les valeurs du libre ont ceci de particulier qu'elles génèrent une adhésion forte au domaine (Meissonier *et al*, 2010). Ce dernier se construit comme espace d'utilisation et de promotion de ces valeurs et

avec des frontières clairement délimitées. En d'autres termes, sur le plan technique, c'est uniquement par leurs compétences dans ce langage que les membres de la communauté se voient attribuer une certaine valeur ; il s'agit de l'unique domaine d'expertise des membres par lequel ils trouveront reconnaissance. La communauté Python circonscrit sa pratique au développement du langage et de ses outils dérivés. L'entreprise commune repose sur la création d'un langage de programmation suivant des caractéristiques propres à la vision de ce que doit être un langage informatique : celles développées dans le langage Python en tant qu'idéal. Le sens commun ainsi créé comporte une forte part normative, formalisée dans des ressources créées par la communauté, que l'on retrouve dans le *Zen de Python* et dans le *Guide du développeur de Python*.

4.1.2 Le répertoire : des ressources partagées

Dans une communauté de pratique, le répertoire partagé est l'ensemble des ressources créées par les membres de la communauté : des outils, des procédures, des routines organisationnelles, des récits. Au sein de la communauté Python, ces ressources sont le plus souvent dématérialisées, puisque ses membres utilisent intensivement les technologies de l'information et de la communication. Les échanges se font presque exclusivement en ligne et de manière asynchrone, les membres de la communauté étant dispersés géographiquement. La communication est ainsi médiatisée par les outils informatiques que sont la messagerie électronique, les listes de discussions, ou les plateformes de collaboration mises en œuvre par la communauté. Ces outils laissent des traces sur Internet et ce sont ces traces qui constituent le matériel étudié dans ce mémoire, issues de cette collaboration en ligne et créées, de fait, par la communauté.

Au sein de la communauté Python, les ressources de ce répertoire sont organisées en trois espaces distincts. Reprenant ici la terminologie de Sack *et al* (2006), nous nous

proposons de caractériser les différents lieux du cyberespace dans lesquels les participants évoluent. Nous présentons ainsi ces trois espaces d'activités : l'espace de documentation, l'espace d'implémentation et l'espace de discussion, chacun disposant de caractéristiques propres que nous développons ici.

D'abord, l'espace de documentation est le plus accessible par tout un chacun. Il est constitué du site web python.org ainsi que du *wiki* de la communauté. Sur ces sites sont répertoriés la documentation officielle de Python, des guides à l'usage du débutant ou du développeur, des récits au sujet du succès de Python dans divers environnements (entreprises, institutions, etc.). On y trouve également la liste des documents de travail de la communauté, des documents de spécifications et un répertoire des ressources disponibles (outil de *chat*, liste de diffusion, outil de gestion de version, plateforme de gestion d'anomalies, etc.). Cet espace agit comme dépôt de connaissance pour la communauté, à la manière d'une mémoire collective. Tout élément considéré comme digne d'intérêt et digne d'archivage par la communauté, pour consultation ultérieure, y trouve sa place. L'importance accordée par les membres de la communauté Python au maintien et au partage d'un savoir collectif démontre bien qu'il s'agit d'une communauté épistémique.

Ensuite, on trouve l'espace d'implémentation, fréquenté par les développeurs. D'une part, il est composé de l'outil de gestion de version du code source, appelé Mercurial et disponible sur hg.python.org, utilisé pour stocker les différentes versions des codes sources et garder trace de l'historique du code. C'est grâce à cet outil que le code est déposé (*commit*), archivé et constitué en un ensemble cohérent. D'autre part, on y retrouve la plateforme de gestion des *bugs* et des évolutions, appelée Roundup et disponible sur bugs.python.org, servant de système de gestion des tâches et de base de données des anomalies rencontrées. Ces deux outils recouvrent de manière relativement exhaustive la pratique des développeurs, soit l'implémentation logicielle et la résolution d'anomalies. Ainsi, les anomalies sont d'abord détectées dans l'outil de gestion des *bugs*, puis traitées et corrigées dans l'outil de gestion de version.

L'espace d'implémentation est donc fortement lié à la pratique logicielle des membres de la communauté Python, il joue ainsi le rôle d'un dispositif central au sein de la communauté de pratique.

Enfin, l'espace de discussion se compose de différentes listes de discussion. On trouve d'abord une liste orientée usage (la liste *python-list*) à destination des utilisateurs de Python et servant à l'entraide entre participants. Il y a également une liste orientée développement (la liste *python-dev*), qui est à destination des développeurs du langage et sert au développement du logiciel. Il s'agit ainsi d'un espace privilégié d'innovation. En effet, c'est sur cette liste que sont publiées les PEP, soit les propositions d'améliorations du langage suggérées par les utilisateurs selon des règles que nous définissons dans le chapitre ci-dessous. Notons simplement que ces améliorations visent la plupart du temps à répondre à des besoins spécifiques de la part d'un utilisateur pionnier. Ce sont donc ces utilisateurs qui, au sein de la communauté Python, sont instigateurs d'innovation. Enfin, il existe également une liste orientée proposition, dans laquelle les participants proposent de nouvelles idées pour le langage. Ces listes de discussions sont publiques et archivées par la communauté sur le web. D'autres types de listes peuvent faire l'objet de restrictions quant à leur publication et servent surtout d'outils de travail internes. C'est notamment le cas d'une liste orientée décision réservée aux développeurs (la liste *python-commiters*) et à travers laquelle les développeurs nomment leurs pairs. Encore une fois, la communauté innove par ce processus de nomination et de vote, nous y revenons plus loin.

Pour conclure, ces espaces, segmentés par l'activité qui s'y déroule, sont le fruit d'un travail de construction et de maintenance d'une infrastructure, qui elle-même soutient les activités de communication et de développement de la communauté. Ceci constitue une spécificité propre aux communautés de l'*open source* qui mettent en œuvre et utilisent des plateformes de collaboration distribuées (SourceForge, GitHub) (Cremer et Gaudel, 2004). Une particularité de la communauté Python est que ses

membres sont à la fois développeurs du langage et utilisateurs d'outils logiciels développés en Python et servant d'infrastructure aux activités de développement. Python est donc tout à la fois un langage, un logiciel et une communauté ; ces utilisateurs discutent de son développement à l'aide d'outils et sur des plateformes développées dans le langage lui-même. C'est ainsi que s'applique le principe de récursivité qu'affectionnent les *hackers* (Williams, 2002).

4.1.3 Ressources particulières dans l'espace de documentation

L'espace de documentation regroupe les ressources documentaires que la communauté choisit d'y ajouter. Au-delà des éléments décrits précédemment, il existe dans la communauté Python des ressources périphériques qui lui sont particulières. C'est notamment le cas du document intitulé le *Zen de Python*.

Le *Zen de Python* est l'expression, sous la forme d'un poème, de l'ensemble des règles idiomatiques que le langage encourage de par sa conception ainsi que le style de développement qu'il propose. Ces conseils, émis à l'attention du développeur Python par la communauté, illustrent la philosophie de ce langage. Ce document constitue une formalisation, sous forme humoristique, de la manière dont le langage devrait être utilisé par les développeurs. Il n'est pas anodin que ce soit un poème : tout comme la poésie est une célébration hautement codifiée du langage, le *Zen de Python* témoigne de la volonté d'élever cette pratique de programmation au statut d'art. En se conformant aux règles de la poésie pour créer un tout harmonieux et agréable à lire, l'auteur Tim Peters démontre de manière éloquente son propos. Le fond et la forme du *Zen de Python* servent donc un même but : expliciter les règles de programmation telles qu'entendues par la communauté Python. Il s'agit d'une forme tout à fait originale de présenter ces règles, à l'image du langage de programmation Python qui se veut clair et explicite dans sa forme.

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Le *Zen de Python* insiste donc sur les éléments clés qui caractérisent Python en regard des autres langages, et ce de manière normative : pour développer en Python, il faut accorder une importance toute particulière à la beauté, à la simplicité, à la lisibilité et à la cohérence du code. Ce texte est d'ailleurs intériorisé par les développeurs de Python qui y font largement référence dans leurs choix de conception. Plus que de simples règles, le *Zen de Python* se veut dogmatique, ce sont les principes fondamentaux sur lesquels repose la cohésion de la communauté, soudée autour d'idées directrices qui assurent l'unité et la cohérence du code.

Il existe d'autres ressources mises à la disposition de tous les développeurs qui souhaitent contribuer au langage : c'est le *Guide du développeur de Python*. Ce guide disponible en ligne se veut un manuel d'introduction à l'activité de développement du

langage de programmation Python. Cela en fait un document très précieux tant pour le développeur débutant que pour le développeur chevronné. Ce guide présente l'ensemble des instructions nécessaires à l'utilisation des outils techniques de la communauté, comme par exemple le dépôt central de code. Il explique notamment comment récupérer le code source du logiciel Python, constituer le logiciel, lancer les tests sur le logiciel et créer des correctifs. C'est un document tant didactique que normatif : il décrit la manière dont le développeur doit interagir avec la communauté, par exemple en indiquant comment et où trouver de l'aide et explique les différents outils de communication à la disposition de chacun ou les tâches qui doivent être effectuées par un développeur novice afin de s'intégrer à la communauté. Tout est ainsi prévu pour que Python s'organise en communauté.

En conclusion, les conditions de l'émergence d'une communauté de pratique, ou facteurs de cohérence, sont réunies au sein de la communauté Python : l'engagement mutuel dans la pratique, l'entreprise commune qu'est la production logicielle et le répertoire partagé au sein des trois espaces d'activités. La communauté Python crée ses propres règles et outils pour faire de leur pratique, le développement d'un logiciel, une activité partagée par tous. La communauté Python est une communauté de pratique dite pilotée, car elle dispose d'un leader qui anime et coordonne l'effort de chacun dans la communauté. Celui-ci joue le rôle de chef de projet au sein de la communauté, comme nous allons le voir dans la section suivante.

4.2 Les rôles dans la communauté Python

Notre analyse du corpus documentaire — constitué de courriels échangés dans le cadre du processus de développement en ligne ainsi que de pages des sites web de la fondation Python (PSF) — nous montre que la communauté Python est constituée de plusieurs individus aux rôles et aux fonctions différenciées. Pour caractériser ces

différents rôles, les acteurs emploient un ensemble de termes spécifiques à la communauté. Nous faisons ici la description de ce vocabulaire.

La communauté Python est d'abord et avant tout organisée autour de son chef de projet : Guido van Rossum. Les membres de la communauté le désignent sous le titre de BDFL, pour *Benevolent Dictator for Life* ou plus simplement Guido. La signification de ce titre est expliquée dans un document, la PEP 1 (voir Annexe A) :

> There are several references in this PEP to the "BDFL". This acronym stands for "Benevolent Dictator for Life" and refers to Guido van Rossum, the original creator of, and the final design authority for, the Python programming language. (PSF, PEP 1)

Il occupe un rôle spécifique et stratégique, notamment dans le cadre de processus de collaboration propre à la communauté et que nous expliciterons dans la section suivante. Le chef de projet (*project leader*) est relativement omnipotent, puisqu'il en est l'initiateur et assure sa direction depuis près de vingt-cinq ans.

En outre, au-delà de ce rôle particulier conféré à Guido, il existe plusieurs autres rôles dans la communauté qui chacun présentent des formes d'autorité variées :

1) le *release manager,* Barry Lastings, ou le gestionnaire des mises en production[9], est la personne qui — de manière relativement classique dans les équipes de développement logiciel — gère et fait la coordination des fonctionnalités intégrées dans les versions successives du code source. Le tout s'effectue en vue de la mise en production et de la constitution de versions spécifiques du code (activité de *versioning*) ;

2) les *project admins,* ou administrateurs du projet, forment un petit groupe d'une dizaine de personnes. Ce sont eux qui ont autorité pour fournir aux membres de la communauté le *droit en écriture* de codes sources directement sur le dépôt de code

---

[9] Le terme « mise en production » renvoie dans ce contexte au terme anglais *release* signifiant littéralement publier, diffuser et mettre en usage le code source.

central (*code repository*), et ce, sous diverses conditions. Ces administrateurs font ainsi la gestion de l'infrastructure du dépôt de code central ainsi que des autorisations de ce dernier ;

3) les *Python committers* sont les membres de la communauté ayant accès au dépôt de code en écriture. Ils y sont autorisés par les administrateurs du projet. Leur activité est l'écriture du code source, en leur nom propre, ou le report de fonctionnalités proposées et amendées par d'autres membres de la communauté (*patches*). Ce sont les *core developers* du projet, au nombre de cent quatre-vingt-trois, mais dont seulement un sous-ensemble est actif. Sur une période d'une année, en 2014, on note cinquante-trois *core developers* actifs, c'est-à-dire ceux ayant fait au moins un dépôt de code dans l'année.

4) Les *experts*, ou plus généralement ceux que l'on appelle les *module maintainers* (mainteneurs de module), sont des *Python committers* ayant un rôle d'expertise sur des portions spécifiques du code. Leur rôle est d'assurer le bon fonctionnement et la cohérence interne du code d'un module de Python. Ce sont les développeurs de référence de leur module, on peut donc considérer qu'ils sont les développeurs les plus expérimentés sur une ou plusieurs portions de code et qu'ils font autorité sur ces dernières. Ils sont au nombre de quatre-vingt-quatre développeurs.

5) Les *Python contributors* sont des développeurs ayant signé le *Contributor Agreement*, c'est-à-dire le formulaire permettant aux contributions d'être licenciées dans le cadre de la licence *open source* du logiciel (PSF) et en vue de l'écriture et la publication du code source. Les *Python contributors* forment un ensemble de développeurs qui n'ont pas accès en écriture au dépôt de code, mais qui participent toutefois à son écriture à travers l'outil de report des *bugs* et des évolutions. Leurs propositions de modifications du code source de Python (*patches*) sont validées et reportées par les *core developers*. Notons que tous les *core developers* ont, en outre, signé ce document et sont donc, par défaut, des contributeurs. Il est difficile de

quantifier le nombre de contributeurs au projet du fait de la nature même des plateformes de collaboration qui sont ouvertes, en ligne et accessible à tous. Toutefois, compte tenu de la taille du groupe de *core developers*, nous estimons ce groupe à quelques milliers d'individus.

6) Les *Python users* sont les utilisateurs de Python, c'est-à-dire des utilisateurs du langage de programmation (de l'ordre de plusieurs millions à travers le monde). Ce sont donc des développeurs *en* Python, mais qui ne participent pas directement à l'écriture du code source. Les utilisateurs dits « actifs » ont des activités variées relatives au projet et reliées aux différents processus de gestion du code et de collaboration au sein de la communauté Python, tels que définis dans la section suivante.

Il est important de noter que ces rôles et leur dénomination correspondent au vocabulaire de travail des acteurs impliqués au sein de la communauté Python, car il n'existe pas de documentation en tant que telle sur ces rôles. Notre tâche consistait donc à reconstruire l'organigramme effectif de cette organisation, sans que celui-ci n'existe à priori. On peut donc penser que ses membres s'auto-organisent en des rôles clés qui sont implicites.

4.3 Une communauté hiérarchisée

4.3.1 Une structure pyramidale

L'étude de notre corpus, constitué des documents collectés en ligne, tant sur les sites web de la communauté Python que lors des échanges des membres de la communauté sur les listes de discussions, nous permet de proposer une représentation graphique des différents rôles au sein de la communauté. A l'issue de notre analyse des rôles, nous proposons une pyramide, que l'on peut également voir comme des strates

superposées. Ces strates représentent les groupes décrits précédemment, définis à la fois par leurs rôles, leur niveau d'autorité et l'étendue de leurs droits d'accès. Concrètement, la représentation graphique développée présente deux caractéristiques : l'axe horizontal représente visuellement le nombre de membres qui prennent part au groupe ; l'axe vertical présente, du bas vers le haut, le niveau d'autorité conférée aux membres de ces groupes. Ainsi, reprenant donc ces rôles dans la figure qui suit :

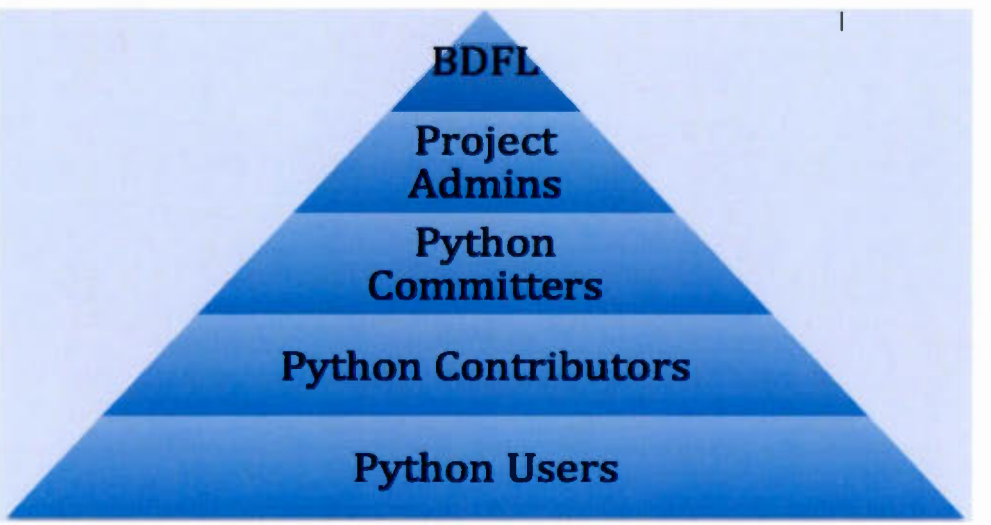

Figure 4.1 : Les membres de la communauté

Pour mieux comprendre les liens entre ces différents groupes et la dynamique particulière au sein de laquelle ils évoluent, nous mettons ici en perspective la représentation graphique de la communauté Python avec une analyse plus large, issue de la littérature, et qui porte sur les communautés *open source* en général. En effet, on constate une nette évolution au sein de cette littérature. Si les auteurs ont d'abord décrit des communautés *open source* dénuées de hiérarchie (Raymond, 2000 ; Ljungberg, 2000), des auteurs plus récents (Meyer et Montagne, 2007) reconnaissent l'importance de la hiérarchisation des rôles par l'autorité.

