Les stratégies open-sources selon le paradigme des modèles économiques

Nicolas Jullien, Robert Viseur

Dans Systèmes d'information & management 2021/3 Volume 26, pages 67 à 103
Éditions ESKA

ISSN 1260-4984
ISBN 9782747232401
DOI 10.3917/sim.213.0067

Les stratégies open-sources selon le paradigme des modèles économiques

Nicolas JULLIEN & Robert VISEUR***

* IMT Atlantique, LEGO, France
** Université de Mons, Belgique

RÉSUMÉ

De nombreuses entreprises utilisent des logiciels libres dans leurs propositions commerciales. Il existe de nombreuses études de cas sur leur modèle d'affaires. Cependant, il manque une analyse globale synthétisant et structurant les différents modèles open-sources. En nous appuyant sur le cadre des modèles économiques et la littérature en systèmes d'information, nous partons des besoins en offres libres pour caractériser et comparer des modèles économiques open-sources. Cela nous permet de discuter le lien entre captation de valeur et ressources clefs. Le modèle de l'open-source apparaît basé sur la gestion et le contrôle de la dynamique d'évolution des logiciels (« logiciel flux »), dans une industrie qui s'était habituée à vendre des produits (« logiciel stock »).

Mots-clés *: open-source, modèles économiques, stratégie, systèmes d'information.*

ABSTRACT

Many companies use free open-source software (FLOSS) in their business proposals. There are many case studies of open-source business models. However, there is no comprehensive analysis articulating these different open-source models. Based on the business model framework, and the IS literature, we start from the needs of FLOSS to analyze and compare open-source business models. This allows us to discuss the links between value capture and key resources. The open-source model appears to be based on the management and control of software evolution dynamics ("software-flux"), in an industry that has become accustomed to selling products ("software-stock").

Keywords: *open-source, business models, strategy, information systems.*

« In essence, a business model [is] a conceptual, rather than financial, model of a business. » (Teece, 2010)

1. INTRODUCTION

Le logiciel libre, système d'innovation initié par et pour ses utilisateurs-développeurs (von Hippel et von Krogh, 2003), a rapidement été adopté par des entreprises pour proposer des solutions commerciales appuyées sur un ou plusieurs logiciels libres, stratégies qualifiées d'*« open-sources »* (Fitzgerald, 2006)[1]. Il n'est donc pas étonnant que la recherche en management stratégique et en économie industrielle se soit intéressée à l'impact du logiciel libre sur l'industrie informatique (Jullien et Zimmermann, 2011b) et, plus généralement, sur les enseignements que l'on peut en tirer pour les chaînes de valeur (Pénin, 2011 pour une synthèse de ces réflexions).

La recherche en management des systèmes d'information s'est d'abord et principalement intéressée au fonctionnement de tels systèmes de développement : motivations des contributeurs (von Hippel et von Krogh, 2003 ; Baldwin et Clark, 2003 ; Carillo et Okoli, 2011 ; von Krogh *et al.*, 2012 ; Howison et Crowston, 2014) et intégration d'un modèle de développement ouvert dans les routines des entreprises (Hauge *et al.*, 2010, pour une revue de la littérature). Elle montre que les logiciels libres sont développés par des experts pour des experts, que la plupart sont des projets individuels, ou d'une équipe très réduite (Capiluppi *et al.*, 2003). Pour les quelques projets phares, l'organisation est très modulaire, avec une équipe « cœur » réduite qui contrôle l'évolution (Mockus *et al.*, 2002). Lorsque les entreprises souhaitent intégrer ces logiciels dans leur SI, plus les projets sont dynamiques, plus elles doivent s'impliquer dans les développements, et adapter leur organisation interne en conséquence (Ãgerfalk et Fitzgerald, 2007 ; Stol *et al.*, 2014). Elles doivent notamment implémenter des méthodes de développement agile, préexistantes mais que le logiciel libre a contribué à diffuser (Spinellis et Szyperski, 2004). Elles doivent aussi exposer au jugement extérieur les codes créés par leurs collaborateurs et développer une organisation coopérative, parfois orthogonal aux routines internes et donc coûteuse en matière de changement (Fitzgerald, 2006 ; Stol *et al.*, 2014).

