Gouvernance d'un projet libre : contrôler un flux d'innovation

Nicolas JULLIEN
IMT Atlantique, LEGO-Marsouin

Robert VISEUR
Université de Mons

Et Jean-Benoît ZIMMERMANN
GREQAM-AMSE (Aix-Marseille Université, CNRS et EHESS)

Dans cet article, nous analysons les modes de gouvernance des projets de logiciel libre, c'est-à-dire l'ensemble des moyens mis en œuvre pour l'orientation, le contrôle et la coordination d'agents économiques totalement ou partiellement autonomes – individus, organisations utilisatrices de la solution, entreprises utilisant la solution pour construire des offres commerciales... – pour le compte d'un projet de développement libre. Ces aspects de gouvernance jouent un rôle essentiel dans l'organisation d'un système d'innovation ouverte, qui fonctionne tant que la dynamique d'innovation est forte.

Introduction

Comme expliqué dans l'article « Le logiciel libre : gérer collectivement les évolutions d'une technologie » (pp. 70-76) de ce numéro, le libre ne constitue pas une négation du principe de propriété intellectuelle, il utilise celle-ci pour organiser l'évolution continue de la demande et de l'innovation. Ce système a été créé par et pour l'utilisateur-innovateur (Lakhani et von Hippel, 2003 ; von Hippel et von Krogh, 2003) : bénéficiant directement de l'innovation qu'il produit, il est incité à la produire et, comme il peut s'attendre à des retours d'information ou des innovations cumulatives sur sa proposition, il bénéficie également d'effets d'apprentissage individuel (Foray et Zimmermann, 2001). Pour un utilisateur doté de compétences informatiques avancées, réaliser lui-même les adaptations dont il a besoin est souvent moins coûteux que d'attendre que l'éditeur les réalise. Du point de vue du projet, une gouvernance ouverte de l'innovation permet l'accès à des connaissances dispersées ainsi que la résolution de problèmes complexes, à condition d'investir dans les compétences nécessaires à la formulation et la décomposition des problèmes (Felin et Zenger, 2014). Moins coûteux ne signifie pas sans coût, et construire de tels projets reste difficile, même pour les utilisateurs les plus compétents (Ãgerfalk et Fitzgerald, 2007).

Coûteux pour les participants d'abord : la ressource spécifique à laquelle on doit avoir accès est un droit de contribution à l'évolution du projet (« logiciel flux»), pour que ses besoins soient pris en compte. Et pour pouvoir l'exercer, il faut contribuer, pour développer ses capacités à comprendre comment sont organisés le code et les différentes fonctionnalités que l'on veut faire évoluer (ce que Cohen et Levinthal, 1990, appellent les « capacités d'absorption »), mais surtout pour être accepté(e) par le projet et sa « communauté » (Dahlander et Wallin, 2006), et ainsi être en mesure d'exercer ce droit. Les équipes sont maintenues petites (Mockus *et al*., 2002), donc n'importe qui ne peut pas y avoir accès, et pour qu'une proposition de code soit acceptée, elle doit être considérée comme la meilleure (la mieux développée, la plus pertinente...).

Coûteux aussi pour les responsables du projet, qui doivent coordonner les différentes contributions, et arbitrer entre les attentes, parfois concurrentes des utilisateurs/contributeurs. Cela peut conduire à diverses formes d'opposition (Viseur et Charleux, 2021) allant jusqu'au *fork*, c'est-à-dire au départ d'une partie des contributeurs pour créer un projet concurrent (Viseur et Charleux, 2019 ; Viseur, 2012). Cette forme ultime d'expression du mécontentement est comparée à une « main invisible » contribuant, en rendant contestable le contrôle de l'évolution du projet, à la durabilité des projets libres (Nyman *et al.*, 2012).

Plus le projet est important sur le plan du volume de code, ou de participants, plus les contraintes techniques et organisationnelles sont fortes. Ainsi, il est essentiel que les projets soient organisés en composantes modulaires clairement définies, pour faciliter l'entrée de nouveaux contributeurs (Bessen, 2005), mais aussi du point de vue du génie logiciel et de l'organisation (Baldwin et Clark, 2003 ; MacCormack *et al.*, 2006), car il est très difficile pour des équipes trop grosses de travailler efficacement (Mockus *et al.*, 2002). Il faut aussi structurer les règles de contribution, les outils d'évaluation du code, de décision, ce qui passe par une certaine bureaucratisation de l'organisation (O'Mahony et Ferraro, 2007). Bref, il s'agit de construire une « architecture de participation » (MacCormack *et al.*, 2006).