D'abord, ces auteurs font référence au modèle du bazar présenté par Raymond (2000) : « Les échanges réciprocitaires entre les développeurs bénévoles constituent un réseau de relations sociales horizontales et décentralisées qui à l'aspect d'un "bazar". » (Meyer et Montagne, 2007, p. 395). En d'autres mots, les auteurs affirment que, dans le contexte du développement du logiciel libre, les réseaux de développeurs tendent à s'organiser horizontalement. C'est ce que nous entendons par les « groupes », ou les « strates ». Cependant, notre analyse de la communauté Python, sous l'angle de la communauté de pratique, montre que cette communauté est hiérarchisée par l'autorité et relativement centralisée. Il existe des *liens sociaux*, ces liens forts rassemblant les développeurs participant activement au développement entre eux et procurant une stabilité identitaire à la communauté ; or, ces liens ne sont pas uniquement horizontaux, mais également *verticaux*. En effet, les *hackers* ne rejettent pas systématiquement l'autorité, dès lors qu'elle n'est pas considérée comme coercitive et est légitime (Williams, 2002). Dans la communauté Python, cette autorité est d'ordre technique, mais également faite d'aptitudes communicationnelles et de leadership. Meyer et Montagne remarquent en effet « qu'il n'existe pas de communauté du libre performante qui ne soit dotée d'une personnalité forte, qui coordonne les croyances individuelles autour d'un projet commun, fait germer de nouvelles idées et suscite la formation de communauté de pratique » (p. 401). C'est le rôle que prennent à la fois le BDFL et dans une moindre mesure le « champion », comme nous aurons l'occasion de développer dans le chapitre suivant.

Ainsi, sur la base de nos résultats, il nous apparait qu'il existe une hiérarchie entre développeurs Python. Ces développeurs sont dotés d'un capital culturel et technique, mais également d'un capital social (Wang, 2005 ; Burt, 1995 ; Bourdieu, 1980) qui leur est propre et qui se construit lors des expériences de développements collaboratifs. Le capital social influence les motivations intrinsèques des développeurs, il est fonction du capital technique, de la réputation, des capacités relationnelles et est le produit de leurs transformations. Reprenant la formulation de

Meyer et Montagne, on peut ainsi dire que le capital social est le « ciment des communautés du libre » (Meyer et Montagne, 2007, p. 394). Ce capital est reconnu par les pairs de manière tacite, mais mène à des choix explicites.

En effet, dans la communauté Python, si un développeur n'est pas reconnu par ses pairs comme faisant partie du groupe des contributeurs actifs — et donc doté d'un capital social inhérent à ses interactions sociales — il ne peut prétendre à l'écriture d'une proposition d'amélioration. C'est notamment de cette manière que se traduit explicitement cette notion au sein de la communauté. Autrement dit, écrire une proposition d'amélioration, c'est officialiser un statut qui s'acquiert dans la communauté : celui de rédacteur d'une nouvelle fonctionnalité pour le langage de programmation. C'est un privilège acquis par l'intermédiaire de compétences techniques et sociales et qui peut mener à l'appartenance au groupe des *core developers*. Rappelons que tout développeur peut se proposer pour écrire une PEP, mais que toutes les idées ne sont pas validées par la communauté.

Ce filtrage des fonctionnalités, basé sur la rationalité des propositions (c'est-à-dire l'action rationnelle et le pragmatisme technique), semble également faire office de sélection des profils de développeurs. Il faut en effet (1) être capable d'argumenter sur le bien-fondé d'une nouvelle fonctionnalité (2) faire preuve d'une excellente connaissance transversale des fonctions de Python pour proposer quelque chose qui n'existe pas et qui serait susceptible d'être intégré et (3) savoir construire des consensus autour de ses propositions et susciter des opinions positives au sujet de ses idées. Ainsi, les développeurs Python dotés de compétences techniques se construisent un capital social à travers l'activité de développement et de conception collaborative du langage. Ces deux formes de capitaux (technique et social) hiérarchisent la communauté de manière « informelle » et sous-tendent les liens inhérents entre développeurs.

Notons d'ailleurs que la notion de hiérarchie est directement incluse dans le modèle classique de l'oignon présenté à la figure 1.1, mais elle est, de fait, informelle : la frontière des groupes étant peu formalisée. Nous montrons par la suite que, dans le cas de la communauté Python, la frontière entre les groupes de développeurs *Python committers*, *Python contributors* et *Python users* est beaucoup plus formelle et rigide, elle est définie par des processus de transition formalisés par la communauté : la signature du Formulaire du Contributeur Python (voir Annexe B) et l'obtention des droits en écriture sur le dépôt de code principal.

La notion de hiérarchie dans les communautés *open source* n'est pas nouvelle, comme l'ont montré les travaux de Gacek et Arief en 2004. Cette idée de hiérarchie présente une distinction majeure face au modèle du bazar, plat et de fait peu hiérarchisé. Les auteurs insistent notamment sur la différentiation des activités entre développeurs (Gacek et Arief, 2004). En l'occurrence, ils expliquent que les développeurs ont des activités différentes, auxquelles on attribue un statut différencié et qui contribue à l'élaboration d'une hiérarchie. D'autres auteurs ont également suggéré l'idée d'une structure hiérarchique en étudiant différentes communautés de développement logiciel *open source* comme Apache et Linux (Crowston et Howison, 2006 ; Mokus *et al*, 2000, 2002 ; Moon et Sproull, 2000). Ils ont constaté que l'activité de développement et de conception est centralisée autour des *core developers*, alors que l'activité de rapport de *bug* est largement décentralisée et menée par un groupe plus large d'utilisateurs *actifs*. Nous observons dans le cadre cette étude une situation semblable, mais avec une différence notable. Dans le cas de la communauté Python, l'activité de conception, centrée autour du chef de projet, est plus largement ouverte aux membres contributeurs actifs, par l'intermédiaire du processus de proposition d'amélioration, par comparaison avec d'autres communautés.

4.3.2 Une autorité détenue par le BDFL

La communauté Python n'est pas à proprement parler une démocratie, car l'autorité procédurale est conférée au BDFL. En effet, Guido van Rossum est l'autorité finale en ce qui concerne le *design* du langage de programmation. Pourtant, des mécanismes démocratiques sont à l'œuvre : le débat, la consultation et le vote en ligne. Bien que la décision finale de conception revienne au BDFL, une forme de participation permet d'influencer la décision et de rendre le choix du BDFL *community aware*, c'est-à-dire à l'écoute de la volonté de la communauté. Le rôle du BDFL est à la fois celui d'un manager et d'un expert de la conception logicielle. La littérature sur le rôle spécifique du BDFL n'est pas très étendue, mais quelques auteurs en traitent brièvement dans le cadre de leurs travaux. Ferrary et Vidal en 2004 indiquent ceci :

> [L]es « responsables » qui dynamisent la communauté, structurent et coordonnent les efforts de l'ensemble de ses membres ne sont donc nullement nommés, mais émergent naturellement de par la qualité de leurs contributions et leur implication à la vie communautaire et également par la reconnaissance de leurs compétences par la communauté. Cette légitimité naturelle fonde leur « autorité » et leur position hiérarchique dans la communauté » (p. 318)

Ce mode de gouvernance est qualifié par Foutel de « gouvernance charismatique ». Ainsi, « la communauté accepte ce despotisme puisqu'elle le considère comme un « dictateur bienveillant » (Fogel, 2010 : 60) « qui maîtrise parfaitement l'œuvre » (Foutel, 2012, p. 17). Ces auteurs vont dans le même sens que précédemment élaboré, à savoir que l'autorité au sein des communautés *open source* émerge à la fois du capital technique et du capital social.

Le capital technique de Guido van Rossum, en ce qui concerne le langage Python, est maximal ; nul ne saurait mieux maitriser le langage que son créateur. Il possède donc une autorité complète sur le projet parce qu'il le maîtrise totalement. Il est plus difficile de distinguer ce qui, au sein du capital social dont jouit le BDFL, provient

strictement de ses aptitudes relationnelles et communicationnelles en regard de son statut. On peut toutefois supposer qu'avant même de devenir la figure emblématique qu'il est aujourd'hui, Guido van Rossum a dû faire preuve d'un charisme certain pour rallier tant de développeurs à son projet. C'est donc à la fois par son capital technique et social que l'autorité de Guido est acceptée par la communauté.

Il est relativement clair à la lecture des échanges sur les listes de discussion que Guido agit en tant que chef de projet, coordonne, prend des décisions et anime les débats. Cependant, une certaine forme de délégation de l'autorité procédurale est à l'œuvre dans la communauté Python. D'abord, dans le processus de proposition d'amélioration du langage, l'autorité peut être conférée à un délégué (*BDFL Delegate*). Le rôle de chef de projet est alors assumé par un autre individu et Guido se décharge du suivi, de l'animation et de la validation d'une proposition d'amélioration du langage donnée. De la même manière, certains rôles présentent une forme de délégation d'autorité de forme collégiale. C'est le cas des éditeurs de PEP (*PEP Editors*) qui valident les propositions d'amélioration du langage. Enfin, une autorité technique est conférée au « champion », auteur d'une proposition.

En conclusion, la figure du « dictateur bienveillant », dans le cas de la communauté Python, doit être nuancée. Certes, l'autorité du chef de projet est incontournable — sauf dans le cas d'une délégation spécifique — et pourtant, les processus de collaboration sont régis par des principes de gouvernance démocratiques : validation collégiale et reconnaissance par les pairs. On peut ainsi qualifier cette structure de méritocratie tintée de démocratie participative.

4.4 Les formes de participation et d'intégration à la communauté

L'objectif de cette section est de décrire les activités qui constituent la pratique de la communauté Python et d'articuler les différents rôles qui lui sont associés. La

participation et l'intégration au sein de la communauté Python sont graduelles. Pour s'intégrer, il faut d'abord développer dans le langage en question, c'est-à-dire pratiquer la programmation informatique, ce qui fait partie intégrante du domaine de la communauté. Un membre de la communauté gagne ensuite progressivement en autorité et en notoriété sur la base quantifiable de ses contributions. Un développeur commence par rapporter des *bugs*, puis les corrige, fait la revue des corrections des autres membres et enfin les intègre dans le plus vaste ensemble de code : il s'agit là du parcours type que nous allons détailler dans ce qui suit.

4.4.1 Déboguer le logiciel

4.4.1.1 Rapporter un bug dans le logiciel

La pratique de « rapport de *bug* » dans le logiciel Python est centrale. C'est par elle qu'un individu révèle un problème, un dysfonctionnement ou un comportement non souhaité dans le logiciel dans le but de l'exposer à la communauté. Pour révéler un *bug*, il faut se rendre sur la plateforme web bugs.python.org (*bug tracker*) et ouvrir un ticket (*issue*) auprès des développeurs. La plateforme est publique et ouverte à tous ; tout individu qui s'y inscrit peut rapporter un *bug*.

Le rapport de *bug* nécessite de respecter plusieurs étapes. Il faut tout d'abord vérifier sur la plateforme si le problème n'a pas déjà été rapporté. Pour ce faire, on trouve sur l'interface un moteur de recherche dédié à cette fonction. Nul besoin donc d'être un développeur sénior pour rapporter une anomalie, les outils de collaboration sont véritablement construits pour être le plus accessible possible, dans une optique inclusive. Si le *bug* n'existe pas dans la base de donnée de *bugs* (*issue database*), le problème peut être rapporté. Notons qu'il n'est pas possible de rapporter de problème de manière anonyme, il faut se connecter à la plateforme. On doit alors renseigner divers champs, tels que le titre, le type de problème et le commentaire. Chaque

rapport de *bug* est assigné à un second développeur plus expérimenté, qui détermine ce qui a besoin d'être fait afin de corriger le problème.

Notons que tous ces éléments sont codifiés dans le *Guide du développeur de Python*, disponible dans l'espace de documentation. La documentation officielle indique également qu'il est possible de fournir soi-même un correctif pour le *bug* rapporté, selon son propre niveau de compétences, de connaissances et d'expérience. Un individu rapportant un *bug* sur la plateforme de rapport de *bugs* est appelé *Bug Reporter* ou *reporter* (voir Annexe C). C'est un membre actif de la communauté appartenant au groupe des *Python users*.

Pour savoir comment rapporter un *bug*, les développeurs se rapportaient initialement à une PEP informationnelle sur ce processus, « PEP 3 -- Guidelines for Handling Bug Reports ». Avec le développement progressif du *Guide du développeur de Python*, devenu la documentation officielle à ce sujet, la PEP ainsi supplantée est devenue obsolète. Cela signifie que ce processus a d'abord fait l'objet d'une codification explicite par la communauté, puis qu'il est devenu plus implicite au fil des pages de la documentation, suggérant ainsi l'apprentissage des membres de la communauté.

4.4.1.2 Corriger un bug dans le logiciel

Dans un deuxième temps, les *bugs* rapportés doivent être corrigés selon leur type et leur sévérité (*priority*). Pour ce faire, un développeur Python doit fournir un correctif appelé *patch.* Fournir un *patch* consiste à : 1) développer le correctif sur son poste de travail ; 2) créer un fichier *de différence* entre la version originale et la version améliorée contenant les modifications apportées, afin qu'elles puissent être répliquées sur le dépôt central de code ; 3) déposer ce fichier de différence, appelé .diff ou .patch, sur la plateforme de gestion de *bugs*.

L'écriture d'un *patch* doit suivre les conventions d'écriture des PEP 7 (*Style Guide for C Code)* et 8 (*Style Guide for Python Code).* Le respect de ces conventions est un

élément important définissant la qualité de la contribution et de la bonne intégration du membre à la communauté. Ainsi, l'écriture de *patches* s'inscrit dans le processus plus large d'écriture du code source, de manière décentralisée, communautaire et par de multiples agents développeurs. C'est ce type de conventions qui permet à la communauté de fonctionner correctement : d'intégrer des correctifs écrits par des auteurs multiples et d'intégrer ses membres à la communauté. L'activité d'écriture de *patches* est réalisée par tous types de développeurs Python, utilisateurs actifs, contributeurs Python et *core developers*.

La procédure pour créer un *patch* est décrite dans le *Guide du développeur de Python* et toutes les lignes de commandes nécessaires sont documentées. Ceci constitue une formalisation de la pratique.

4.4.1.3 Réviser et intégrer le correctif

Dans le cadre de la correction d'anomalie dans le langage, un processus de revue par les pairs est mis en œuvre. C'est ce que l'on appelle la revue de code (*code review*). Dans le domaine de la programmation informatique, il s'agit d'une démarche collaborative visant à améliorer la qualité du logiciel. Cette pratique prend forme sur la plateforme de gestion des *bugs* et des évolutions.

Après qu'une anomalie s'est vue corrigée par un membre de la communauté, c'est-à-dire qu'un *patch* a été proposé, son statut passe à celui de « *patch review* ». À cette étape, un autre membre de la communauté est chargé de relire, revoir, parfois annoter et modifier le code source proposé. Cette pratique de révision du correctif est mise en œuvre avec deux objectifs précis : d'une part, il faut limiter les erreurs de codage ou de conception et d'autre part, il importe de rendre le code source de Python relativement homogène dans son style d'écriture. Une fois révisé, le correctif est intégré dans le dépôt central de code par un développeur expérimenté, possédant le droit en écriture sur le dépôt. L'intégration du code est la suite logique de sa révision, c'est donc le plus souvent le même développeur qui s'en charge.

On constate ainsi que les anomalies sont gérées en quatre étapes ; elles sont respectivement rapportées, corrigées, révisées, puis intégrées. Ce processus peut être parfois long (de l'ordre de plusieurs mois). Chaque étape fait l'objet d'un message, ou enregistrement (*record*) dans la plateforme de gestion de *bugs* et des anomalies. Ainsi cette plateforme sert d'outil de gestion des connaissances et de support aux activités de correction, de révision et d'intégration.

Il est intéressant de noter que quiconque inscrit sur cette plateforme, et donc possédant les autorisations minimales, peut se permettre d'intervenir et de donner son avis. On peut supposer que les avis émis auront plus ou moins de poids selon l'expertise et la notoriété de leur émetteur, ce que notre analyse ultérieure des PEP nous permet de valider. Tout au long de ce processus, les développeurs concernés entretiennent un dialogue via la plateforme de gestion de *bugs* et d'anomalies. Ce dialogue, souvent technique, est le fruit d'une expertise mise en œuvre par les membres de la communauté.