Dans la littérature, il y a moins de travaux sur le lien entre position dans la chaîne de valeur et organisation : comment délivre-t-on de la valeur à un utilisateur, quelle organisation doit être mise en place pour cela, ou réside l'avantage concurrentiel du logiciel libre ? La théorie des modèles économiques du libre reste à faire. De nombreuses études existent et montrent la diversité des modèles open-sources, mais elles sont soit partielles, sur la présentation d'une liste de cas d'entreprises (West, 2003 ; Bonaccorsi et Rossi, 2003 ; Dahlander et Wallin, 2006 ; Charleux et Mione, 2018), sans que l'on comprenne bien quels sont les critères qui président à telle ou telle stratégie, soit restreintes à un type d'acteur de la chaîne de valeur informatique, les éditeurs par exemple. Il y a dix ans, Lecocq *et al.* (2010, paragraphe 22) soulignaient que l'étape de théorisation était encore balbutiante pour les modèles économiques (lien entre les différents éléments du modèle, et avec les grandes théories de la stratégie et de l'économie). Cela reste vrai, au moins pour ce qui concerne l'open-source, et c'est ce à quoi nous proposons de contribuer ici.

L'article est organisé de la façon suivante. La revue la littérature des modèles

[1] Nous y reviendrons dans la revue de la littérature, mais suivant Fitzgerald (2006), nous utiliserons le terme « logiciel libre » pour parler des dimensions technico-organisationnelle (développement ouvert et collectif d'un logiciel) et produit (le logiciel), et l'adjectif « open-source » pour parler des dimensions et stratégies marchandes du logiciel libre.

économiques open-sources est structurée suivant la présentation du concept de modèle économique de Massa *et al.* (2017). Cela nous permet de définir les concepts clefs à étudier (partie 2), et nous amène à séparer l'analyse en deux parties. D'abord, nous nous intéressons à la création de valeur (partie 3). Nous montrons que les besoins informatiques n'ont pas beaucoup évolué depuis la création de l'industrie, malgré les évolutions technologiques et commerciales. Le logiciel libre est une des évolutions et pas une révolution : la création de valeur du logiciel libre est le fait d'un écosystème d'acteurs, comme souvent en informatique. Cependant, une spécificité du logiciel libre est qu'il n'y a pas toujours une entreprise en son centre. Ce type d'écosystème semble particulièrement adapté quand il y a des besoins d'évolution et de maintenance logicielles (ce que nous appellerons le logiciel « flux »), pour des raisons de sécurité, d'évolutions technologiques ou fonctionnelles.

La réponse à ces besoins et les actifs et activités que des entreprises doivent contrôler pour pouvoir en extraire de la valeur sont étudiés ensuite (partie 4). Là encore, nous partons d'une analyse générale pour définir et expliquer les *modèles d'affaires open-sources*. Nous soulignerons, retrouvant là l'analyse de Demil et Lecocq (2010), que la captation de valeur s'appuie sur une articulation dynamique, difficilement imitable, entre proposition de valeur et ressources.

En résumé, la création de valeur principale du logiciel libre est de gérer le « flux » logiciel, plutôt que le « stock » et c'est en participant à cette gestion que les entreprises open-sources arrivent à monétiser leurs services. En conclusion, nous discutons ce que ces résultats apportent aux praticiens des entreprises open-sources et à la théorie sur les modèles économiques.

2. REVUE DE LITTÉRATURE : MODÈLES ÉCONOMIQUES ET LOGICIEL LIBRE

2.1. Du logiciel libre à l'industrie open-source

À l'origine du mouvement libre, on trouve des professionnels de l'informatique dont le métier se complexifiait avec des logiciels de plus en plus difficiles à adapter, et qui revendiquaient plus de contrôle sur leur outil de travail[2].