Autrement dit (Markus, 2007), la gouvernance libre, c'est l'ensemble des moyens mis en œuvre pour l'orientation, le contrôle et la coordination d'agents économiques totalement ou partiellement autonomes – individus, organisations utilisatrices de la solution, entreprises utilisant la solution pour construire des offres commerciales – pour le compte d'un projet de développement libre.

Après avoir brièvement décrit les éléments principaux de cette gouvernance, nous expliquerons les liens entre participation aux projets libres et modèles d'affaires basés sur le logiciel libre (ce que nous appellerons, avec Fitzgerald, 2006, les stratégies *open source*), avant de conclure.

Quelques éléments de gouvernance libre

La gouvernance se structure généralement de manière progressive. De Laat (2007) identifie trois phases, plus ou moins successives. L'organisation initiale est souvent informelle, et liée aux règles fixées par la licence (par exemple GPL pour "General Public License") choisie par les initiateurs du projet, qui sont ceux ayant la propriété intellectuelle sur les premiers éléments de code. Le *leadership* est exercé par des développeurs sur la base de leurs performances (méritocratie). Dès lors que la taille du projet augmente, un ensemble d'outils formels et explicites doivent être mis en place : modularisation du code source, division des rôles, formalisation des procédures et du régime de décision. Lorsque le projet devient visible au-delà de ses utilisateurs-développeurs, ce *leadership* est de plus en plus sollicité (gestion de demandes de plus en plus nombreuses et hétérogènes d'utilisateurs et d'acteurs économiques), et parfois attaqué (par exemple par des acteurs qui essaient de récupérer du code sans respecter la licence) ; et il doit assurer sa pérennité sur les plans matériel, financier et légal. Cette phase d'institutionnalisation passe souvent par la création ou le rattachement à des structures de soutien telles que les fondations, souvent états-uniennes (Free Software Foundation, Apache Foundation, Linux Foundation pour les plus connues), mais qui existent aussi en Europe (la fondation Eclipse, maintenant domiciliée en Europe, ou l'association OW2). L'animation et la défense du projet peuvent être aussi confiées à un éditeur privé, à l'initiative des utilisateurs (cas du Department of Defense et d'AdaCore pour la gestion du langage Ada) ou à l'initiative d'un acteur, souvent à l'origine du projet, qui cherche à organiser, *via* une organisation libre, une interaction avec un club d'utilisateurs : discussion des besoins et de leur financement, voire co-développements avec certains utilisateurs (cas de la société française XWiki).

La gouvernance est, en résumé (Markus, 2007), un phénomène multidimensionnel, qui organise le contrôle du projet, *via* le contrôle de différents éléments : les droits de propriété (sur le code, mais aussi sur les marques et les licences que l'on accorde), la rédaction des objectifs du projet (par exemple charte et vision), la gestion de la communauté, le processus de développement logiciel (par exemple identification des besoins et affectation des tâches), la résolution de conflits et le changement de règles, et l'utilisation de l'information et des outils (par exemple modalités d'accès aux outils et aux répertoires de code source). Cette organisation, et la façon dont les participants acquièrent des droits (de contribution, de gestion de l'évolution) *via* différents systèmes (légaux, mais aussi organisationnels, techniques), est proche de la gestion des communs telle que décrite par Ostrom (1990). Jullien et Roudaut (2020) ont montré que l'on pouvait parler de commun de connaissance, en considérant que la ressource que ces communs gèrent est le logiciel flux (les contributions). Cela fait que, globalement, tous les projets de logiciel libre sont structurés de la même façon. On parle de modèle « en oignon » (Ye et Kishida, 2003 ; Crowston et Howison, 2005), avec un « cœur », réduit, de développeurs qui gèrent le projet, tant d'un point de vue stratégique (direction des développements) qu'organisationnel (validation des contributions), et différents niveaux de plus en plus périphériques de contributeurs (gestionnaires de sous-projet ou de *package*), contributeurs simples, utilisateurs simples (voir le Tableau 1 ci-dessous).

Dahlander et Magnusson (2008) ont montré comment le contrôle du projet, *via* des salariés-contributeurs, l'embauche des développeurs qui animent ledit projet, parfois la propriété sur le code ou sur la marque, sont au cœur de la stratégie des entreprises utilisatrices du libre, mais aussi, surtout, des entreprises *open source*. Non seulement ces entreprises renoncent à une partie du contrôle fourni par la propriété sur le code, et aux revenus afférents, mais elles investissent pour participer au projet, elles partagent pour contrôler. Nous avons expliqué, dans l'introduction et dans l'article « Le logiciel libre : gérer collectivement les évolutions d'une technologie » de ce numéro (pp. 70-76), ce paradoxe. Reste à montrer pourquoi ce contrôle leur permet de développer une offre commerciale, comment les ressources permettent de créer de la valeur pour leurs clients et surtout de capturer cette valeur.