4.4.2 Devenir *Python contributor*

Accumulant de l'expérience dans les activités de gestion des *bugs*, un développeur Python devient contributeur par sa participation et sa pratique. Ce sont ces deux éléments qui fondent l'intégration du programmeur. De plus l'appartenance au groupe des *Python contributors* est formalisée par un mécanisme de collaboration particulier au sein de la communauté Python : celui de la signature du « formulaire du contributeur Python ». Le formulaire du contributeur Python (*PSF Contributor Agreement for Python*) est un document performatif, c'est-à-dire qu'en le signant on obtient un rôle et un statut.

Proposé par la Fondation Python et propre à ce langage, ce document légal a comme objectif d'éviter les principaux problèmes de licence liés au logiciel Python et à son code source ouvert :

> Licensing
>
> For non-trivial changes, we must have your formal approval for distributing your work under the PSF license. Therefore, you need to fill out a contributor form which allows the Python Software Foundation to license your code for use with Python (you retain the copyright).
>
> Note : You only have to sign this document once, it will then apply to all your further contributions to Python (PSF, Python Developer's Guide).

Le logiciel Python est un logiciel libre et *open source*. Cela signifie que le logiciel ainsi que son code source sont disponibles et distribués sous une licence spécifique. Celle-ci est compatible avec les définitions du logiciel libre de la Free Software Foundation et du logiciel *open source* de l'Open Source Initiative. Il s'agit de la licence PSF *(Python Software Foundation License Agreement)*, compatible GPL selon la documentation officielle de Python.

Ce formulaire permet à chaque contributeur de formaliser les termes selon lesquels ils rendent disponibles leurs contributions à la PSF. En même temps, les contributeurs demeurent les détenteurs de leur droit de propriété intellectuelle (*copyright*). En effet, il n'est jamais demandé de céder son droit de *copyright* au profit de la PSF.

Dans les faits, ce formulaire est proposé à tous développeurs ayant fait une contribution non triviale et devant être incorporée (*committed*) dans le logiciel. C'est la signature de ce formulaire qui fait d'un individu, auparavant un développeur Python non contributeur, un contributeur Python à part entière (*Python contributor*). Chaque contributeur doit alors insérer dans le code source de chacune de ses contributions, à côté de sa notice de *copyright,* la mention suivante : « *Licensed to PSF under a Contributor Agreement.* ».

On notera qu'une contribution peut également être incorporée dans le logiciel par le biais d'un *core developer* (seuls développeurs de Python ayant un accès en écriture au dépôt de code), une mention dans les commentaires de la contribution sera alors ajoutée au profit du contributeur. Le *Contributor Agreement* est un élément central dans le processus d'intégration des modifications du code source et, de fait, de l'intégration des membres dans la communauté Python.

4.4.2.1 L'accompagnement des nouveaux développeurs

On constate l'existence d'un processus d'accompagnement des nouveaux développeurs de Python, principalement les *Python contributors* novices, par des *core developers* (ou *Python committers*) qui sont disponibles pour les aider. L'accompagnement est un élément clé des communautés de pratique. Leur champ d'action comprend différents aspects internes à la communauté, tant procéduraux que techniques. Ces bénévoles sont appelés les *Python mentors*. Dans le *Guide du développeur de Python* on peut lire :

> If you are interested in improving Python and contributing to its development, but don't yet feel entirely comfortable with the public channels mentioned above, Python Mentors are here to help you. Python is fortunate to have a community of volunteer core developers willing to mentor anyone wishing to contribute code, work on bug fixes or improve documentation. Everyone is welcomed and encouraged to contribute. (PSF, Python Developer's Guide)

Les mentors possèdent un site dédié (pythonmentors.com) où l'on explique l'objectif du *Python Core Mentor Program* : aider les personnes intéressées à contribuer au développement de Python (*Python Core*) en offrant un endroit accueillant pour poser des questions auprès de développeurs plus expérimentés dans la communauté. Cet accompagnement est partie prenante de l'intégration des nouveaux développeurs.

Il existe une liste de discussion dédiée, *core-mentorship*, dont les archives ne sont accessibles qu'aux membres. On peut principalement y lire des courriels de présentations de nouveaux développeurs cherchant une aide sur un sujet spécifique ou un mentor. La principale ressource disponible au développeur novice est le *Guide du*

développeur de Python (*Python Developer's Guide*), un manuel dédié à leur intention (voir Annexe D).

> Python Developer's Guide
>
> This guide is a comprehensive resource for contributing to Python – for both new and experienced contributors. It is maintained by the same community that maintains Python. We welcome your contributions to Python!
>
> Contributing
>
> We encourage everyone to contribute to Python and that's why we have put up this developer's guide. If you still have questions after reviewing the material in this guide, then the Python Mentors group is available to help guide new contributors through the process. The Developer FAQ is another useful source of information. (PSF, Python Developer's Guide)

4.4.3 Devenir *Python committer*

Une fois qu'un développeur Python est contributeur, il peut avoir l'opportunité de changer de statut. L'« autorisation d'écriture sur le dépôt de code » (*grant commit privileges to code repository*) est le processus formel par lequel un *Python contributor* devient *Python committer*. En d'autres termes, il s'agit du processus par lequel un développeur Python actif sur la plateforme de rapport des *bugs* et des évolutions (*bug tracker*) devient membre à part entière du groupe des *core developers*. Cela s'articule principalement autour de la proposition de la candidature d'un développeur par l'un de ses pairs et respectant plusieurs conditions qui vont légitimer ce choix.

Pour obtenir cette autorisation (*commit privileges*) et ainsi changer de « groupe », les développeurs doivent contribuer de façon suffisante au projet : leurs contributions doivent être de qualité, nombreuses, et ne plus nécessiter de révisions directes de code (*code review*). En effet, on constate que ce sont le plus souvent des développeurs réunissant ces critères qui voient leur candidature proposée par leurs pairs.

Ce processus est ouvert, c'est-à-dire public et en ligne. Il s'organise sur la liste de discussion *python-committers* qui regroupe tous les membres de Python disposant du droit en écriture sur le dépôt central de code (*Python committers)*. Le processus est le suivant : d'abord, un membre de ce groupe propose (ou nomme) un membre de la communauté, en général un *Python contributor* actif et remarqué, pour qu'une autorisation en écriture sur le dépôt de code lui soit accordée. Cette proposition doit être motivée par le proposant sur la base des contributions du développeur nommé.

Par exemple, on peut lire les propos de Daniel (*core developer*), proposant Bernard (contributeur) et justifiant la qualité de son travail sur les *patches* à la fois en qualité d'auteur, mais aussi en tant que correcteur :

> `Proposing Bernard for push privileges`
>
> `I'd like to propose that we give Bernard push privileges. He's provided a number of good quality patches and done good review work on other issues, and has been an active contributor for quite some time.`

Un autre exemple nous montre la variété des types de participations dont il est possible de faire preuve pour l'obtention de cette autorisation. En effet, Bart propose Robert sur la base de la qualité de sa participation sur la liste de discussion *python-dev* et l'écriture de PEP :

> `Proposing Robert for commit privileges`
>
> `Robert has been participating on python-dev for quite a while, he is a committer on pip, and (co-)author on 5 PEPs.`

Les types de contributions justifiant la nomination peuvent être nombreux allant de l'écriture de code et *patches,* à l'écriture de PEP, en passant par une intense participation à la liste de discussion *python-dev*.

C'est la première étape de justification de la validité d'une telle demande d'autorisation : s'assurer que le contributeur participe suffisamment à la communauté pour que son statut soit modifié. Ensuite, cette autorisation doit être justifiée par son utilité pour la communauté. Il s'agit ici de la délester d'un travail, la revue de code,

devenue inutile compte tenu de la qualité des modifications proposées par le développeur nommé. Ainsi, l'on cherche à promouvoir des individus qui font preuve d'une bonne qualité de code et surtout qui peuvent se montrer autonomes dans leurs activités. Par exemple, cette autorisation peut être utile pour une contribution directe à l'implémentation d'une PEP.

Reprenant le cas de Bernard, les propos de Daniel justifiant l'utilité d'accorder cette autorisation reflètent ce pragmatisme :

> "IMO we've reached the point where it will be easier to let him push patches."

Enfin, c'est un privilège et une promotion au sein de la communauté comme l'indique Ethan, un autre *core developer* au sujet des nombreux développeurs de la communauté : *« Developers have the same range in talents and concerns as other folks; not every developer is a "core developer". Likewise, it is possible to be extremely helpful without being a core developer. ».*

Parfois, cet argument est directement stipulé lors de l'appui de la nomination, comme dans le cas de Henry, proposé par John et indiquant que c'est une promotion :

> For anyone looking, he has posted on the tracker as both 'hseliv' (2 years) and 'Henry.Seliv' (3 1/2 years). He has submitted patches on about 12 issues, 7 closed, and commented on another 10. These are mostly issue I have not be active on, but the numbers are typical for when we think about promoting someone.

L'autorisation est donc discutée en fonction des contributions du développeur nommé, de sa connaissance des processus de collaboration du projet (*workflow, code review*) et de son autonomie. Par ailleurs, le membre qui fait la proposition peut ensuite agir comme mentor et fournir une aide spécifique au développeur, nommé *Python committer* débutant. Il le soutient en ce qui a trait au code, aux aspects techniques et aux processus de collaboration de la communauté Python.

Par exemple, une nouvelle fois dans le cas de Bernard, Daniel se propose pour être son mentor :

> "I volunteer to act has his mentor for learning how to push to the repository & co."

La proposition d'autorisation est simultanément votée par la communauté des *Python committers* (voir Annexe E). Un nombre suffisant de votes favorables est nécessaire pour accorder l'autorisation au développeur, mais ceci n'est pas explicitement codifié. En effet, le nombre de votes, de même que le niveau de notoriété des votants varie selon les cas ; on remarque des différences flagrantes quant à la participation selon les individus nommés. Parfois, seuls deux à trois avis favorables peuvent suffire, surtout s'ils proviennent de développeurs renommés comme Guido van Rossum ou des membres de l'équipe d'administrateurs (*project admins*) qui « valident » cette proposition. Dans d'autres cas, on assiste à une liste importante d'avis favorables. Cela montre souvent la notoriété du contributeur nommé et la qualité des relations interpersonnelles qu'il a su tisser lors de ses interventions. Il n'est pas rare de voir des messages de sympathie de la part des membres du groupe accueillant ainsi un nouveau membre.

A l'issue du vote, le développeur ainsi nouvellement autorisé doit en conséquence envoyer sa clé cryptographique publique pour assurer des échanges authentifiés et sécurisés sur le dépôt. Daniel l'explique bien lors du résultat favorable en ce qui concerne Bernard :

> "I have emailed Bernard to (among other things) send in his keys and introduce himself here after signing up."

Dans certains cas, il peut arriver que la non-objection des *core developers* à la nomination d'un développeur puisse suffire à sa promotion, mais ceci est relativement rare. Nous observons qu'en cas de refus, l'objection doit être motivée sur la base de la qualité des contributions du développeur, de ses compétences en terme de communication ou de connaissance des processus internes de gestion. Un mentor peut alors accompagner le développeur avant que l'autorisation d'écriture ne

soit accordée dans le but d'améliorer la qualité de ses contributions ; citons par exemple Serhy à propos de Vajrasky et sa désapprobation, qu'il justifie adroitement :

> Vajrasky is good candidate. He is very active and interested in Python maintaining, he is respondable. he makes review of others code. But his code still not mature. He is often doesn't noticed many details in first versions of his patches. He just lacks experience. I believe that a year late he will be more experienced.
>
> Perhaps a mentor would help, but every Vajrasky patch, even simplest, should be reviewed. And in this case there are no many benefits from commit right (except moral encouragement).
>
> Sorry, I'm -0.1 for right now.

Dans ce cas précis, la désapprobation est certes claire, mais néanmoins teintée d'espoir pour les mois à venir et le mentor pourra aider. Donner ce droit trop tôt n'aiderait pas la communauté à mieux travailler. A la fin de cette citation, on voit que Serhy indique sa désapprobation par une valeur décimale négative (« `I'm -0.1 for right now` »). Il s'agit d'une pratique courante, le vote, que nous explicitions dans le chapitre suivant.

4.5 Le cadre normatif de la pratique

Le développement logiciel est une pratique normée par des conventions d'ordre technique, cependant elle est également normée socialement. Comme le montrent Meyer et Montagne (2007), l'action collective des communautés du logiciel libre et *open source* est auto-régulée par des normes sociales issues de la culture *hacker*. Reprenant Levy (1984), les auteurs définissent cette culture comme étant porteuse des valeurs suivantes : « rejet de la hiérarchie, la promotion de la décentralisation, le partage de l'information et l'attachement à la communauté » (Meyer et Montagne, 2007, p. 117). Ces valeurs créent des normes sociales qui structurent la communauté. Cependant, dans le cas de Python, nous observons qu'elles ne sont pas toutes reprises par la communauté ; il s'agit d'une communauté *open source* aux valeurs quelque

peu différentes. En l'occurrence, tel qu'expliqué précédemment la communauté Python est hiérarchisée, relativement centralisée et accorde une certaine valeur à l'autorité. Le partage de l'information et l'attachement à la communauté sont toutefois des valeurs fortes qui sont intégrées par ses membres et qui marquent leurs pratiques..

Par ailleurs, nous montrons dans cette recherche que cette pratique est également normée par des valeurs techniques. Le *Zen de Python*, et dans une certaine mesure le *Guide du développeur de Python*, représentent ces normes techniques, codifiées explicitement et qui sont alors hissées au rang de valeurs. Ces valeurs techniques définissent la manière dont les développeurs interprètent et conçoivent l'activité de conception logicielle. Elles sont également ancrées dans une culture de l'ingénierie informatique (simplicité, modularité, beauté du code, pragmatisme et rationalité technique). Les développeurs suivent les « règles » instaurées par Tim Peters, l'auteur du *Zen de Python* et *core developer*, sous la forme d'un poème et ces règles sont acceptées de manières tacites par l'ensemble des développeurs de la communauté comme un ensemble d'instructions utiles et qui guident le développement du langage de programmation. Ces règles ne sont pas à suivre au pied de la lettre sans discernement, elles sont soumises à interprétation et doivent servir de canevas aux choix de conception.

4.6 Synthèse

Dans ce quatrième chapitre, nous avons dressé le portrait de la communauté Python. Cette communauté est rassemblée autour d'une même pratique et d'un domaine spécifique, la programmation informatique. Elle partage des ressources qui lui sont propres, créées par les membres de la communauté. Ces ressources sont organisées en trois espaces distincts : l'espace de documentation, constitué des sites web de la

communauté ; l'espace d'implémentation, composé de l'outil de gestion des versions du code source et de la plateforme de gestion des *bugs* et des anomalies ; et l'espace de discussion qui se compose des différentes listes de discussion par courrier électronique. Des ressources particulières dans l'espace de documentation servent à baliser le développement du langage, c'est le cas du *Zen de Python*. Ce poème exprime les règles idiomatiques que les concepteurs recommandent de respecter pour développer en Python : beauté, simplicité et cohérence du code. Ce sont des caractéristiques élevées au rang de valeurs par les développeurs et fondamentalement intériorisées par ces derniers.

Dans la communauté Python, le chef de projet, Guido van Rossum, est l'autorité finale en ce qui a trait à la conception du langage. Cependant, il existe des rôles et statuts spécifiques. C'est le cas du *release manager* qui gère les mises en production des codes sources et des administrateurs (*project admins*) qui gèrent les accès au dépôt de code central et l'infrastructure du projet. Ensuite, les développeurs sont répartis en plusieurs groupes en fonction de leur degré d'autorité : les *Python committers*, dont le rôle est de développer et concevoir le langage Python, les *Python contributors,* dont le rôle de participer au développement en proposant des modifications (*patches*) et enfin les *Python users*, dont le rôle est de révéler des *bugs* ou anomalies dans le fonctionnement du logiciel. La communauté auto-régulée apparaît ainsi fortement hiérarchisée. L'intégration et la participation des membres de la communauté est graduelle et suit une voie spécifique : on est d'abord utilisateur du langage, puis contributeur en signant le formulaire du contributeur et enfin, on devient *committer* ou *core developper* à la suite d'une nomination par un pair et par le vote des membres de ce groupe.

CHAPITRE V

L'INNOVATION COMME PROCESSUS COLLECTIF

Dans ce cinquième chapitre, nous nous proposons de décrire les processus d'innovation collectifs qui caractérisent la communauté Python. Nous présentons d'abord le vote en ligne comme outil participatif et le processus de proposition d'amélioration du langage. Puis, prenant deux cas d'évolutions majeures du langage de programmation, nous reconstruisons les interactions, lors des échanges par courriels sur les listes de discussion, afin de saisir au mieux l'innovation « en train de se faire ». Enfin, nous décomposons ce processus en différentes activités afin de montrer ses caractéristiques innovantes.