Des projets de développement de logiciels libres se sont alors développés au sein de communautés, avec une organisation complexe et hiérarchisée autour de rôles bien définis donnant plus ou moins de droits sur le projet. Le modèle « en oignon » (Crowston et Howison, 2005 ; Ye et Kishida, 2003) présenté en tableau 1 en propose une synthèse que nous utiliserons.

Dans le paradigme libre, la principale question de recherche reposait sur les raisons des développeurs à participer à de tels projets (Lerner et Tirole, 2002). La plupart étaient des professionnels de l'informatique (Lakhani et Wolf, 2005 ; Meissonier *et al.*, 2010) qui contribuaient, car cela facilitait leur travail (Dahlander et Magnusson, 2005 ; Shah, 2006). En miroir, cela a rapidement posé la question de pourquoi les entreprises laissaient leurs salariés participer à de tels projets, pourquoi elles investissaient dans de tels projets. Ainsi, on est passé de l'analyse d'une organisation de collaboration virtuelle entre utilisateurs (ce que Fitzgerald, 2006 appelle « l'époque du logiciel libre ») à des questions d'organisation industrielle et de retour sur investissement (« l'époque de l'open-source », qui est celle qui va nous

[2] Sur la place du logiciel libre dans l'histoire de l'informatique, on pourra consulter Jullien et Zimmermann (2002), et bien sûr le manifeste de création de la notion de logiciel libre et de la Free Software Foundation.

	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				

[a]Aliénation au sens juridique du terme, c'est-à-dire le transfert de droit de propriété intellectuelle. Il s'agit ici de la licence du logiciel, mais aussi de contrats liés à la marque, notamment.

Tableau 1 (adapté de Jullien et Roudaut, 2020) : Faisceaux de droits associés avec la position (le rôle) dans le cas des modèles de développement de logiciels libres, ou organisation en oignon (Crowston et Howison, 2005).

intéresser) : en quoi, et quand, une stratégie « open-source », basée sur l'utilisation d'un ou plusieurs logiciels libres, crée plus de valeur qu'une stratégie classique, une valeur différente, ou un avantage concurrentiel déterminant, et sur quel marché ? Comment les entreprises de l'open-source captent une partie de cette valeur ? Il s'agit donc d'étudier des modèles économiques.

Dahlander et Magnusson (2008) ont montré comment le contrôle de la « communauté », via des salariés-contributeurs, l'embauche des développeurs qui animent la communauté, parfois la propriété sur le code ou sur la marque, est au cœur de la stratégie des entreprises utilisatrices du libre, mais aussi, surtout, des entreprises open-sources. Si, dans la suite de von Hippel ils ont constaté que c'était une manière pour les utilisateurs de logiciel (individus ou entreprises) de développer collectivement les ressources complémentaires dont ils avaient besoin (von Hippel et von Krogh, 2003 ; Lakhani et von Hippel, 2003), se pose la question du retour sur investissement. Habituées à vendre des licences d'utilisation de leurs logiciels (leur propriété intellectuelle), elles semblent détruire leur modèle d'affaires en adoptant des modèles de licence ouverte. Une théorie des modèles open-sources doit expliquer ce lien, à l'image de ce qu'ont commencé à proposer ces auteurs, ou Jullien et Zimmermann (2009, 2011a).

2.2. L'originalité de l'approche par les modèles économiques

Le cadre analytique du modèle économique complète les cadres précédents (Ressource Based View, par exemple), en prenant mieux en compte trois aspects (Massa *et al.,* 2017) : création et captation de la valeur sont deux choses différentes ; la valeur est souvent créée par un réseau ou un écosystème d'acteurs ; ces acteurs construisent des relations en situation d'asymétrie d'information. Ces trois éléments apparaissent au cœur du fonctionnement de l'industrie du logiciel, et donc de l'open-source.