	Propriétaire	**Administrateur, contributeur « cœur », « gestionnaire de communauté »**	**Responsable *package*, « co-développeur », « gestionnaires de projet »**	**Contributeur**	**Utilisateur**
Accès au stock de connaissance	X	X	X	X	X
Accès au dispositif de production	X	X	X	X	
Gestion des contributions et des contributeurs	X	X	X		
Gestion des évolutions du projet	X	X			
Gestion de l'aliénation[a] (marque, code, bases de test...)	X				

Tableau 1. Droits associés avec la position (le rôle) (Jullien et Roudaut, 2020), dans le cas des logiciels libres, ou organisation « en oignon » (Crowston et Howison, 2005)

[a] Aliénation au sens juridique du terme, capacité à contrôler le projet/le bien et à gérer les niveaux d'exclusion.

Participation aux projets libres et stratégies *open source*

Rappelons que le logiciel stock (la solution technologique à un instant donné) ne représente qu'une toute petite partie des besoins et des coûts totaux d'une solution logicielle (le TCO ou *total cost of ownership*). De même, la vente de licence n'est qu'une (petite) partie de l'activité économique de l'industrie informatique (Yost, 2017). On distingue deux classes d'activités marchandes chez les entreprises informatiques (Ethiraj *et al*., 2005) : les activités spécifiques aux clients et les activités de gestion de projet. Les premières s'attellent à la compréhension des besoins des clients et aident à la mise en œuvre de la conduite du changement, ou « besoins d'assistance » (Jullien et Viseur, 2021). Les activités de gestion de projet portent sur la gestion d'une technologie (par exemple création d'un logiciel interne ou activité d'édition logicielle) ou la mise en œuvre d'un logiciel (par exemple adaptation d'un logiciel, intégration dans le système d'information). Les tâches de gestion de projet sont très diverses, mais peuvent être séparées suivant qu'elles sont ou non visibles par le client (Aubry, 2015) : les capacités de gestion de projet client (liées à l'adaptation de logiciels existants par configuration), qui se rapprochent des tâches de spécifications fonctionnelles et répondent à des besoins d'adaptation, et les capacités de gestion de projet technique (liées à la maintenance sur le long terme de la technologie), répondant aux besoins d'assurance. Découlent donc de ce qui précède trois activités (Gabay, 2014) : les capacités spécifiques aux clients (MoA pour maître d'ouvrage), les capacités de gestion de projet client (MoA vers MoE, pour maître d'œuvre) et les capacités de gestion de projet technique (MoE).

Plus il y a d'utilisateurs et de contributions, plus les activités de suivi sont coûteuses, ce que la littérature sur l'industrie des services (Gadrey, 2003) a appelé « la fourniture de capacités techniques entretenues » (la capacité technique étant le logiciel flux). Les besoins des utilisateurs s'organisent donc autour de ce que Jullien et Zimmermann (2006) appellent les « trois A » : Assurance qualité, Adaptation (rapide) à leurs besoins, et Assistance à l'utilisation.

Ces services peuvent être rendus (et facturés) par un éditeur privé, quand celui-ci contrôle le développement du logiciel (par exemple AdaCore Technology, Odoo, XWiki). Dans ce cas, le modèle n'est pas en soi très différent du modèle traditionnel des producteurs d'outils informatiques, comme Oracle. Mais l'ouverture permet sans doute un meilleur retour de la part des utilisateurs, et donc une plus forte externalisation des coûts de développement, tout en assurant une évolution plus rapide du logiciel. Comme l'actif spécifique de ces entreprises repose sur la gestion de l'édition de logiciels, celles-ci doivent investir dans le logiciel qu'elles modifient, et l'implication des développeurs salariés de l'entreprise est souvent concentrée sur son produit (libre).