5.1 Le vote en ligne : outil participatif de décision et de consultation

Dans le cadre de la communauté Python, les choix de conception se font de manière participative, collective et communautaire. En effet, il est important pour la communauté que l'on prenne en compte la volonté et les différents points de vue de ses membres, de manière plus ou moins formelle, sur les aspects qui font débat. La communauté est largement distribuée géographiquement, alors pour s'en assurer, elle a adopté un mécanisme de vote en ligne. Ce mécanisme de représentativité de l'opinion des membres de la communauté Python, bien qu'inspiré d'un mécanisme semblable existant au sein de la communauté Apache, est propre et spécifique à la communauté dans son utilisation. Il prend forme sur les listes de discussion que nous

avons étudiées. C'est par ce mécanisme que la voix de la communauté est entendue par le BDFL ou autre autorité désignée. C'est un outil important même s'il n'a pas de valeur contraignante, ni de valeur « officielle ». Il est reconnu tacitement par l'ensemble des membres de la communauté. Ce mécanisme est codifié par la communauté dans une PEP de type processus (*PEP 10 -- Voting Guidelines*) dont voici un extrait :

> This PEP outlines the python-dev voting guidelines. These guidelines serve to provide feedback or gauge the "wind direction" on a particular proposal, idea, or feature. They don't have a binding force. (PSF, PEP 10)

Le vote en ligne est utilisé principalement sur la liste *python-dev*. La communauté des *Python committers* et des *Python contributors*, les développeurs de Python, en sont les principaux usagers. Plus qu'un simple outil, le vote en ligne devient une activité, réservée à certains membres de la communauté et par laquelle ils s'y intègrent : voter c'est prendre part à la communauté. Son usage est requis lorsqu'une nouvelle idée émerge, qu'une nouvelle fonctionnalité est proposée ou encore qu'une PEP autour du langage est discutée. Il arrive à l'occasion que le BDFL souhaite procéder à une forme de sondage non officiel ; il se sert alors du vote en ligne. Enfin, le vote en ligne est également utilisé, comme on a pu le voir sur la liste *python-committers,* afin d'accorder le droit en écriture sur le dépôt de code. C'est donc un outil essentiel pour évaluer le sentiment général des individus de la communauté et grâce auquel tout un chacun peut s'exprimer.

Les votes sont généralement accompagnés d'une argumentation justificative et toutes les opinions, aussi différentes soient-elles, sont évaluées avec la plus grande attention. C'est particulièrement le cas des votes s'opposant à une idée, pour lesquels la justification doit être la plus instructive et précise possible quant aux raisons de cette objection.

Les principes du vote au sein de la communauté Python sont dérivés de ceux utilisés par la communauté Apache. Plusieurs choix de votes sont possibles, comme le stipule

la proposition PEP qui encadre ce mécanisme (PEP 10). Il y a quatre valeurs possibles au vote : +1 « J'apprécie la proposition, j'approuve », +0 : « Je m'en fiche, mais allez-y », -0 : « Je m'en fiche, alors pourquoi s'embêter » et enfin -1 : « Je déteste la proposition, je désapprouve ». Certains peuvent manifester leur grand enthousiasme par des valeurs inusitées telles que +2, +1000 ou -1000, mais au-delà de l'aspect ludique de la chose, ces votes n'ont pas plus de poids que +1 ou -1. L'extrait de la PEP nous explique ces valeurs selon le propre vocabulaire de la communauté :

> +1 I like it
>
> +0 I don't care, but go ahead
>
> -0 I don't care, so why bother?
>
> -1 I hate it
>
> You may occasionally see wild flashes of enthusiasm (either for or against) with vote scores like +2, +1000, or -1000. These aren't really valued much beyond the above scores, but it's nice to see people get excited about such geeky stuff.

Le vote en ligne peut être utilisé par un individu simplement pour affirmer sa position sur un sujet divers, dans le cadre de débats et de son argumentation. Il n'est pas encadré par un processus englobant de prise de décision de manière formelle et officielle. Le vote en ligne dans la communauté Python reste un outil assez informel et non rigide.

Bien que le vote ligne au sein de la communauté Python soit fortement inspiré d'un processus semblable au sein de la communauté Apache et ne constitue donc pas une innovation propre à la communauté Python, il n'en demeure pas moins que ce type d'outil de prise de décision communautaire, issue de la culture du libre, représente une innovation organisationnelle dans le cadre du développement logiciel. Par ailleurs, ce type d'outil et d'activité de prise de décision est essentiel dans le mécanisme d'innovation par l'utilisateur comme nous allons le voir dans la section suivante.

5.2 Les propositions d'amélioration de Python

Une « proposition d'amélioration de Python » ou *Python Enhancement Proposal* (PEP) est un processus écrit par lequel les évolutions du langage de programmation sont conçues et documentées. Ce mécanisme est majeur dans la communauté Python : il sert de support à l'innovation par les utilisateurs au sein de la communauté. Cela signifie qu'une proposition est un document de travail, de conception technique (*design*) et de documentation. Ces documents, placés dans le domaine public, sont à destination des membres de la communauté et sont revus collectivement. Ainsi, une PEP est un document qui suit un flux d'activité (*workflow*), peut présenter différents statuts et fait l'objet de traitements et d'amendements par la communauté.

5.2.1 Un processus formel

Ce mécanisme, par lequel on améliore le langage, est lui-même décrit par un document : *PEP Purpose and Guidelines*. La communauté a ainsi codifié la manière dont elle doit concevoir son activité de développement logiciel.

Ainsi, la proposition est un document textuel qui doit respecter un format prescrit : celui de la RFC 822 (*Standard for the format of ARPA Internet text messages*). Ce processus d'écriture du document débute toujours par une idée d'amélioration du langage. On recommande fortement de ne décrire dans une proposition qu'une seule idée nouvelle.

Les PEP peuvent viser des objectifs variés : outre les évolutions techniques du langage, on peut y décrire les procédures du fonctionnement interne de la communauté, par exemple la publication du code (*code release*) ou le choix du logiciel de gestion du code source. Les modifications mineures ou les *patches* n'ont

pas besoin de PEP et peuvent suivre le processus de développement de Python décrit précédemment (« Rapporter un *bug* dans le logiciel »).

Il existe trois types de PEP : la PEP Standard (*Standards Track*), la PEP Information (*Informational*) et la PEP Processus (*Process*). Une PEP dite Standard décrit une nouvelle fonctionnalité ou implémentation ajoutée à Python. Ce type de PEP permet d'anticiper et de préparer les nouvelles fonctionnalités mises en œuvre par la communauté. Une PEP Information décrit un problème technique de conception et fournit des informations générales à la communauté Python, mais ne propose pas de nouvelle fonctionnalité. Dans ce cas-ci, le document ne requiert par de consensus ou de recommandation, il s'agit plutôt d'une manière de documenter, à titre de référence, un choix technique ; la communauté emploie donc ce type de PEP pour gérer ses connaissances communes. Enfin, une PEP Processus décrit un processus au sein de la communauté. Ce type de PEP n'a pas de rapport direct avec le langage de programmation Python, mais concerne la communauté. Il s'agit, par exemple, de décrire des procédures, des changements dans le processus de décision ou dans le type d'outils utilisés par la communauté. Ce troisième type de PEP permet à la communauté de codifier son fonctionnement et ainsi de s'organiser.

Concrètement, pour rédiger une PEP on lui donne d'abord un titre. Ensuite, la PEP doit comporter quatre sections : un résumé (*abstract*), les motivations (*rationale*), les références bibliographiques (*abstract*) et un descriptif détaillé. Cette dernière section peut contenir des exemples de code, un résumé des discussions autour de l'évolution, les limites de la proposition, des cas d'utilisation. On ajoute enfin la liste des auteurs et les liens de références vers les discussions en ligne.

Finalisées, les PEP sont formées de deux parties : le document de conception (le texte) et l'implémentation de référence (le code). Ce sont ces deux parties qui sont discutées et ainsi conçues collectivement en ligne par la communauté Python, et ce sur les listes de discussion publiques.

5.2.2 Un processus collectif

La publication et la discussion en ligne d'une PEP permettent aux membres de la communauté de participer aux décisions de conception, de relever des problèmes et de collecter les opinions sur divers sujets, souvent techniques, liés à la proposition. Ceci correspond à la phase de revue par la communauté (*review*). Le processus devient ainsi collectif. L'auteur d'une PEP à la responsabilité, en plus de l'écriture de la spécification technique, d'obtenir les retours de la communauté (*feedback*), de construire un consensus avec la communauté et de documenter les opinions divergentes. Le dévoilement de la proposition dans un espace public engendre des débats, des confrontations et la résolution de controverses techniques soit par l'obtention de consensus, soit par décision d'une autorité procédurale conférée au BDFL, soit par l'abandon pur et simple d'idées non réalisables techniquement. Une pratique courante, afin de montrer la faisabilité d'une proposition, est de réaliser un prototype, c'est-à-dire de prouver sa proposition par du code informatique « qui fonctionne ».

Une PEP est validée, collectivement, en deux phases. La PEP est d'abord discutée par la communauté, dans une première phase appelée Pré-PEP sur la liste *python-ideas*, et les retours sont intégrés par l'auteur dans le document. Les échanges sur le fond peuvent ainsi continuer entre développeurs sur une base commune pré-validée et ayant obtenu un premier niveau de consensus. L'idée est jugée acceptable, utile pour la communauté et intéressante. C'est ainsi qu'une proposition est rendue officielle dans la communauté, qu'un numéro lui est attribué et que s'amorce la seconde phase : la discussion de la proposition sur la liste *python-dev*.

Plusieurs fils de discussion peuvent être lancés en parallèle au sujet d'une même proposition, c'est alors à son auteur, appelé le champion, d'organiser les débats et d'amender le document avec les retours fournis par la communauté. Ces allers-retours entre discussion et implémentation peuvent durer plusieurs jours. Le champion, à la

fois auteur — écrivant le document de conception de manière formelle, tel que demandé par le processus — et animateur mène les discussions sur les forums et construisant un consensus avec la communauté autour de son idée.

La décision finale d'acceptation de la proposition doit être demandée au BDFL, sur la liste *python-dev*, qui la révise. C'est lui qui fait office d'autorité finale dans ce processus. Sa décision est appelée un « *pronouncement* ». Une fois la proposition acceptée, l'implémentation de référence doit être terminée. Le code source doit être incorporé au code principal et déposé (*committed*) sur le dépôt.

Ce qui est intéressant dans cette procédure, et qui tranche avec le modèle du « bazar » décrit par Raymond (2000), est cette organisation collective, relativement hiérarchisée, fonctionnant par paliers et formalisée. La communauté Python, par son processus d'amélioration du langage, met en œuvre une innovation organisationnelle au sein du domaine du développement logiciel afin de faire participer ses utilisateurs.

5.2.3 Une innovation organisationnelle

Le processus d'amélioration du langage Python constitue une innovation dans le sens où il est original dans le monde de l'*open source*, codifié et spécifique à la communauté Python.

L'innovation par la communauté Python est soutenue par le processus d'amélioration du langage (PEP) et est centrée sur les utilisateurs. C'est ce processus qui, lors de la phase d'innovation, concentre les activités de réflexion, de révision et de correction autour de la proposition par la communauté. Il est initié par un auteur, ou groupe d'auteurs, puis mis en œuvre collectivement par un nombre relativement important de participants : les utilisateurs développeurs du langage. Nous identifions le rôle du champion, véritable *lead user*, qui exprime de nouveaux besoins en amont de la communauté. Par le processus d'amélioration du langage, il dévoile sa proposition à la communauté toute entière qui la valide et permet ainsi sa conception collective. La proposition d'amélioration du langage est le processus qui permet d'inclure les utilisateurs dans l'activité de développement. Ainsi organisés par le processus, les développeurs Python forment un réseau d'individus innovateurs. C'est que ce que Von Hippel nomme le « réseau horizontal d'innovation ». Ce réseau est constitué des membres de la communauté qui participent aux discussions sur l'évolution du langage. C'est en cela que la communauté Python innove : en mettant en œuvre un processus léger, mais formalisé, distribué géographiquement, par courrier électronique de gestion de l'innovation par les utilisateurs.

C'est précisément ce que nous allons montrer par la suite à l'aide de deux cas, en insistant sur la manière dont la gestion de l'innovation est prise en charge par les membres de la communauté grâce au processus d'amélioration de Python et le logiciel lui-même qui forme une boîte à outils mise à disposition des utilisateurs par la communauté.

5.3 Cas n° 1 : PEP 435 – Ajouter un type énumération au langage

Nous présentons dans cette section un cas de proposition d'amélioration du langage dont la réalisation est allée à son terme dans la version actuelle du logiciel Python. Cela signifie que la proposition fut dûment proposée, discutée, validée, implémentée et incorporée dans le langage en respectant le processus formel décrit précédemment. On trouve dans cette proposition toutes les étapes qui l'ont fait passer, d'une simple idée formulée par un utilisateur-innovateur, à un document de spécification et à une implémentation technique écrits, revus et validés collectivement par les utilisateurs puis ajoutés au logiciel. Par l'étude d'un cas concret, nous souhaitons ainsi montrer les choix d'implémentation, les décisions qui ont été mises en œuvre par les membres de la communauté ainsi que les enjeux relevés afin de mener à bien ce projet d'innovation collective. C'est tout l'intérêt de ce cas que de montrer la manière dont l'innovation est gérée collectivement par les utilisateurs de Python organisés en réseau d'utilisateurs-innovants.

La proposition d'amélioration du langage que nous analysons porte sur l'ajout d'un type énumération à la bibliothèque standard, c'est-à-dire un nouveau type de données à l'ensemble des fonctionnalités du langage de programmation Python. Pour illustrer le mécanisme d'innovation, nous étudions les points qui font particulièrement débats lors de la phase de discussion en ligne de la proposition. Ces points de discussions sont éclairants, non pas du fait de leur caractère technique intrinsèque, mais plutôt par le caractère innovant et collectif du mécanisme d'exposition des enjeux et de prise de décisions inhérents au processus. En effet, outre le respect des étapes procédurales du processus, ces points de discussion, par messages électroniques interposés, donnent corps au mécanisme d'innovation par les utilisateurs qui se déroule en ligne sur les listes de discussion publiques de la communauté Python : ils en sont le cœur du processus.

5.3.1 Définition d'une énumération

Dans le domaine de la programmation informatique, une énumération, ou type énuméré, est le nom que l'on donne à un certain type de données. Ces données consistent en un ensemble de noms symboliques (les membres), à la fois liés ensemble, uniques et de valeurs constantes. Ce type de données du langage est utilisé pour représenter des données qui sont connexes les unes par rapport aux autres. Lorsque l'on crée un type énuméré, on définit ainsi une énumération, c'est-à-dire une liste de valeurs liées ensemble. Une énumération « Couleur des cartes » sera, par exemple, constituée des membres de valeur constante « Carreau », « Cœur », « Pique » et « Trèfle ».

Concrètement, en langage de programmation Ada, un autre langage de programmation pris à titre d'exemple, l'énumération précédente est écrite de la manière suivante :

```
type CouleurCartes is (carreau, coeur, pique, trefle);
```

En Python, cette fonctionnalité n'existait pas avant la proposition d'amélioration que nous allons étudier. Nous cherchons donc à comprendre l'innovation à l'œuvre qui va mener à la mise en place d'une telle fonctionnalité.

5.3.2 Fils de discussion étudiés

La proposition PEP, c'est-à-dire le document de spécification, est présentée sur les listes de discussion publiques de la communauté (*python-ideas* et *python-dev*). Deux fils de discussion ont servi de support aux activités du processus d'amélioration du langage. Le tableau 5.1 présente leur titre, le nombre de messages échangés, le nombre de participants et la durée de la conversation.

Tableau 5.1 : Fils de discussion étudiés – PEP 435

Liste	Titre du fil	Mess.	Particip.	Durée
python-ideas	PEP for enum	117	32	12j
python-dev	PEP 435 - Adding an Enum type	227	30	17j

Nous constatons que ce processus fait intervenir un nombre important de participants, une trentaine (autant sur la liste *python-ideas* que *python-dev*), sur une période relativement courte, inférieure à un mois. Cela montre que l'évolution proposée a largement été discutée par la communauté. Ce cas correspond à l'une des plus longues discussions concernant une évolution.

5.3.3 Analyse du contenu des échanges du fil « PEP for enum »

Le fil de messages « PEP for enum » se déroule sur la liste de discussion *python-ideas* – une liste qui regroupe les différentes propositions d'idées pour le langage faites par les utilisateurs – sur une période de douze jours, soit du mardi 12 février 2013 au samedi 23 février 2013 inclus. La conversation est initiée par Elias, qui explique les enjeux de la discussion, à savoir : échanger au sujet de l'idée d'un nouveau type énumération dans le langage et statuer si un document PEP est nécessaire pour mener à bien cette évolution. Le fil débute par le message suivant :

> Hi all,
>
> The ideas thrown around and Tom's prototype in the thread "constant/enum type in stdlib" show that it's possible to use some metaclass magic to implement very convenient syntax for enums, while only including them in the stdlib (no language changes):
>
> [...]I think it may be worthwhile to start from the other direction by writing a PEP that aims to include this in 3.4. The way I see it, the PEP should discuss and help us settle upon a minimal set of features deemed important in an enum.
>
> If that sounds OK to people, then someone should write that

```
PEP :-) This could be Tom, or Harry who's been maintaining
flufl.enum for a long time.
If no one steps up, I will gladly do it.

Thoughts?

Elias
```

Ce premier message est intéressant, car il montre la manière dont l'idée a émergé, en l'occurrence lors d'une discussion sur un sujet connexe sur une autre liste de discussion. Nul besoin d'un long discours, on fait simplement état de l'idée brute, du point de vue technique qui doit être débattu. On observe ici que la question posée à la communauté est de savoir si elle juge nécessaire l'écriture d'un document PEP. On assiste ainsi à l'exposition d'une idée à l'ensemble des membres de la communauté : c'est le début du processus d'innovation par les utilisateurs. Proposer de nouvelles idées correspond précisément à la finalité de la liste de discussion *python-ideas*. On note que les auteurs d'une éventuelle formalisation de la proposition PEP ne sont pas tout de suite identifiés. Ils sont nommés, mais choisis sur la base du volontariat et Elias se propose pour le faire.