Que la création et la captation de la valeur soient deux choses différentes (Lepak *et al.*, 2007), et la seconde ne va pas de soi, particulièrement dans le numérique « – même en l'absence d'externalités marchandes – semble tout sauf trivial »*[3] (Massa *et al.*, 2017, p. 91). Dans l'analyse de von Hippel, création et captation de valeur sont liées : les utilisateurs participent au développement des logiciels libres qu'ils utilisent, car cela leur permet de les adapter à leurs besoins, tout en profitant des contributions des autres. Mais si des entreprises sont prêtes à payer pour pouvoir utiliser des logiciels libres, cela veut dire soit qu'elles ont besoin d'aide pour accéder à ces logiciels, soit qu'elles payent d'autres choses, des valeurs complémentaires. Dans tous les cas, création libre et captation open-source ne sont pas équivalentes et il faut mieux comprendre les différents besoins auxquels le logiciel, libre ou non, répond.

Concernant le deuxième point souligné par Massa *et al.* (2017, p. 91), la création de la valeur est le fait d'un réseau (Lepak *et al.*, 2007, p. 513), ou d'un « écosystème » (Adner et Kapoor, 2010) et non pas le résultat de l'action d'un seul acteur économique, notamment dans le numérique (Amit et Zott, 2001). Ainsi l'industrie informatique est une industrie où la concurrence se fait entre plates-formes technologiques regroupant un écosystème d'entreprises, plutôt qu'entre entreprises (Campbell-Kelly et Garcia-Swartz, 2007, 2015), même si ces plates-formes sont souvent

[3] Les éléments suivis d'un astérisque sont des traductions. Sauf mention contraire, il s'agit de nos traductions.

contrôlées par un ou deux acteurs (Gawer et Cusumano, 2002) : Windows contre Mac et IBM contre Intel pour les ordinateurs personnels, Windows contre Unix dans les serveurs (Gueguen et Torres, 2004), Android contre iOS et Amazon Web Service contre Azure ou Google Cloud (Fautrero et Gueguen, 2012 ; Marston *et al.*, 2011).

La notion que l'utilité créée par un agent économique est supérieure au profit qu'il en retire (la valeur qu'il capte) est une des bases de la micro-économie et de l'économie industrielle (Tirole, 1989) ou de l'analyse stratégique classique (Porter *et al.*, 1996). La théorie des modèles économiques souligne que ce n'est pas forcément celui qui crée la valeur qui en retire un revenu. Elle explique aussi qu'un bien produit sans rémunération peut générer, en complément d'autres biens ou services, une valeur et, qu'une partie de cette valeur est captable par une entreprise (Massa *et al.*, 2017). Pour comprendre l'open-source, il faut donc aller au-delà du fait qu'un logiciel libre est gratuit. Charleux et Mione (2018) ont ainsi montré la diversité des écosystèmes soutenant la production d'un logiciel libre : les acteurs privés qui s'appuient sur ces logiciels pour développer des activités commerciales ne les contrôlent pas toujours (parfois ce sont des fondations, parfois un réseau plus lâche, une « communauté », sans qu'il existe une corrélation flagrante entre un type de contrôle et l'existence d'acteurs open-sources)[4]. D'autre part, si le logiciel est distribué sous une licence libre, et que des acteurs arrivent à monétiser cette distribution, cela veut dire que l'écosystème crée de la valeur, mais aussi que le code source n'est pas le seul élément que l'utilisateur recherche. Est-ce une nouvelle valeur, créée et captée par une entreprise, ou est-ce que certaines entreprises arrivent à capter une partie de la valeur créée par l'écosystème, et à quelle(s) condition(s) ? Ces questions font écho au travail de Chesbrough *et al.* (2018) en innovation ouverte, l'open-source étant considéré comme un cas extrême d'innovation ouverte (Jullien et Pénin, 2014). Cependant, ces modalités de captation restent largement à explorer.