Mais on trouve aussi, surtout, des entreprises de services qui suivent des logiciels libres pour le compte de leurs clients qui n'ont pas les compétences ou le temps. Elles doivent garantir la qualité, permettre l'intégration dans le système d'information du client, etc. Elles jouent alors le rôle d'utilisateur-développeur et ont aussi besoin, pour évaluer ces composantes, de participer à leur production (Dahlander et Magnusson, 2008). Cette nécessité d'une évaluation et de contrôle va croissante avec l'importance du composant dans les solutions proposées par l'entreprise ou demandées par ses clients. On peut même formuler l'hypothèse que plus les clients sont qualifiés, plus l'entreprise doit maîtriser la technologie, en raison du niveau croissant de complexité des rétroactions et de la demande. Cela correspond à l'affirmation plus générale (Cohen et Levinthal, 1989 ; 1990) de la nécessité pour une entreprise de faire des efforts internes de R&D une condition préalable à l'absorption de technologie extérieure.

Autrement dit, à chaque position dans la gouvernance du projet correspond une capacité, difficilement imitable, et une proposition de valeur monétisable (le lecteur intéressé par le détail des différents modèles d'affaires *open source* pourra consulter Jullien et Viseur, 2021) : l'accès à la technologie permet de la tester et d'assister les utilisateurs sur son usage (maîtrise d'ouvrage, formation) ; la possibilité de faire accepter des modifications dans les versions officielles et de garantir des ajustements pour l'intégration dans la durée dans le système d'information (assurance, adaptation) ; la gestion de modules renforce cette position et donne un avantage concurrentiel sur certains métiers (lien entre maîtrise d'ouvrage et maîtrise d'œuvre, intégration, adaptation, assurance) ; la participation à l'évolution du projet, avec des développeurs « cœurs », est importante pour garantir la bonne marche du logiciel, accélérer la correction des *bugs* pour les métiers d'intégrateurs (assurance). Finalement, les capacités d'aliénation permettent surtout le contrôle d'actifs immatériels dynamiques : la marque (protégée par le droit de propriété intellectuelle, qui n'a de valeur que si le projet est reconnu et est dynamique), ou des bases de données (de tests d'erreurs par exemple, qu'il faut constamment mettre à jour), protégées par le secret, qui sont clefs dans les modèles basés sur l'assurance.

Conclusion

Ce modèle économique original, organisé pour favoriser la dynamique d'innovation, illustre, paradoxalement, l'importance du contrôle d'actifs pour assurer un avantage concurrentiel. Simplement, il repose sur un contrôle dynamique de la production intellectuelle et non plus, ou moins, sur un contrôle statique, à l'image d'autres industries créatives (Bach *et al*., 2010), et des stratégies d'*open innovation* (Pénin, 2011). On retrouve les « capacités dynamiques » chères à Teece *et al.* (1997), qui sont pour ces auteurs au cœur de l'économie numérique (Teece, 2018). Notons, pour finir, toujours avec Jullien et Viseur (2021), que quand la dynamique du projet s'épuise (tarissement du logiciel flux), les modèles d'affaires se rapprochent des modèles classiques (accès au logiciel stock, *via* la licence ou des systèmes comme le *cloud*), et d'autres ressources clefs que la participation au projet deviennent primordiales, laissant la place à d'autres acteurs.

Références bibliographiques

ÃGERFALK P. J. & FITZGERALD B. T. (2007), "Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy", *MIS Quarterly*, 32(2), pp. 385-409.

AUBRY C. (2015), *Scrum* : *Le guide pratique de la méthode agile la plus populaire*, InfoPro, Dunod.

BACH L., COHENDET P., PÉNIN J. & SIMON L. (2010), "Creative industries and the ipr dilemma between appropriation and creation: Some insights from the videogame and music industries", *Management international / International Management / Gestión Internacional*, 14(3), pp. 59-72.

BALDWIN C. Y. & CLARK K. B. (2003), "The architecture of cooperation: How code architecture mitigates free riding in the open source development model".

BESSEN J. (2005), "Open source software: Free provision of complex public goods", rapport technique.

COHEN W. M. & LEVINTHAL D. A. (1989), "Innovation and learning: The two faces of r&d", *Economic Journal*, 99, pp. 569-596.

COHEN W. M. & LEVINTHAL D. A. (1990), "Absorptive capacity, a new perspective of learning and innovation", *Administrative Science Quarterly*, 35, pp. 128-152.

CROWSTON K. & HOWISON J. (2005), "The social structure of free and open source software development", *First Monday*, 10(2).

DAHLANDER L. & MAGNUSSON M. G. (2008), "How do firms make use of open source communities?", *Long Range Planning*, 41, pp. 629-649.

DAHLANDER L. & WALLIN M. W. (2006), "A man on the inside: Unlocking communities as complementary assets", *Research Policy*, 35, pp. 1243-1259.