L'idée est simple et pourtant, comme nous le montrent les échanges qui suivent sa réalisation et ses enjeux sont complexes. Pour plus de clarté, nous découpons le fil de discussion en plusieurs sujets thématiques selon l'ordre chronologique des échanges.

Le choix d'une syntaxe

L'objet du premier thème de discussion est de trancher entre deux implémentations majeures. Une implémentation, dans le domaine de la programmation informatique, est l'ensemble du code source d'un programme. Ces deux propositions syntaxiques concurrentes sont soutenues par deux utilisateurs distincts : Tom contre Harry.

Nous observons que sur la liste orientée proposition d'idées (*python-ideas*), il existe déjà deux versions du code informatique qui sont débattues, alors qu'aucun document de spécification n'existe encore. Le document issu du processus, dans ce cas, va

officialiser une proposition d'amélioration de Python et, comme allons le voir, soutenir les débats à ce sujet.

La version de Tom a l'avantage d'être claire et concise, mais nécessite une certaine dose de « magie », c'est-à-dire une forme d'automatisme et une syntaxe de la fonctionnalité rendue peu explicite. Elle comporte de surcroît un *bug*, mais celui-ci sera corrigé lors de l'échange. La syntaxe de l'énumération suivant cette implémentation est :

```
class Color(Enum):
  RED, BLUE, GREEN
```

De son côté, la version de Harry a l'avantage d'être explicite, générique et ne nécessite pas de « magie » dans son code. Les échanges laissent à penser que cette implémentation n'est pas nouvelle, qu'elle est éprouvée et robuste. La syntaxe de l'énumération suivant cette implémentation est :

```
class Color(Enum):
  RED = 1
  BLUE = 2
  GREEN = 3
```

Lors de ces premiers échanges, nous remarquons tout de suite que c'est du code informatique qui est échangé par courriel et que ce code soutient toute la discussion technique au sujet de l'idée d'amélioration. Nous observons que sur cette liste de discussion, un pragmatisme ambiant « force » les utilisateurs à échanger sur la base d'un code informatique déjà écrit ou qui serait fonctionnel. Il ne s'agit pas ici de discuter de la validité théorique de la proposition, mais bien de sa faisabilité technique.

Nous remarquons aussi que, contrairement à ce que l'on pourrait penser, les échanges ne se déroulent pas en faveur de l'une ou l'autre des propositions de Tom ou de Harry. Chacun des participants fait une contre-proposition spécifique, différente de celles qui sont débattues ; seuls quelques intervenants seulement soutiennent ou désavouent directement les propositions majeures. En effet, à ce stade-ci des

discussions, tous les participants ont un niveau d'autorité semblable et se sentent à l'aise de faire valoir leur propre code, sur la base de leur expertise commune en développement logiciel. Ainsi, les contre-propositions de syntaxe sont variées, on compte dans le fil de discussion jusqu'à dix-huit variations de syntaxe proposées. On assiste ainsi à une véritable controverse technique au sujet de la syntaxe qui devra être choisie. Les idées sont lancées les unes à la suite des autres et chacune des propositions obtient son lot de partisans.

Très rapidement, durant ce premier temps de la conversation, on voit apparaître un consensus au sujet de la question posée à la communauté : une PEP devient une nécessité compte tenu de la complexité du sujet et de la diversité des propositions. Ainsi, Ethan (*core developer*) indique :

> A PEP would be excellent. I tried searching for one a couple days ago to help my understanding with Tom's methods, but sadly didn't find one. :(
>
> -Ethan-

Cette citation nous montre que l'entrée ou non dans le processus PEP est défini par les utilisateurs eux-mêmes, cela ne vient pas de l'autorité procédurale. La décision de se coordonner autour d'un document de spécification émerge comme une nécessité et se rapporte à une routine organisationnelle forte, puisqu'ancrée dans l'habitude et demandée par les utilisateurs.

Notons que Guido se prononce à plusieurs reprises, mais simplement contre l'implémentation de Tom : il n'approuve ni sa syntaxe ni son caractère « magique ».

Dans les échanges qui suivent, Harry et Tom défendent chacun leur implémentation. Cependant, c'est au terme de nombreux échanges que la controverse autour de la syntaxe est finalement close par Guido lui-même : il choisit l'implémentation de Harry sur la base de sa syntaxe claire et explicite, en dépit de l'obligation de définir l'énumération que certains trouvaient problématique.

> Frankly, enums are not that useful in small programs. For

> large programs or libraries, and especially for public APIs, the extra cost of defining the enum shouldn't count against them.
>
> Let's just import Harry's enums into the stdlib.
>
> --Guido van Rossum

Ici, l'on voit très bien le rôle d'autorité finale qu'exerce Guido dans le processus d'innovation. Cela ne clôt pourtant pas le fil de discussion, mais le BDFL n'interviendra plus et la syntaxe est ainsi « validée ».

Notons que deux autres consensus mineurs sont apparus lors de cette conversation : la nécessité d'avoir de bonnes « représentations », c'est-à-dire de bonnes fonctions d'affichage d'une énumération, et celle d'utiliser le mot clé « Enum » dans la syntaxe retenue.

Ce premier thème des échanges est instructif sur plusieurs points : (1) l'introduction dans le processus d'innovation formelle à partir d'une idée émergente est décidé par les membres de la communauté ; (2) la liste de discussion sert de plateforme d'échange et de soumissions de potentielles idées d'amélioration du langage ; (3) ces idées sont librement échangées sur une base technique connue et maitrisée par tous et exprimée en langage Python ; (4) les choix de la communauté sont faits sur la base de consensus ; (5) l'autorité procédurale exercée par Guido est forte et il tranche les choix d'implémentations majeurs.

Une décision et une fonctionnalité débattues

A la suite de ce premier thème de discussion, le second thème émerge avec un message d'Elias, l'initiateur du fil, approuvant la décision de Guido.

La discussion porte principalement sur cette décision qui est tantôt approuvée par la communauté, tantôt désavouée par certains membres. Nous observons que les développeurs ont l'opportunité d'exprimer leur désapprobation et que cela n'entache aucunement la valeur de leur contribution. C'est le cas de Tom, dont

l'implémentation n'a pas été retenue et qui constate que cette décision ne satisfera pas ses besoins :

> *Let's just import Harry's enums into the stdlib.*
>
> That's entirely your call. FWIW I probably won't use them, as they fail to meet my needs for an enum, #1 being not having to specify their values in any way if I don't want to. Once you get past 3 or 4 values, it's too easy to miss or reuse a value.
>
> Tom

Pourtant, Tom restera un interlocuteur privilégié lors des échanges suivants, intervenant souvent auprès de Harry. Il devient ainsi un spécialiste de l'énumération puisqu'il a écrit sa propre version d'une implémentation de la fonction.

Au fil des échanges, l'implémentation de Harry, choisie depuis l'intervention de Guido, est scrutée par l'ensemble des développeurs. Une nouvelle question émerge alors au sujet d'une fonctionnalité issue de celle-ci : la fonction *make()*. Cette fonction permet à l'utilisateur de Python de créer une énumération sous la forme d'une liste de membres : ces derniers étant écrits les uns à la suite des autres plutôt que sous la forme classique indiquée à la section précédente. Voici un exemple de cette fonction :

```
Color = Enum.make('Color', 'RED BLUE GREEN')
```

Trois interrogations apparaissent au fil de la discussion : cette fonction serait-elle adéquate ? Est-ce une fonction *utile* dans le cadre de l'implémentation de l'énumération ? Et enfin, le débat n'étant plus seulement utilitaire, mais devenant normatif : les normes de développement suivies nous autorisent-elles à inclure une telle fonctionnalité ? En effet, cette fonction offre à l'utilisateur une seconde manière de déclarer une énumération, c'est-à-dire une seconde façon d'écrire syntaxiquement le code. Or, bien que la norme n'encadre par spécifiquement la manière dont une énumération devrait être déclarée, une « règle » encadre le nombre de façon de faire que l'on propose à l'utilisateur.

Cette « règle » est issue du *Zen de Python* : « *There should be one-- and preferably only one --obvious way to do it.* ». Ainsi, l'existence de cette fonction produirait un code qui dérogerait de cette « règle ». La fonction *make()* ne serait pas une approche « *Pythonic* » c'est-à-dire ne respectant pas le *Zen de Python*.

Ici, les développeurs se questionnent quant à la singularité d'une telle approche dans leur activité même de conception et sont réflexifs sur leur propre pratique. Doit-on ou non donner à l'utilisateur de Python différentes manières de déclarer ce nouveau type de données ? Mickeal (*core developer*), par exemple, argumente adroitement en faveur d'une approche moins rigide de cette « règle » sujette à interprétation :

```
Stephen wrote:
> Besides, the presence of a second, non-obvious
> solution is not a violation of One Obvious Way.

Something that is often forgotten is that having two ways
to do something is often *good* for your API design,
because it lets you design one simple API that covers a
large fraction of use cases, and then a more complex
underlying API that covers all (or almost all) of the rest.
--
Mickeal
```

Les messages s'enchainent pour déterminer d'une part, si cette fonction est explicite, flexible et utile pour l'utilisateur et d'autre part, la manière d'interpréter la « règle ». Les échanges de ce second objet de discussion se concluent alors que cette question reste en suspend et aucune décision n'est prise à ce stade du développement du type énumération.

Le thème relaté ici est important dans le cadre de ce cas de processus d'innovation, car il montre plusieurs éléments : (1) bien que Guido soit l'autorité procédurale, ses choix peuvent être débattus librement sur la liste de discussion ; (2) un membre de la communauté gagne en notoriété sur la base de ses propositions et de son implication dans les débats ; (3) même si sa solution n'est pas retenue, cet utilisateur-innovateur reste un intervenant important ; (4) les règles du *Zen de Python* influent directement sur la manière dont le code est produit, revu et validé ; (5) les règles du *Zen de*

Python sont sujettes, sans cesse, à interprétation et replacées dans le contexte du développement du logiciel Python idéal.

Le résumé des échanges

Troisième sujet dans le cadre de ces échanges : neuf jours plus tard, Alex résume les discussions et souhaite apporter des éclaircissements sur la finalité de l'énumération.

```
Hi all,

Sorry to jump into this discussion so late, but I just
finished reading through this thread and had a few
thoughts...
[...]
It seems to me that this particular thread started out as a
call to step away from existing implementations and take a
look at this issue from the direction of "what do we
want/need" instead, but then it quickly got sidetracked
back into discussing all the details of various
existing/proposed implementations. I'd like to try to take
a step back (again) for a minute and raise the question:
What do we actually want to get out of this whole endeavor?
```

Cependant, cette tentative n'aura que peu de succès et ne relancera pas les débats. Ceci est d'abord dû au temps écoulé depuis les vifs échanges et ensuite, au fait que l'auteur ne respecte pas l'une des normes de la communauté : un message se doit d'être court, explicite et le plus souvent soutenu par du code source explicatif. L'appel à cette norme prend la forme : « TL ; DR » pour « *Too Long ; Didn't Read* ». Devançant cette critique, l'auteur du message récapitulatif s'en excuse en amont.

Cela montre que le processus d'innovation est aussi un processus inscrit dans le temps et relativement normé. Il ne s'agit pas de discuter illusoirement et hypothétiquement de la validité d'une proposition alors que les vifs échanges sont terminés, mais de faire tester sa proposition d'innovation par la communauté, l'éprouver techniquement et la faire réviser par tous les utilisateurs disponibles, et ce, relativement rapidement.

Nous avons tenté d'établir dans cette première phase du processus d'innovation certains faits qui lui semblent propres : le rôle prédominant des utilisateurs dans les débats, le rôle de Guido prenant acte de décisions, l'établissement de consensus au sein de la communauté. Nous aurons l'occasion de poursuivre le développement de notre analyse lors de la seconde phase de revue de la proposition, celle sur la liste de discussion *python-dev* à destination plus largement des développeurs du langage.

5.3.4 Analyse du contenu des échanges du fil « PEP 435 - adding an Enum type »

Le fil « *PEP 435 - adding an Enum type* » est le second fil de discussion ayant pour sujet l'Enumération. C'est la suite logique du processus d'innovation prenant part sur une seconde liste de discussion. Il se déroule cette fois-ci sur une période de dix sept jours, soit du vendredi 12 avril 2013 au dimanche 28 avril 2013 inclus. Comme indiqué dans la description du processus d'amélioration, la proposition PEP est désormais concrètement débattue sur la liste dédiée au développement (*python-dev*). Une proposition d'amélioration PEP a été écrite formellement, un titre lui a été donné et elle possède désormais un numéro : PEP 435.

La conversation est initiée par Elias et Harry qui expliquent les enjeux de la discussion, à savoir : faire la révision de la proposition PEP présentée en pièce jointe à la communauté des développeurs de Python.

```
Hello python-dev,

We're happy to present the revised PEP 435, collecting
valuable feedback from python-ideas discussions [...]  We
believe the proposal is now better than the original one,
providing both a wider set of features and more convenient
ways to use those features.
[...]
Comments welcome,

Harry and Elias
```

Ce premier message d'ouverture du fil de discussion est intéressant, car il établit clairement les enjeux du fil, à savoir : cadrer les débats en ce qui a trait à la PEP 435. On apprend que les premiers retours, issus des discussions sur la liste précédente *python-ideas,* ont été intégrés pour créer le document. De nombreux thèmes de discussions vont émerger de la conversation. Comme pour le fil précédent, nous avons découpé celui-ci en plusieurs thèmes selon l'ordre chronologique des échanges, à des fins de clarté.

Le type d'exception

L'objet du premier thème de discussion est d'évaluer la pertinence du type d'exception. L'exception, dans le domaine de la programmation informatique, est le mécanisme permettant de gérer, en amont, les erreurs d'un programme. Tout développeur doit s'assurer que les erreurs susceptibles de se produire lors de l'exécution d'un programme soient correctement prises en compte.
Lorsqu'un programme rencontre une erreur, on dit qu'il y a « levée » d'une exception. L'exemple classique est celui de la division par zéro, division qui n'a pas de sens. Lorsque l'on souhaite exécuter une division à l'aide d'un programme, il doit y avoir un mécanisme d'exception sous-jacent afin de s'assurer que la division par zéro n'est pas possible. Dans ce cas précis, en Python, l'exception du type `ZeroDivisionError` est prise en compte. Chaque cas d'erreur possible doit ainsi être documenté et pris en compte par le développeur.

Dans le cas qui nous intéresse, la comparaison entre deux membres d'une même énumération semble également dénuée de sens.
Par exemple, dans l'énumération Fruits :

```
class Fruits(Enum):
  BANANE = 1
  ORANGE = "2"
  KIWI = 3
```

La comparaison entre `BANANE` et `ORANGE` doit échouer, car la classe Fruits n'est pas ordonnée. En d'autres termes, cela signifie qu'elle ne prend pas en charge l'ordonnancement de ses membres. Cette comparaison doit donc lever une exception.

Ainsi, à la lecture de la proposition publiée sur la liste de discussion des développeurs, Dirkjan est le premier à réagir, et ce très rapidement. Il apprécie la PEP, mais trouve que l'exception `NotImplementedError`, levée dans le cas précis de cette comparaison, est étrange et inadéquate. Il propose d'utiliser à la place une exception nommée `TypeError`.

Nous observons que cette remarque ne fait pas immédiatement consensus et est sujette à plusieurs échanges. La pertinence de l'exception levée fait débat et une discussion technique s'en suit. Guido intervient et appuie la recommandation de Dirkjan. Cela mènera à une modification à la fois de l'implémentation, c'est-à-dire du code, de la proposition PEP et de la documentation associée par Harry, coauteur de la PEP.

La remarque soulevée par Dirkjan est intéressante, car on se rend compte du mode de fonctionnement sous-jacent au mécanisme de révision par les pairs d'une proposition d'amélioration. Plusieurs éléments peuvent être soulevés ici.

D'abord, puisque la proposition d'amélioration est rendue publique, chaque choix d'implémentation effectué par l'auteur d'une PEP peut faire l'objet d'une remarque par l'un des développeurs abonné à la liste de discussion *python-dev*. Notons que le fait que le code informatique soit scruté par un nombre important d'individus développeurs est emblématique du monde de *l'open source*. C'est la loi de Linus (*Linus's Law*) rédigée par Eric Raymond « given enough eyeballs, all bugs are shallow » (Raymond, 2000). Ce qu'il y a de particulier ici est que c'est la proposition d'amélioration qui est scrutée et non pas le code informatique directement. La

proposition contient à la fois la spécification de l'implémentation et des exemples de code, de même que l'implémentation en elle-même. Cela signifie que le texte descriptif de l'évolution et le code démontrant son fonctionnement sont tout autant évalués que le code lui-même. En d'autres termes, l'évaluation de l'outil informatique écrit en Python et en même temps celle du « manuel » décrivant son fonctionnement, représenté par le document de PEP, sont effectuées par l'ensemble des développeurs évaluateurs.