Le troisième point souligné par Massa *et al.* (2017, p. 92) permet d'avancer sur ce terrain. Les consommateurs de biens sont dans des situations d'information imparfaite sur la qualité des fournisseurs et des produits ou services proposés, et cela explique en partie la construction des relations d'échange. Dans l'industrie informatique, très technique (Cusumano, 2004), les compétences des clients, leur capacité à traduire des besoins métiers (fonctionnels) en spécifications (logicielles) sont au moins aussi importantes que les capacités des fournisseurs de solutions (De Bandt, 1998 ; Ross *et al.*, 1996 ; Lee et Lee, 2004). La littérature sur les ressources (Cohen et Levinthal, 1990 ; Van Den Bosch *et al.*, 1999 ; Zahra et George, 2002) explicite que, pour absorber une ressource, il faut avoir des capacités d'absorption, qui dépendent, entre autres, des compétences de la firme (et de ses employés), ou de sa capacité à accéder à des intermédiaires pour pallier ce manque de compétence interne (Spithoven *et al.*, 2010 ; Lee et Lee, 2004). Jullien et Zimmermann (2009) ont proposé de classer les consommateurs de logiciel en trois niveaux selon leur capacité à développer un logiciel et à interagir avec les fournisseurs de solutions. Le premier niveau concerne la capacité à traduire, programmer les besoins

[4] Ou tout le moins, on trouve des logiciels libres avec un fort écosystème commercial – des propositions open-sources selon notre terminologie – comme des logiciels sans écosystème commercial dans chaque catégorie. La recherche en génie informatique s'est d'ailleurs emparée du terme « écosystème » pour souligner qu'il faut analyser l'ensemble des acteurs d'un projet logiciel et non le projet de développement seul pour évaluer son efficacité, sa santé, etc. Franco-Bedoya *et al.* (2017) proposent une revue récente de l'usage et de l'acceptation de ce terme dans ce champ de recherche.

en logiciel, présente chez les utilisateurs-innovateurs décrits par von Hippel (1998) et par conséquent baptisés utilisateurs « VH ». Le second niveau porte sur la capacité à spécifier les besoins en exigences fonctionnelles, présente chez les utilisateurs avancés et qu'ils ont baptisés « utilisateurs (KM) », en référence à Kogut et Metiu (2001) qui ont souligné l'importance des utilisateurs capables d'exprimer leur demande ou de relever des erreurs pour les projets libres. Le troisième niveau, caractérisé par l'absence de l'une et l'autre capacité, est celui des utilisateurs « naïfs », au sens inexpérimentés, profanes, et qui prennent les solutions telles qu'elles sont.

Ce ne sont pas forcément les utilisateurs les moins compétents qui sous-traitent le plus (De Bandt, 1998) : s'ils peuvent faire eux-mêmes, ils peuvent aussi être plus disposés à sous-traiter, car ils connaissent l'importance du besoin et sont aussi plus à même d'évaluer la qualité du prestataire. Les consommateurs VH ont donc accès à un portefeuille stratégique de solutions, entre faire et faire faire, plus vaste que les utilisateurs KM, qui doivent faire faire plus souvent tout en étant capables de définir, spécifier leurs besoins et évaluer les solutions, et que les naïfs, qui ont des capacités limitées d'exploration.

En résumé, pour comprendre ce que les entreprises open-sources proposent et facturent, il faut s'intéresser à l'écosystème de production libre sur lequel elles se basent, à la valeur créée pour les utilisateurs et à leur capacité à participer à cette création de valeur, mais aussi à l'évaluer. Dans ce contexte, la typologie de Massa *et al.* (2017), et les questions que soulèvent Foss et Saebi (2017) sur les bases théoriques du concept et sur la façon d'évaluer le caractère innovant des modèles, vont nous permettre de discuter les travaux existants.

2.3. Les modèles économiques open-sources

La plupart des travaux sur les modèles économiques open-sources se concentre soit sur une ou des études de cas, souvent inspirées des modèles classiques de l'industrie informatique, soit sur un aspect particulier du modèle économique (gestion de la propriété intellectuelle, par exemple).

2.3.1. Les études de cas

Les études de cas s'appuient sur une distinction classique dans l'industrie entre deux grands modèles, définis a priori : l'éditeur et la société de service (les sociétés de services en ingénierie informatique, SSII, aujourd'hui appelées ESN, pour entreprises de services du numérique). Leur traduction dans le monde du logiciel libre étant « éditeur open source » ou « libre » et les « sociétés de service en logiciel libre », ou « SSLL », et aujourd'hui ENL pour entreprises du numérique libre. Nous reviendrons dans la partie 3.2 sur les limites d'une telle distinction dans l'industrie informatique en général.