DE LAAT P. B. (2007), "Governance of open source software: State of the art", *Journal of Management & Governance*, 11(2), pp. 165-177.

ETHIRAJ S. K., KALE P., KRISHNAN M. S. & SINGH J. V. (2005), "Where do capabilities come from and how do they matter? A study in the software services industry", *Strategic Management Journal*, 26(1), pp. 25-45.

FELIN T. & ZENGER T. R. (2014), "Closed or open innovation? Problem solving and the governance choice", *Research policy*, 43(5), pp. 914-925.

FITZGERALD B. (2006), "The transformation of open source software", *MIS Quarterly*, 30(3), pp. 587-598.

FORAY D. & ZIMMERMANN J.-B. (2001), « L'économie du logiciel libre : organisation coopérative et incitation à l'innovation », *Revue économique*, 52, pp. 77-93.

GABEY J. (2014), *Maîtrise d'ouvrage des projets informatiques - 3e édition* : *Guide pour le chef de projet MOA*, Dunod.

GADREY J. (2003), *Socio-économie des services,* Paris, La Découverte, coll. « Repères ».

JULLIEN N. & ROUDAUT K. (2020), « Commun numérique de connaissance : définition et conditions d'existence », *Innovations*, 3(63), pp. 69-93.

JULLIEN N. & VISEUR R. (2021), « Les stratégies open-sources selon le paradigme des modèles économiques », *Systèmes d'information et management*, 26(3), pp. 67-103.

JULLIEN N. & ZIMMERMANN J.-B. (2006), "New approaches to intellectual property: From open software to knowledge based industrial activities", *in* LABORY S. & BIANCHI P. (éd.), *International Handbook on Industrial Policy*, Edward Elgar (EE), pp. 243-264.

LAKHANI K. & VON HIPPEL E. (2003), "How open source software works: Free user to user assistance", *Research Policy*, 32, pp. 923-943.

MACCORMACK A., RUSNAK J. & BALDWIN C. Y. (2006), "Exploring the structure of complex software designs: An empirical study of open source and proprietary code", *Management Science*, 52(7), pp. 1015-1030.

MARKUS M. L. (2007), "The governance of free/open source software projects: monolithic, multidimensional, or configurational?", *Journal of Management & Governance*, 11(2), pp. 151-163.

MOCKUS A., FIELDING R. T. & HERBSLEB J. D. (2002), "Two case studies of open source software development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3), pp. 309-346.

NYMAN L., MIKKONEN, T., LINDMAN J. & FOUGÈRE M. (2012), "Perspectives on code forking and sustainability in open source software", *in* HAMMOUDA I., LUNDELL B., MIKKONEN T. & SCACCHI W. (éd.), *IFIP International Conference on Open Source Systems*, Berlin, Heidelberg, Springer, pp. 274-279.

O'MAHONY S. & FERRARO F. (2007), "The emergence of governance in an open source community", *Academy of Management Journal*, 50, pp. 1059-1106.

OSTROM E. (1990), *Governing the Commons: The Evolution of Institutions for Collective Action*, Cambridge, Cambridge University Press.

PÉNIN J. (2011), "Open source innovation: Towards a generalization of the open source model beyond software", *Revue d'économie industrielle*, 136, pp. 65-88.

TEECE D., PISANO G. & SHUEN A. (1997), "Dynamic capabilities and strategic management", *Strategic Management Journal*, 18(7), pp. 509-533.

TEECE D. J. (2018), "Business models and dynamic capabilities", *Long Range Planning*, 51(1), pp. 40-49.

VISEUR R. (2012), "Forks impacts and motivations in free and open source projects", *International Journal of Advanced Computer Science and Applications*, 3(2), pp. 117-122.

VISEUR R. & CHARLEUX A. (2019), « Changement de gouvernance et communautés open source : le cas du logiciel Claroline », *Innovations*, (1), pp. 71-104.

VISEUR R. & CHARLEUX A. (2021), "Open source communities and forks: A rereading in the light of Albert Hirschman's writings", *Proceedings* of the 17th IFIP International Conference on Open Source Systems (OSS), pp. 59-67.

VON HIPPEL E. & VON KROGH G. (2003), "Open source software and the 'private-collective' innovation model: Issues for organization science", *Organization Science*, 14(2), pp. 209-223.

YE Y. & KISHIDA K. (2003), "Toward an understanding of the motivation of open source software developers", *Proceedings* of the 25th International conference on Software Engineering, IEEE, pp. 419-429.

YOST J. R. (2017), *Making IT Work: A History of the Computer Services Industry*, MIT Press.