Le second point intéressant dans le cadre de ce mode de fonctionnement sous-jacent est le rôle exercé par Guido sur un sujet très technique. Guido, en effet, agit ici comme un développeur expert du développement du logiciel Python. Le BDFL, outre sa position de décideur technique final en ce qui concerne la proposition d'amélioration du langage d'un point de vue global, se place comme un décideur à un niveau « local » de détail très fin, presque à la place de l'auteur, pour évaluer son choix technique.

Enfin, troisième et dernier point soulevé par ce cas de figure : il revient à l'auteur de la proposition PEP de prendre acte des suggestions, remarques et décisions prises dans la cadre de la phase de révision de la proposition d'amélioration du langage, et ce, dans deux espaces d'activité distincts. L'un est celui de l'espace d'*implémentation,* représenté par le document de PEP et son implémentation ; l'autre est l'espace de *documentation,* représenté par la documentation officielle de l'amélioration développée. Cette modification se fait conjointement dans les deux espaces d'activité à partir du troisième : l'espace de *discussions*. Ainsi, les trois espaces d'activités sont interreliés par le processus d'amélioration du langage et c'est ce qui le rend orignal, propre à Python et spécifique au monde de l'*open source*.

L'ordonnancement des énumérations

Le second thème des échanges débute par un commentaire de Guido indiquant qu'il apprécie la PEP, mais qu'il aurait préféré voir les énumérations ordonnées, dans la mesure où les valeurs sous-jacentes le sont. Cela relance le débat autour de l'ordonnancement des énumérations entre elles et le fait d'inclure ou pas une fonctionnalité de comparaison.

Plusieurs points sont débattus : d'abord, cela fait-il sens de parler de comparaisons entre énumérations dont les types sous-jacents sont différents ? La question est rapidement close : non, cela ne fait pas sens. Pour Guido, la comparaison devrait échouer dans ce cas particulier et réussir seulement dans le cas où la classe des membres est identique.

Ensuite, dans les cas des classes énumération dérivées de type entier (*IntEnum*), c'est-à-dire des énumérations dont les valeurs des membres sont des entiers numériques, doit-on ou non autoriser la comparaison ? La réponse est oui, suivant un consensus entre Harry et Tom.

Enfin, la comparaison des membres de même type d'une énumération doit-elle être en fonction de l'ordre de définition des membres, des noms des membres ou de la valeur des membres ? La réponse n'est pas simple et on parvient difficilement à un consensus, notamment parce que Guido n'est pas convaincu par un ordre précis. Finalement, une décision sera prise et ce sera l'ordre de définition qui sera choisi. La raison en est simple : c'est l'ordre intuitif qui est le meilleur. Encore une fois, ici, la communauté fait preuve de pragmatisme et se réfère implicitement au *Zen de Python* pour prendre une décision d'implémentation ardue.

Une nouvelle fois, ce cas de figure différent montre l'importance du mécanisme de revue de la proposition d'amélioration du langage. D'abord, il importe de démontrer la position prise par Guido dans le cadre des choix d'implémentation effectués par l'auteur du document de PEP. Il s'agit ici tantôt de démontrer la légitimité d'une

posture d'ordre technique, tantôt de faire valoir une décision auprès d'un autre expert. Guido intervient comme un guide que l'on consulte et qui va aiguiller le développement de la proposition.

Le second point que l'on observe est le suivant : les choix d'implémentation ne sont pas aisés et il est difficile de répondre à de multiples questions techniques lorsque l'on est seul. En effet, on ne peut d'une part pas toutes les soulever et d'autre part, on ne sait pas répondre à toutes. La revue de la proposition d'amélioration du langage sur la liste de discussion des développeurs agit comme un mécanisme de consultation des développeurs, de révélateur des failles de la proposition et, simultanément, de vecteur de résolutions des anomalies rencontrées. Le prototype de l'évolution est ainsi développé de manière collaborative, ou co-développé, par l'ensemble des membres de la communauté évaluateurs de la proposition.

Ainsi par l'étude de ce premier cas d'innovation par les utilisateurs au sein de la communauté Python, à travers le processus d'amélioration du langage, il nous apparaît que : (1) la proposition et le code informatique sont étroitement développés ensemble et en collaboration avec les autres utilisateurs ; (2) le rôle d'utilisateur-évaluateur émerge de par l'activité même d'évaluation et de revue de la proposition d'amélioration du langage en ligne ; (3) le code développé par les utilisateurs, et intégré dans la proposition par l'auteur, est réflexivement évalué sur la base des règles du *Zen de Python*, ce qui normalise l'activité de développement ; (4) Guido intervient sur deux plans, à la fois global et où il tient le rôle de décideur final de la proposition et local, où il joue le rôle d'expert guidant le développement grâce à sa connaissance pointue du processus de développement et du langage de programmation Python.

5.4 Cas n° 2 : PEP 450 – Ajouter un module de statistiques au langage

La seconde PEP que nous souhaitons analyser a pour objet l'ajout d'un module de statistiques à la bibliothèque standard. En d'autres termes, il s'agit d'incorporer un ensemble de fonctionnalités liées aux statistiques et qui seront directement disponibles dans le langage, à la manière d'une boîte à outils. En effet, les fonctions de statistiques telles que la moyenne, la médiane, la variance ou la déviation standard n'étaient pas présentes dans Python ; c'est par le processus d'amélioration du langage qu'elles seront discutées en ligne, puis intégrées. Cette proposition est intéressante, car elle se déroule dans un contexte ou Guido intervient peu ; le rôle de leader technique étant porté par l'auteur de la proposition, le champion. Nous présentons ici différents cas de réflexions collaboratives pour montrer l'organisation collective du processus d'amélioration du langage.

5.4.1 Fils de discussion étudiés

Tableau 5.2 : Fils de discussion étudiés – PEP 450

Liste	Titre	Mess.	Particip.	Durée
python-ideas	Pre-PEP – statistics module	117	23	14j
python-dev	PEP 450 – statistics module	73	21	3j

Comme le précise le tableau 5.2, le processus d'ajout du module de statistiques s'effectue à la fois sur les listes *python-ideas* et *python-dev*. On y trouve respectivement la pré-PEP et la PEP, la première étant discutée sur des périodes de quatorze jours et cent dix-sept messages sont échangés, la seconde ne durant que trois jours durant lesquels soixante-treize messages sont publiés. Il nous apparaît alors que la phase de pré-PEP est celle qui fait l'objet du plus de discussion. Une fois

l'orientation définie les choses s'accélèrent dans la phase de PEP. De même, on constate que ce processus fait intervenir plus d'une vingtaine d'utilisateurs différents.

5.4.2 Analyse du contenu des échanges du fil « Pre-PEP – statistics module »

Analysons maintenant ces échanges en détail : le premier fil « *Pre-PEP - adding a statistics module* » se déroule sur la liste de discussion *python-ideas*, orientée proposition et sur une période de huit jours, soit du vendredi 2 août 2013 au vendredi 16 août 2013. On observe le même modèle de fonctionnement que pour la PEP précédente : la discussion prend d'abord forme sur la liste orientée proposition d'idées. La conversation est initiée par Stephen, qui en explique les enjeux. Il faut valider la PEP, alors à l'état de brouillon, sur l'ajout d'un nouveau module de statistiques dans le langage. Ainsi Stephen écrit :

> I have raised an issue on the tracker to add a statistics module to Python's standard library:
>
> http://bugs.python.org/issue18606
>
> and have been asked to write a PEP. Attached is my draft PEP. Feedback is requested, thanks in advance.
>
> --
>
> Stephen

Respect du processus d'inclusion

Dans le cadre des échanges qui suivent, la première problématique qui se pose est la suivante : respecte-t-on le processus standard pour l'inclusion d'un nouveau module dans la bibliothèque Python ? En effet, un nouveau module, c'est-à-dire un nouvel ensemble de fonctionnalités, est souvent d'abord proposé comme bibliothèque tierce, c'est-à-dire à l'extérieur de Python. Testé et éprouvé, il est après un certain temps

intégré à la bibliothèque standard. Plusieurs points de vue s'affrontent et c'est finalement la position de Stephen, l'auteur de la PEP, appuyée par Ethan, qui s'impose : le module a déjà été proposé comme une bibliothèque tierce pendant plus de trente mois et cela justifie la maturité du code que l'on souhaite maintenant inclure dans Python.

Ce point est intéressant, car ce sont les routines organisationnelles de la communauté Python qui sont ici testées. On se demande si l'on suit correctement ce qui est décrit dans l'une des PEP Processus rédigée par la communauté Python. Autrement dit, ce processus particulier de collaboration et de gestion du code est réévalué à chaque nouvelle occasion. Ce faisant, les membres de la communauté adoptent une nouvelle fois une attitude réflexive quant à leur manière de travailler et de mettre en œuvre des changements dans le logiciel Python.

Par ailleurs, il importe de déterminer si la fonction *sum()* (somme) est appropriée dans un module de statistiques ou alors si cela aurait plus de sens d'utiliser la fonction somme du module de mathématique (*math*). Les enjeux techniques sous-jacents doivent être en adéquation avec le *Zen de Python* et les « règles » de développement du langage. Le dilemme que présente ce choix de conception n'est pourtant pas simple à résoudre. Il faut soit autoriser une fonction similaire à deux endroits dans la bibliothèque standard, soit modifier la fonction somme du module de mathématique pour qu'elle puisse convenir aux besoins du module de statistiques. D'un point de vue technique, les deux options se valent. Ce qui nous intéresse, c'est la mise en œuvre du processus de décision collectif. Ainsi, il nous paraît que la résolution de ce dilemme est innovante. En effet, pour statuer sur la question : Stephen propose un vote auprès des développeurs. Avec une participation de quatorze développeurs, le résultat du vote est le suivant : l'utilisation d'une fonction somme dédiée au module de statistiques l'emporte. L'implémentation du module de statistiques ne sera donc pas modifiée.

Ce point est intéressant, car il s'agit d'étudier le mode de fonctionnement de la prise de décision par le mécanisme de vote. En effet, l'auteur de la PEP convie les développeurs à se prononcer sur le point technique qui fait débat, et ce, afin d'obtenir un consensus légitime.

5.4.3 Analyse du contenu des échanges du fil « PEP 450 – statistics module »

Nous observons que lors de la discussion de la PEP 450, différents aspects sont discutés. Les développeurs se demandent si le code présenté est « *Pythonic* », c'est-à-dire s'il suit les conventions et les « règles » du *Zen de Python*. Il s'agit d'un élément déjà couvert par notre analyse. Les développements effectués doivent respecter cette pratique et ses normes. D'autre part, il importe pour les membres de la communauté de s'assurer que les fonctions soient correctement nommées, compte tenu de la complexité du programme. En effet, la cohérence du programme informatique produit est de la plus haute importance pour ses développeurs. Toutefois, ces questions ne remettent pas en cause l'utilité du module et il est généralement bien accepté par la communauté.

Nous observons que dans le cas de cette proposition, les interventions de Guido ne sont pas nombreuses. Toutefois, bien que sa présence se fasse peu sentir par des messages, il apparaît clairement qu'il supervise les discussions. En effet, il agit principalement comme coordinateur de l'effort de développement. Par exemple, lors de discussions peu constructives, on peut lire dans les échanges l'intervention suivante de Guido :

```
This argument is getting tedious. Instead of arguing who
said or meant what when, get the code working.
```

Nous observons également, comme dans le cas précédent que Guido agit comme décideur final de l'acceptation de la proposition, il écrit :

```
Congrats, I've accepted the PEP. Nice work! Please work
with the reviewers on the issue on the code.
```

Par l'analyse de ces échanges, nous constatons que les rôles et responsabilités des utilisateurs, dans le cadre du processus d'amélioration du langage, sont variés et différenciés suivant le niveau d'autorité détenu au sein de la communauté.

Différents éléments de discours sont ainsi élaborés par les développeurs lors de la phase de discussion des propositions d'amélioration du langage. Ces éléments de discours peuvent être regroupés par type d'activités langagières. Nous nous proposons de les analyser dans la section qui suit.

5.5 Analyse des activités lors du processus PEP

L'analyse du discours des intervenants, dans le cadre du processus PEP qui se déroule en ligne sur les listes de discussion *python-ideas* et *python-dev*, nous mène à présenter les activités du discours ci-après afin d'expliciter la complexité du processus.

5.5.1 Présenter les enjeux

Trois activités conjointes se déroulent lors du premier message d'ouverture du fil de discussion : annoncer le sujet, décrire le processus dans lequel ce message s'inscrit et enfin, demander des commentaires ou des réactions à l'idée ou à la PEP proposée.
Ce message d'ouverture est primordial : c'est celui qui va amener la communauté à réagir sur la proposition d'amélioration de Python. C'est également celui qui cadre les débats : il s'agit de discuter ensemble d'un même sujet. Pour ce faire, on invite les membres de la communauté à participer aux débats, et ce, dans une ambiance où chacun peut s'exprimer sur une base partagée, c'est-à-dire sur la base du document de

spécification de l'évolution proposée. Ce message est envoyé par le champion de la PEP à la communauté des développeurs de Python.

5.5.2 Participer aux discussions

Plusieurs formes de discours permettent de participer aux débats. L'objectif est ici d'alimenter la discussion dans un cadre collectif, mais toutefois dans un mode plutôt passif.

D'abord, la plus simple forme de participation est de prendre connaissance de la PEP et d'apporter son soutien par un message bienveillant envers son auteur et quant à la qualité du travail fourni. Ensuite, et c'est le cas le plus fréquent : il est possible de donner son avis, souvent d'ordre technique, parfois théorique. Un échange entre deux participants, au sujet d'un point de vue présenté par un participant tiers, peut aussi éclairer un cas particulier de la PEP. Si cet échange est complexe, il est souhaitable d'accompagner son avis de liens et de références, fournir des liens de références est ainsi une autre forme de participation aux débats. Dans le cadre de la revue de la PEP, il est également possible de poser une question. On remarque toutefois que cette question doit être suffisamment légitime pour soulever un cas relatif à la PEP en question et non pas simplement pour se faire expliquer un élément basique de programmation. Enfin, faire de l'humour n'est pas une chose impossible lors de ces échanges. On relève en effet à l'occasion des notes d'ironie ou de dérision, mais ceci est toujours bienveillant. L'ensemble des débats se déroule toujours dans une ambiance de respect mutuel.

5.5.3 Orienter les débats

Lors d'une troisième activité durant les échanges, il est possible d'orienter les débats. Il s'agit d'une forme de participation active dans la conversation : cela va permettre l'orientation technique, l'évolution des opinions et la prise de décision, soit par « directive », soit par consensus.

On remarque en effet que quelques personnes, comme Guido ou des développeurs possédant un capital symbolique fort comme le champion, peuvent donner des directives. Ainsi, par ces commentaires avisés, ils dirigent les choix d'implémentation qui sont faits. Il s'agit d'un élément capital du processus PEP, car le BDFL reste l'autorité finale dans le cadre des activités de conception du langage. C'est ainsi que cette autorité prend forme dans l'action collective.

En outre, il arrive que des développeurs proposent des alternatives de conception. C'est ainsi que les développeurs peuvent directement critiquer la PEP. Reprenant la phrase d'Elias (*core developer*) : « *It's perfectly OK to look critically at all new proposals. Having one way to do it is a Python design goal.* ». Ainsi, il est tout à fait correct de se montrer critique vis-à-vis d'une nouvelle proposition, cependant avoir « une seule manière de faire » dans Python est un but de conception.

Aussi, on relève des cas où le développeur peut demander une amélioration de manière directe. Ensuite, assez classiquement, un développeur va aborder un cas d'utilisation et partager un retour d'expérience, par exemple comparer Python à un autre langage de programmation par rapport au problème soulevé.

Cependant, le cas le plus fréquent dans cette catégorie de discours est celui de proposer un exemple de code. C'est le moyen « codifié » de se faire comprendre par tous : expliciter son point de vue par du code et ainsi utiliser le langage commun, celui qui réuni chacun dans sa pratique : la programmation informatique.

5.5.4 Modifier la PEP ou le code

Enfin, cette activité du discours exprime la finalité de la consultation des membres de la communauté et permet ainsi l'évolution du code produit. Deux types d'activités apparaissent : proposer directement une correction à l'auteur de la PEP ou changer l'implémentation. Dans les deux cas, il ne s'agit plus ici de réfléchir ensemble, mais plutôt d'appliquer les modifications souhaitées, soit à la spécification technique que représente la PEP, soit au code. C'est le résultat du processus de PEP : parvenir à une proposition d'évolution à la fois (1) consensuelle aux yeux de l'ensemble des membres de la communauté ; (2) revue et corrigée par des pairs développeurs et (3) validée par l'autorité de la communauté, soit le BDFL.