Shahrivar *et al.* (2018) proposent une revue de la littérature des modèles éditeurs open-sources. Elle permet une bonne compréhension des différentes activités et offres commerciales. Cependant, l'ensemble reste très descriptif et il est difficile d'en tirer des éléments qui permettraient de construire une typologie de ces modèles. Cette analyse est aussi limitée aux éditeurs privés open-sources quand Charleux et Mione (2018) ont montré l'importance d'aller au-delà d'une focale sur l'entreprise éditrice de logiciel pour comprendre le fonctionnement des modèles économiques de production du logiciel libre. Si le travail de ces dernières propose une première typologie des différents modèles d'édition, il doit être étendu à l'écosystème global pour comprendre son modèle économique et l'intérêt des

différents modèles d'éditeur. Pour aller plus loin, il serait notamment intéressant de distinguer les activités de la partie édition (choisir les fonctionnalités, gérer les versions), de celles de la partie distribution (donner accès à ces versions, garantir leur bon fonctionnement).

Les sociétés de service ont été moins étudiées, et souvent intégrées dans des études plus générales sur les entreprises open-sources, autour de la question du lien produit/service (Lisein *et al.,* 2009 ; Mouakhar et Tellier, 2013 ; Mouakhar et Benkeltoum, 2020). Ainsi Mouakhar et Tellier (2013) analysent, sur 71 cas, les liens entre positionnement stratégique et discours des entreprises sur ce qu'est l'open-source, entre un outil technique, une facilité d'accès à un code, jusqu'à une volonté de rendre de la liberté, du contrôle au client, dans le respect des « valeurs » originelles du logiciel libre. Ils montrent que ce ne sont pas forcément les entreprises qui ont le discours le plus libre qui utilisent le moins les systèmes de double licence. On retrouve les résultats de Dahlander et Magnusson (2005, 2008) sur les comportements très variables d'implication dans les projets, sans que les liens entre discours et implication, ou entre implication et performance, soient très clairs... Pourtant la littérature pointe une prédominance des services dans la captation de valeur par les entreprises (Fitzgerald, 2006, p. 593).

Avec ces études on a une bonne description des différentes pratiques de captation de valeur des entreprises, synthétisée par Popp (2019) et vulgarisée, par exemple, dans Wikipédia[5]. Cependant, les déterminants stratégiques de ces modèles de création de valeur, et les liens avec les autres éléments du modèle économique ne sont pas toujours très clairs : ainsi, on ne sait pas quels services fournissent les sociétés de service, et s'ils sont différents de ceux fournis autour d'un logiciel non libre. Certains travaux ont, en parallèle, cherché à approfondir certains éléments propres à l'open-source, et notamment la licence et l'investissement dans les projets libres.

2.3.2. Les éléments du modèle d'affaires.

L'aspect le plus étudié, à nouveau via des études de cas, est la gestion de la propriété intellectuelle, dans une acceptation large : comment les entreprises valorisent-elles leur propriété intellectuelle en hybridant les modèles classiques, via des stratégies de double licence (Valimaki, 2003) ou des modèles hybrides mêlant offre libre et propriétaire (Bonaccorsi *et al.*, 2006) ? Vendome *et al.* (2017), dans leur analyse de l'écosystème autour de Java, montrent finalement assez peu d'évolution de licence une fois celle-ci choisie, et peu de réflexion stratégique derrière le choix. Tout au plus, le contrôle industriel du projet peut pousser à des licences moins contaminantes, pour permettre l'appropriation du code par sa fermeture.