Le tableau 5.3 récapitule les activités lors du processus PEP. On constate que ces activités sont variées et prennent de multiples formes. Le processus PEP et les discussions associées sont ainsi déconstruits pour mettre au jour leur complexité. Ce tableau montre la diversité des activités qui ont lieu lors des échanges au sujet d'une PEP et la manière dont les développeurs peuvent intervenir dans les discussions qui sont au cœur même du processus d'innovation par les utilisateurs. Ainsi, la diversité de ces activités enrichit le contenu des discussions. Cela va de l'annonce d'une idée à l'évaluation de celle-ci, puis à la mise en commun d'une proposition, à sa correction et finalement à la proposition améliorée d'un code source et d'une PEP validés formant ainsi le terrain propice à l'innovation par les utilisateurs. Le processus PEP, basé sur l'échange des utilisateurs sur les listes de discussion de la communauté au sujet des évolutions du langage, est le mécanisme par lequel l'innovation par les utilisateurs prend forme en ligne et de manière distribuée au sein de la communauté Python.

Tableau 5.3 : Les activités lors du processus PEP

Activité	Sous-activité
Présenter les enjeux	Annoncer le sujet
	Décrire le processus
	Demander des commentaires
Participer aux discussions	Apporter son soutien
	Donner son avis
	Fournir des liens vers des références
	Fournir une explication
	Poser une question
	Faire de l'humour
	Résumer les enjeux
Orienter les débats	Donner des directives
	Faire une proposition
	Demander une amélioration
	Aborder un cas d'utilisation
	Partager un retour d'expérience
	Proposer un exemple de code
	Soulever une contradiction
Modifier la PEP ou le code	Proposer une correction
	Changer l'implémentation

5.6 Synthèse

Dans ce cinquième chapitre, nous avons analysé le processus d'innovation mis en œuvre collectivement par la communauté Python. Pour ce faire, la communauté utilise d'une part, le vote en ligne comme outil participatif de décision et de consultation et d'autre part, le processus de proposition d'amélioration du langage (PEP). Ce dernier est un processus formel unique à la communauté Python qui est utilisé pour décrire tant les évolutions techniques majeures du langage que les procédures du fonctionnement interne de la communauté. Il se présente sous la forme de documents publiés en ligne, qui sont discutés, rédigés et amendés par les membres de la communauté à travers les listes de discussion. Une proposition d'amélioration

(PEP) est validée en deux phases, d'abord lors d'une première phase sur la liste *python-ideas* puis lors d'une seconde phase sur la liste *python-dev*. La décision finale d'acceptation d'une PEP est accordée par le BDFL sur la liste *python-dev*.

Le processus d'amélioration du langage constitue une innovation organisationnelle. L'auteur d'une PEP, appelé le « champion » est un utilisateur-innovateur et le processus PEP permet d'inclure les utilisateurs durant la phase d'innovation. C'est ce que nous avons montré à l'aide de nos deux cas : (1) l'entrée en phase d'innovation est décidée par les utilisateurs eux-mêmes ; (2) les choix techniques sont faits sur la base commune : le langage Python ; (3) les décisions sont prises sur la base de consensus entre utilisateurs ; (4) le BDFL détient l'autorité procédurale qu'il exerce de manière forte en validant les choix d'implémentations majeurs. Il joue deux rôles, à la fois celui d'un expert technique et celui de manager de l'innovation. Enfin, par l'analyse des activités lors du processus PEP, nous montrons la variété des activités exercées par les membres de la communauté et ce qui permet de concevoir l'innovation comme un processus collectif.

CONCLUSION

En conclusion, nous présentons de façon synthétique les principaux éléments ayant émergé de cette recherche. Ensuite, nous revenons sur les limites inhérentes à celle-ci et enfin nous proposons des pistes de réflexion pour de prochaines études.

Les communautés du logiciel libre et *open source* constituent un mouvement original dans l'écosystème des initiatives du monde de l'industrie informatique. Dans le cadre de cette étude, notre analyse a porté sur la communauté Python, son organisation sociale et la mise en œuvre de l'innovation par cette dernière. Nous voulions ainsi mettre au jour les formes organisationnelles de cette communauté dont les membres sont des bénévoles passionnés et dont les activités ne sont pas régies par le cadre classique de l'entreprise. Notre recherche poursuivait trois objectifs principaux : (1) dresser un portrait de la communauté en tant qu'organisation ; (2) décrire le mécanisme d'innovation à l'œuvre au sein de la communauté ; (3) mettre au jour et analyser un ensemble de discussions internes visant à faire évoluer le langage de programmation.

Pour mener à bien cette étude, nous avons mobilisé trois approches théoriques complémentaires, à même de refléter la réalité de la communauté Python. D'abord, la théorie de la communauté de pratique (Wenger, McDermott et Snyder, 2002) puisqu'il s'agissait, dans cette recherche, de décrire et de comprendre l'organisation sociale d'une communauté d'individus réunis autour d'une pratique, celle du développement logiciel. Nous voulions observer l'action collective à l'œuvre, modéliser la manière dont la communauté est organisée, identifier les rôles et statuts

des membres de la communauté. Ensuite, nous avons mobilisé le concept de communauté épistémique pour nous permettre de comprendre l'activité de création, de gestion et de circulation des connaissances au sein de la communauté. Nous voulions être à même de comprendre sur quoi était basée l'expertise des membres, comment s'organisait la codification des connaissances et la production de nouvelles connaissances. Enfin, nous avons mobilisé la théorie de l'innovation par l'utilisateur (Von Hippel, 1988, 2005) comme cadre nous permettant de comprendre le processus d'innovation à l'œuvre dans la communauté Python. Nous voulions mettre au jour le rôle des utilisateurs au sein de la communauté et comprendre en quoi la communauté Python était une communauté innovante.

Sur le plan méthodologique, nous avons choisi d'entreprendre une recherche de type qualitatif basée une ethnographie virtuelle (Hine, 2000) et sur une observation non participante en ligne des sites web et listes de discussion que maintiennent la communauté Python. Ainsi, l'étude descriptive de type ethnographique de la communauté est au cœur de cette recherche. Sur la base du matériel recueilli lors de la phase d'observation, nous avons procédé à une analyse de contenu et nous avons pu ainsi relever l'organisation interne et faire un certain nombre de constatations. Nous avons d'abord analysé la dimension sociale de la pratique de développement logiciel et la manière dont les membres de la communauté sont organisés. La communauté Python est une communauté de pratique ayant pour objet le développement d'un logiciel innovant : Python. La communauté Python est une communauté virtuelle, dans le sens où elle utilise exclusivement les technologies de l'information, à savoir principalement le courriel et des outils en ligne, pour se coordonner. Les membres de la communauté échangent en effet, par courriel et sur des listes de discussion, pour mener à bien leur projet commun. Pour ce faire, les membres de la communauté sont organisés autour du chef de projet et initiateur Guido van Rossum. Le chef de projet, appelé BDFL pour *Benevolent Dictator for Life*, est l'autorité finale en ce qui concerne la conception du logiciel.

Cependant, outre ce rôle majeur, notre étude montre qu'il existe plusieurs autres rôles et statuts au sein de la communauté. Le *release manager* est la personne qui coordonne les mises en production (*release*). Il gère les éléments intégrés dans le code source afin de produire un livrable, le logiciel Python. Les administrateurs du projet (*project admins*) sont un petit groupe d'une dizaine d'individus qui possèdent des droits et autorisations sur l'infrastructure du projet lui-même, c'est-à-dire sur les outils qu'utilisent les membres de la communauté pour développer le code source. Ils ont autorité pour fournir aux membres de la communauté le droit en écriture sur le dépôt central de code.

Ensuite, les développeurs Python sont répartis en plusieurs groupes distincts selon leur niveau d'autorité dans la communauté. Les *Python committers* sont les membres de la communauté ayant le droit en écriture sur le dépôt de code central. Leur activité est l'écriture du code source en leur nom propre ou le report de fonctionnalités développées par d'autres développeurs. Les experts, ou les mainteneurs de module (*module maintainers*), sont des développeurs ayant un rôle d'expertise sur des portions (ou modules) spécifiques de code. Les *Python contributors* forment un ensemble de développeurs ayant signé le formulaire du contributeur Python (*Contributor Agreement*) et qui n'ont pas accès en écriture au dépôt central de code. Ils proposent des modifications de code (*patches*) à travers l'outil en ligne de gestion des anomalies qui sont ensuite revues et validées par un ou plusieurs développeurs ayant une autorité plus importante (*Python committers*, experts). Enfin, les utilisateurs de Python (*Python users*) sont les utilisateurs du langage de programmation. Leur rôle, lorsqu'ils sont actifs, est de relever des anomalies dans le fonctionnement du logiciel et de produire de la documentation. La structure qui caractérise cette organisation est de forme pyramidale, où les utilisateurs de Python forment la base et le chef de projet le sommet.

Une autre caractéristique importante de la communauté Python est que ses membres se réfèrent dans leurs activités à des documents rédigés collectivement en ligne : les propositions d'améliorations de Python (PEP). Ces documents ont de multiples fonctions, ils peuvent soit préciser les procédures et fonctionnements internes, soit décrire une nouvelle fonctionnalité dans le langage de programmation. Lorsqu'ils décrivent des procédures, ils régissent le fonctionnement interne de la communauté. Lorsqu'ils décrivent une nouvelle fonctionnalité du langage, ces documents sont débattus en ligne et rédigés collectivement alors même que les évolutions du langage sont produites. La publication et la discussion en ligne des PEP permettent aux membres de la communauté de participer aux décisions de conception du langage. Ainsi, le mécanisme de proposition d'amélioration du langage vient soutenir l'activité d'innovation par les utilisateurs au sein de la communauté. L'innovation est alors menée conjointement par des utilisateurs-pionniers, ou *lead users*, et de multiples utilisateurs-évaluateurs. Les premiers expriment des besoins nouveaux en amont de ceux de la communauté, rédigent les spécifications techniques inhérentes au développement d'une nouvelle fonctionnalité, mènent les débats en ce qui a trait à l'évolution proposée et enfin, rédigent le code correspondant. Les seconds ont comme rôle de discuter, valider et amender la proposition d'évolution du langage proposée. Cette structure de production de l'innovation est appelée réseau horizontal d'innovation, parmi lequel tous les utilisateurs expérimentés du langage sont appelés à s'exprimer quant aux évolutions proposées, et ce, dans un mécanisme qui ne fait intervenir nulle autre autorité que celle des membres de la communauté. Le logiciel Python est alors utilisé comme une boîte à outils logicielle permettant aux utilisateurs d'examiner, de tester et de modifier le code source puis d'incorporer leurs propositions d'évolutions au logiciel lui-même, le logiciel Python et sa communauté fournissant tous les outils pour ce faire.

Dans le contexte du champ des sciences, technologies et société, cette étude permet de mettre en lumière l'innovation produite par une communauté *open source*.

Ce faisant, elle ouvre la voie à une réflexion sur de nouvelles formes de propriété intellectuelle et de partage de l'information. En effet, la communauté Python formalise les règles d'utilisation, de distribution et de partage du code source à travers sa licence PSF compatible GPL. Cette licence est également ouverte en ce qui concerne la réutilisation commerciale du code source et du logiciel Python. C'est un des points forts de Python que d'être disponible sur de nombreuses plateformes informatiques et distribué par de nombreux agents. Ce type de licence est dit *open source,* car elle respecte les dix critères de l'Open Source Initiative et ainsi toute utilisation non discriminante du logiciel et de son code source.

Sur la base de cette licence, le cas de Python est une initiative originale dans le domaine des technologies de l'information que l'on a souhaité étudié à travers le champ STS. En effet, ce logiciel est une innovation menée collectivement, de manière collaborative et ouverte, entre les utilisateurs du langage. Sur le plan technique, Python constitue un langage innovant formalisé par des balises que sont le *Zen de Python* et le *Guide du développeur de Python*. Ceci est tout à fait original dans la conduite d'un projet informatique en regard du processus d'innovation technique de la communauté puisque encadré et systématique, et ainsi permettant de mener à bien l'innovation numérique.

Toutefois, toute étude présente des limites et celle-ci n'y échappe pas. Tout d'abord, pour des raisons de temps et de faisabilité, l'observation était non participante : nous ne nous sommes pas mêlés aux utilisateurs actifs de Python et n'avons pas pris part aux conversations sur les développements en cours. Nous avons plutôt choisi d'étudier des développements passés et d'analyser les traces de ces discussions. De ce fait, nos observations n'ont pas eu lieu en temps réel et ne portaient pas sur des discours des utilisateurs eux-mêmes sur une action collective en cours, mais bien des seuls résultats objectifs de leurs activités. Ensuite, l'observation n'a pu durer que quatre semaines et notre analyse se pencher que sur seulement deux cas d'études.

Nous n'avons pas pu examiner l'ensemble des cas et des problèmes de conception rencontrés par la communauté, ni donc des réponses variées qu'ils ont suscitées au cours d'un cycle complet de développement par exemple, notre étude est de fait réduite à nos cas. Enfin, il faut noter que l'analyse manuelle de quelques centaines de courriels n'offre pas la profondeur que pourrait apporter l'analyse informatique en terme de relations entre individus, de réseau de confiance et d'analyse de contenu.

Pour terminer, nous concluons cette section en proposant des pistes de recherche qu'il pourrait être intéressant de poursuivre dans le cadre de futures études. Tout d'abord, la présente recherche est manuelle et porte sur un nombre relativement limité de courriels échangés entre les *core developers*. Il serait donc intéressant d'automatiser la recherche, avec des outils informatiques dédiés à l'analyse de réseaux, afin de montrer les liens qui unissent fortement ou faiblement les utilisateurs du langage et leur poids dans les conversations étudiées. Ensuite, il serait intéressant d'interroger des utilisateurs clés de la communauté Python, notamment le chef de projet, afin de connaître ses réflexions sur la forme et l'organisation de la communauté Python. Enfin, nous pourrions comparer, lors d'une étude future, l'organisation et le fonctionnement de la communauté Python avec ceux d'une autre communauté *open source*, l'organisation et le développement logiciel communautaires étant une tendance de fond en informatique.

ANNEXES

ANNEXE A : EXTRAITS DE LA PEP 1

What is a PEP ?

PEP stands for Python Enhancement Proposal. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment. The PEP should provide a concise technical specification of the feature and a rationale for the feature.

We intend PEPs to be the primary mechanisms for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python. The PEP author is responsible for building consensus within the community and documenting dissenting opinions.

Many ideas have been brought forward for changing Python that have been rejected for various reasons. Asking the Python community first if an idea is original helps prevent too much time being spent on something that is guaranteed to be rejected based on prior discussions (searching the internet does not always do the trick). It also helps to make sure the idea is applicable to the entire community and not just the author. Just because an idea sounds good to the author does not mean it will work for most people in most areas where Python is used.

"The final authority for PEP approval is the BDFL. However, whenever a new PEP is put forward, any core developer that believes they are suitably experienced to make the final decision on that PEP may offer to serve as the BDFL's delegate (or "PEP czar") for that PEP. If their self-nomination is accepted by the other core developers and the BDFL, then they will have the authority to approve (or reject) that PEP. This process happens most frequently with PEPs where the BDFL has granted in principle approval for *something* to be done, but there are details that need to be worked out before the PEP can be accepted.

If the final decision on a PEP is to be made by a delegate rather than directly by the BDFL, this will be recorded by including the "BDFL-Delegate" header in the PEP.

ANNEXE B : EXTRAITS DU FORMULAIRE DU CONTRIBUTEUR

This agreement solves, for the PSF, the major issue of relicensing in open source software: Formally, most licenses at most allow for relicensing the software on the same terms as the original license. With the license form, PSF receives the permission to relicense the software under a different (open source) license than the initial license - and indeed we distribute Python under the Python license, not under one of the initial licenses.

At the same time, contributors remain copyright holders of their contributions - they are not asked to assign the copyright to the PSF. They only formally confirm the terms under which they made their contribution available to the PSF.

Contributor understands and agrees that PSF shall have the irrevocable and perpetual right to make and distribute copies of any Contribution, as well as to create and distribute collective works and derivative works of any Contribution, under the Initial License or under any other open source license approved by a unanimous vote of the PSF board. Contributor shall identify each Contribution by placing the following notice in its source code adjacent to Contributor's valid copyright notice: "Licensed to PSF under a Contributor Agreement."

ANNEXE C : CITATIONS BUG REPORTER

"Waiting for the bug reporter's feedback."
(Guido van Rossum)

"At nearly 5 years of age, this issue unfortunately never saw a viable demonstration of how to provoke the reported behavior. Hopefully the reporter either (1) discovered the nature of the problem originated elsewhere and he was able to quickly fix it, or (2) won the lottery and merrily forgot all about this issue (and many other real-world issues)."
(Davin)

"I can certainly write the reporter glue to work with either a string or a full reference."
(Robert)

"My problem, and the problem if the original bug reporter (sirilyan) is that the load method ignores names that don't have values."
(Andres)

"Ok, downgrading to critical.
I'm awaiting the reporter's answer anyway."
(Antoine)

"I have been able to reproduce it in python 2.5 and 2.7, on debian 6 (64 bits). Exactly like the reporter, I am making extensive use of threads."
(Manu)

"Since both the reporter and I believes that this is not a bug in the subprocess module, I'm stepping back."
(Peter)

"We can't do anything about it, the bug reporter can take it up with server."
(Senthil)

ANNEXE D : EXTRAITS DU GUIDE DU DÉVELOPPEUR DE PYTHON

Reporting a bug

Python is a mature programming language which has established a reputation for stability. In order to maintain this reputation, the developers would like to know of any deficiencies you find in Python.

Documentation bugs : If you find a bug in this documentation or would like to propose an improvement, please submit a bug report on the tracker. If you have a suggestion how to fix it, include that as well.

Using the Python issue tracker

Bug reports for Python itself should be submitted via the Python Bug Tracker (https://bugs.python.org/). The bug tracker offers a Web form which allows pertinent information to be entered and submitted to the developers." (PSF, bugs.html)

"The first step in filing a report is to determine whether the problem has already been reported. The advantage in doing so, aside from saving the developers time, is that you learn what has been done to fix it; it may be that the problem has already been fixed for the next release, or additional information is needed (in which case you are welcome to provide it if you can!). To do this, search the bug database using the search box on the top of the page .

"If the problem you're reporting is not already in the bug tracker, go back to the Python Bug Tracker and log in. If you don't already have a tracker account, select the "Register" link or, if you use OpenID, one of the OpenID provider logos in the sidebar. It is not possible to submit a bug report anonymously.

Being now logged in, you can submit a bug. Select the "Create New" link in the sidebar to open the bug reporting form.

The submission form has a number of fields. For the "Title" field, enter a *very* short description of the problem; less than ten words is good. In the "Type" field, select the type of your problem; also select the "Component" and "Versions" to which the bug relates.

In the "Comment" field, describe the problem in detail, including what you expected to happen and what did happen. Be sure to include whether any extension modules were involved, and what hardware and software platform you were using (including version information as appropriate).

Each bug report will be assigned to a developer who will determine what needs to be done to correct the problem. You will receive an update each time action is taken on the bug.

Getting started contributing to Python yourself :Beyond just reporting bugs that you find, you are also welcome to submit patches to fix them. You can find more information on how to get started patching Python in the Python Developer's Guide. If you have questions, the core-mentorship mailing list is a friendly place to get answers to any and all questions pertaining to the process of fixing issues in Python.

How to Become a Core Developer

What it Takes

When you have consistently contributed patches which meet quality standards without requiring extensive rewrites prior to being committed, you may qualify for commit privileges and become a core developer of Python. You must also work well with other core developers (and people in general) as you become an ambassador for the Python project.

Typically a core developer will offer you the chance to gain commit privilege. The person making the offer will become your mentor and watch your commits for a while to make sure you understand the development process. If other core developers agree that you should gain commit privileges you are then extended an official offer.

You may request commit privileges yourself, but do not be surprised if your request is turned down. Do not take this personally! It simply means that other core developers think you need more time contributing patches before you are able to commit them without supervision.

ANNEXE E : EXEMPLES DE VOTES EN LIGNE

Bart votant l'autorisation de Bernard :

```
"Works for me to give Bernard commit privileges."
```

John votant l'autorisation de Bernard :

```
"I had just that thought two days ago when applying a patch
of his."
```

Andy votant l'autorisation de Bernard :

```
"+1"
```

Tony votant l'autorisation de Robert :

```
"Entirely +1."
```

William votant l'autorisation de Robert :

```
"+1, definitely"
```

Bart validant l'autorisation de Robert après de nombreux votes :

```
"That's enough +1s. =) I'll let Robert know that he's been
approved from commit privileges."
```

BIBLIOGRAPHIE

AMIN A., COHENDET P., 2004, *Architectures of Knowledge : firms, capabilities, and communities*, Oxford, Oxford University Press.

AURAY N., 2007, « Le modèle souverainiste des communautés en ligne : Impératif participatif et désacralisation du vote », *Hermès*, 47, p. 137-144.

BARATS C., 2013, *Manuel d'analyse du Web en sciences humaines et sociales*, Paris, Armand Colin.

BERGQUIST M., LJUNGBERG J., 2001, « The power of gifts: organizing social relationships in open source communities », *Information Systems Journal*, *11*, 4, p. 305-320.

BEUSCART J.-S., DAGIRAL E., PARASIE S., 2009, « Sociologie des activités en ligne (introduction) », *Terrains & travaux*, *1*, 15, p. 3-28.

BONNEUIL C., JOLY P.-B., 2013, *Sciences, techniques et société*, Paris, La Découverte.

BONNEVILLE L., GROSJEAN S., LAGACE, M., 2007, *Introduction aux méthodes de recherche en communication*, Montréal, G. Morin.

BOURDIEU P., 1980, « Le capital social », *Actes de la recherche en sciences sociales*, *31*, 1, p. 2-3.

BROCA S., 2013, *Utopie du logiciel libre : du bricolage informatique à la réinvention sociale*, Neuvy-en-Champagne, Le passager clandestin.

BURT R.S., 1995, « Le capital social, les trous structuraux et l'entrepreneur », *Revue Française de Sociologie*, *36*, 4, p. 599.

COHENDET P., CREPLET F., DUPOUËT O., 2003, « Innovation organisationnelle, communautés de pratique et communautés épistémiques : le cas de Linux », *Revue française de gestion*, *29*, 146, p. 99-121.

COHENDET P., DIANI M., 2003, « L'organisation comme une communauté de communautés croyances collectives et culture d'entreprise », *Revue d'économie politique*, *113*, 5, p. 697-720.

CONSEIL DE RECHERCHES EN SCIENCES HUMAINES DU CANADA, CONSEIL DE RECHERCHES EN SCIENCES NATURELLES ET EN GENIE DU CANADA, INSTITUTS DE RECHERCHE EN SANTE DU CANADA, 2010, *Énoncé de politique des trois Conseils : Éthique de la recherche avec des êtres humains*, 234p.

CORIS M., 2007, « La culture du don dans la modernité: Les communautés du logiciel libre », *Réseaux*, *140*, 1, p. 161.

CORIS M., LUNG Y., 2005, « Les communautés virtuelles : la coordination sans proximité ? Les fondements de la coopération au sein des communautés du logiciel libre », *Revue d'Économie Régionale & Urbaine*, *Juillet*, 3, p. 397.

CREMER J., GAUDEUL A., 2004, « Quelques éléments d'économie du logiciel libre », *Réseaux*, *124*, 2, p. 111-139.

CROWSTON K., HOWISON J., 2005, « The social structure of free and open source development », *First Monday*, *10*, 2.

CROWSTON K., HOWISON J., 2006, « Hierarchy and centralization in free and open source software team communications », *Knowledge, Technology & Policy*, *18*, 4, p. 65-85.

CROWSTON K., LI Q., WEI K., ESERYEL U.Y., HOWISON J., 2007, « Self-organization of teams for free/libre open source software development », *Information and Software Technology*, *49*, 6, p. 564-575.

CUCCHI A., FUHRER C., 2011, « Capital social et usage des technologies de l'information et de la communication (TIC) : une analyse par les réseaux sociaux », *Management & Avenir*, *45*, 5, p. 179.

DAVID P.A., FORAY D., 2002, « Une introduction à l'économie et à la société du savoir », *Revue internationale des sciences sociales*, *171*, 1, p. 13.

DEMAZIERE D., HORN F., ZUNE M., 2006, « Dynamique de développement des communautés du logiciel libre »,.

DEMAZIERE D., HORN F., ZUNE M., 2007a, « Des relations de travail sans règles ? L'énigme de la production des logiciels libres », *Sociétés contemporaines*, *66*, 2, p. 101-125.

DEMAZIERE D., HORN F., ZUNE M., 2007b, « The Functioning of a Free Software Community: Entanglement of Three Regulation Modes–Control, Autonomous and Distributed. », *Science Studies*, *20*, 2.

DEMAZIERE D., HORN F., ZUNE M., 2009, « La socialisation dans les «communautés» de développement de logiciels libres », *Sociologie et sociétés*, *41*, 1, p. 217-238.

DEMAZIERE D., HORN F., ZUNE M., 2011, « Ethnographie de terrain et relation d'enquête. Observer les «communautés» de logiciels libres », *Sociologie*, *2*, 2, p. 165-183.

FEENBERG A., 2004, *(Re)penser la technique : vers une technologie démocratique*, Paris, La Découverte / M.A.U.S.S.

FLICHY P., 2010, « Le mouvement des logiciels libres », dans *Le sacre de l'amateur : sociologie des passions ordinaires à l'ère numérique*, Paris, Seuil.

FORAY D., 2000, *L'économie de la connaissance*, Paris, La Découverte.

FRANKE N., VON HIPPEL E., 2003, « Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software », *Research Policy*, *32*, 7, p. 1199-1215.

GACEK C., ARIEF B., 2004, « The many meanings of open source », *Software, IEEE*, *21*, 1, p. 34-40.

GAGLIO G., 2011, *Sociologie de l'innovation*, Paris, Presses universitaires de France (Que sais-je?), 128 p.

GINGRAS Y., 2013, *Sociologie des sciences*, Paris, Presses universitaires de France (Que sais-je?), 128 p.

GIURI P., RULLANI F., TORRISI S., 2008, « Explaining leadership in virtual teams: The case of open source software », *Information Economics and Policy*, *20*, 4, p. 305-315.

GUIMARÃES M.J.L., 2005, « Doing Anthropology in Cyberspace: Fieldwork Boundaries and Social Environments », dans HINE C. (dir.), *Virtual methods issues in social research on the Internet*, Oxford; New York, Berg.

HAAS P., 1992, « Introduction : Epistemic Communities and International Policy Coordination », *International Organization*, *46*, 1, p. 1-35.

HENRI F., PUDELKO B., 2006, « Le concept de communauté virtuelle dans une perspective d'apprentissage social », dans *Comprendre les communautés virtuelles d'enseignants : pratiques et recherches*, Paris, L'Harmattan.

HILDRETH P., KIMBLE C., WRIGHT P., 2000, « Communities of practice in the distributed international environment », *Journal of Knowledge management*, *4*, 1, p. 27-38.

HILLAIRET D., 2012, « Créativité et inventivité des utilisateurs-pionniers. Le cas de la communauté des kitesurfers », *Revue française de gestion*, *38*, 223, p. 91-104.

HIMANEN P., 2001, *L'éthique hacker et l'esprit de l'ère de l'information*, Paris, Exils.

HINE C., 2000, *Virtual ethnography*, London, Sage.

HINE C., 2005, *Virtual methods issues in social research on the Internet*, Oxford ; New York, Berg.

KAROUI ZOUAOUI S., HCHICH HEDHLI R., 2014, « Communautés de savoir et innovation : le rôle de l'apprentissage. Une analyse sous l'éclairage d'une théorie basée sur les connaissances », *Management & Avenir*, *67*, 1, p. 155.

LAKHANI K.R., VON HIPPEL E., 2003, « How open source software works : "free" user-to-user assistance », *Research policy*, *32*, 6, p. 923-943.

LATRIVE F., 2004, « Savoirs et cultures libres », dans *Du bon usage de la piraterie : culture libre, sciences ouvertes*, Paris, Exils.

LATZKO-TOTH G., 2010, « Metaphors of Synchrony: Emergence and Differentiation of Online Chat Devices », *Bulletin of Science, Technology & Society*, *30*, 5, p. 362-374.

LAZARO C., 2008, *La liberté logicielle : une ethnographie des pratiques d'échange et de coopération au sein de la communauté Debian*, Louvain-la-Neuve, Academia Bruylant.

LERNER J., TIROLE J., 2002, « Some simple economics of open source », *The journal of industrial economics*, *50*, 2, p. 197-234.

LEVY S., 2013, *L'éthique des hackers*, Paris, Globe.

LJUNGBERG J., 2000, « Open source movements as a model for organising », *European Journal of Information Systems*, *9*, 4, p. 208-216.

MANGOLTE P.-A., 2005, « Le "Chaudron Magique" et "L'Invention Collective" », *Economie Appliquée*, *LVIII*, 1, p. 59-83.

MARTINEZ-TORRES M.R., DIAZ-FERNANDEZ M.C., 2014, « Current issues and research trends on open-source software communities », *Technology Analysis & Strategic Management*, *26*, 1, p. 55-68.

MASMOUDI H., 2006, *La résolution distribuée dans les communautés Open Source : propriétés organisationnelles et modes de coordination*, Thèse de doctorat, Université Paris-Dauphine.

MEISSONIER R., BOURDON I., HOUZE E., AMABILE S., BOUDRANDI S., 2010, « Comprendre les motivations des développeurs de l'open source à partir de leur participation », *Systèmes d'information & management*, *15*, 2, p. 71-97.

MEYER M., MONTAGNE F., 2007, « Le logiciel libre et la communauté autorégulée », *Revue d'économie politique*, *Vol. 117*, 3, p. 387-405.

MEYER M., MOLYNEUX-HODGSON S., 2011, « «Communautés épistémiques» : une notion utile pour théoriser les collectifs en sciences ? », *Terrains & travaux*, 1, p. 141-154.

MULLER P., 2004, « Autorité et gouvernance des communautés intensives en connaissances : une application au développement du logiciel libre », *Revue d'économie industrielle*, *106*, 1, p. 49-68.

OLIVERI N., 2011, « Logiciel libre et open source : ue culture du don technologique », *Quaderni*, 76.

O'MAHONY S., 2003, « Guarding the commons: how community managed software projects protect their work », *Research Policy*, *32*, 7, p. 1179-1198.

O'MAHONY S., 2005, « Nonprofit Foundations and Their Role in Community-Firm Software Collaboration »,.

O'MAHONY S., 2007, « The governance of open source initiatives: what does it mean to be community managed? », *Journal of Management & Governance*, *11*, 2, p. 139-150.

O'MAHONY S., FERRARO F., 2007, « The emergence of governance in an open source community », *Academy of Management Journal*, *50*, 5, p. 1079-1106.

PELLEGRINI F., CANEVET S., 2013, *Droit des logiciels : logiciels privatifs et logiciels libres*, Paris, Presses universitaires de France.

PINCH T J., BIJKER W.E., 1984, « The Social Construction of Facts and Artefacts: Or How the Sociology of Science and the Sociology of Technology Might Benefit Each Other », *Social Studies of Science*, *14*, 3, p. 399-441.

PONTHIEUX S., 2006, *Le capital social*, Paris, La Découverte.

PROULX S., 2006, « Les communautés virtuelles : ce qui fait le lien », dans PROULX S., POISSANT L., SENECAL M. (dirs.), *Communautés virtuelles : penser et agir en réseau*, Québec, Presses de l'Université Laval.

RAYMOND E., 1999, *The Cathedral and the Bazaar : Musings on Linux and Open Source by an Accidental Revolutionary*, Sebastopol, CA, O'Reilly.

RAYMOND E., 2000, « Comment devenir un hacker », dans BLONDEAU O. (dir.), *Libres enfants du savoir numérique*, p. 255-277.

SABOURIN P., 2003, « L'analyse de contenu », dans GAUTHIER B. (dir.), *Recherche sociale: de la problématique à la collecte des données*, Sainte-Foy, Presses de l'Université du Québec.

SACK W., DETIENNE F., DUCHENEAUT N., BURKHARDT J.-M., MAHENDRAN D., BARCELLINI F., 2006, « A Methodological Framework for Socio-Cognitive Analyses of Collaborative Design of Open Source Software », *Computer Supported Cooperative Work (CSCW)*, *15*, 2-3, p. 229-250.

SCHEID F., CHARUE-DUBOC F., 2011, « Le rôle des lead users dans le processus d'innovation logicielle », *Revue française de gestion*, *37*, 210, p. 133-147.

SNYDER W., WENGER E., 2010, « Our world as a learning system: A communities-of-practice approach », dans *Social learning systems and communities of practice*, London, Springer.

SQUIRE M., 2012, « How the FLOSS Research Community Uses Email Archives », *International Journal of Open Source Software and Processes*, *4*, 1, p. 37-59.

STALLMAN R.M., WILLIAMS S., MASUTTI C., 2010, *Richard Stallman et la révolution du logiciel libre*, Paris, Eyrolles.

TRAVERSO V., 1999, *L'analyse des conversations*, Paris, Nathan.

VON HIPPEL E., 1986, « Lead users: A source of novel product concepts », *Management Science*, *32*, 7.

VON HIPPEL E., 1988, *The sources of innovation*, New York, Oxford University Press.

VON HIPPEL E., 2001, « Innovation by user communities: Learning from open-source software », *MIT Sloan management review*, *42*, 4, p. 82-86.

VON HIPPEL E., 2005, *Democratizing Innovation*, Cambridge, Mass., MIT Press.

VON HIPPEL E., 2007, « Horizontal innovation networks--by and for users », *Industrial and Corporate Change*, *16*, 2, p. 293-315.

WANG J., 2005, « The role of social capital in open source software communities », *AMCIS 2005 Proceedings*, p. 427.

WEBER S., 2004, *The Success of Open Source*, Cambridge, MA, Harvard University Press.

WENGER E., 1998, *Communities of practice : learning, meaning, and identity*, Cambridge, U.K.; New York, N.Y., Cambridge University Press.

WENGER E., MCDERMOTT R.A., SNYDER W., 2002, *Cultivating communities of practice: a guide to managing knowledge*, Boston, Mass., Harvard Business School Press.

WILLIAMS S., 2002, *Free as in freedom : Richard Stallman's crusade for free software*, Sebastopol, CA, O'Reilly.