D'autres auteurs ont discuté les stratégies de contribution aux projets libres. West et Gallagher (2006) distinguent quatre grandes stratégies d'implication des entreprises : R&D/produits co-développés, à la manière d'un bien public industriel (Romer, 1993 ; Jullien et Zimmermann 2011b), développement, externalisation (« spinout »), et vente de compléments (services ou licence de produits propriétaires). Ils n'expliquent pas pourquoi ces stratégies sont si courantes dans le logiciel, et moins ailleurs, ni comment la vente de compléments peut fonctionner. Pour Mouakhar et Benkeltoum (2020), ce sont les intentions des dirigeants qui

[5] Les articles en anglais «Business models for open-source software» et en français « Modèles économiques des logiciels open source » sont tous deux très détaillés, et assez proches, même si l'anglais est plus fourni.

semblent expliquer l'implication dans les projets, et les bénéfices de ces stratégies restent à évaluer sur un temps long... Pourtant, ces quatre catégories permettent de structurer la réflexion, comme le proposent Jullien et Zimmermann (2009), autour du lien entre implication et compétence des utilisateurs : plus les utilisateurs d'un logiciel libre sont compétents, plus il y aura d'implication dans les développements libres de la part des entreprises open-sources.

Une des critiques de l'approche par les études de cas (Massa *et al.*, 2017 ; Foss et Saebi, 2017) reste la perception d'une accumulation sans construction théorique permettant de classer, ou d'évaluer ces modèles d'affaires, les conditions dans lesquelles ils peuvent être mis en œuvre, ou tout simplement pourquoi les entreprises les ont choisis. Les éléments du modèle économique à étudier pour ce faire sont pourtant bien circonscrits dans la littérature (Baden-Fuller et Mangematin, 2013 ; Maucuer et Renaud, 2019) : qui sont les utilisateurs, quelle est la valeur proposée, comment les entreprises captent (monétisent) une partie de la valeur créée, et quelle organisation (ressources clefs, activités clefs) doivent-elles contrôler pour cela (Teece, 2010 ; Foss et Saebi, 2017) ? C'est cette structure que nous allons utiliser pour analyser les modèles open-sources : création (partie suivante), puis captation et ressources et activités clefs dans la partie 4.

3. LA CRÉATION DE LA VALEUR... PAR DES ÉCOSYSTÈMES

S'intéresser aux utilisateurs du logiciel, c'est s'intéresser à la complexité des usages de l'informatique, puisque les logiciels ne peuvent fonctionner sans la composante matérielle. Si les auteurs s'accordent pour dire que les usages et les solutions développées en réponse se sont complexifiés (Campbell-Kelly et Garcia-Swartz, 2015), les dimensions traitées par les solutions informatiques, basées sur du traitement d'information, restent stables, autour de tâches d'automatisation, de communication, et de modélisation, telles que résumées par Horn (2004, p. 11) par exemple. Ces dimensions, et les besoins auxquels elles répondent, sont exprimées par différents acteurs, utilisateurs, services informatiques parfois. Dans notre analyse, nous nous concentrerons sur les utilisateurs entreprises, même si ce n'est pas l'ensemble des utilisateurs de logiciels libres, car c'est le marché de l'open-source[6]. Se référer aux usages et aux technologies dans la théorie des modèles économiques, c'est étudier les ressources (Grant, 1991 ; Conner, 19 91 ; Kogut et Zander, 1992) et le fait que ces ressources soient plus ou moins stratégiques, ou clefs, pour construire un modèle économique (Teece, 2010). Quelles sont les dimensions qui permettent de définir une typologie des ressources, et ultimement, une segmentation des clients (des utilisateurs de prestations payantes, assurant donc un flux de revenus à des entreprises) ? On parle bien sûr des ressources de l'utilisateur du ou des logiciels, et non des ressources des prestataires que nous étudierons plus tard.

3.1. Le logiciel, une ressource

Il y a deux dimensions stratégiques principales à la ressource logicielle (Cusumano, 2004 ; Horn, 2004) : le degré d'adaptation ou de personnalisation de la fonctionnalité demandée et son évolutivité. Selon l'importance stratégique de la ressource logicielle en tant que produit, ou stock (la ressource à l'acquisition est-elle plus

[6] On consultera, pour s'en convaincre, l'étude réalisée pour le Syntec Numérique et le Pôle Systematic Paris-Region par Teknowlogy Group (anciennement PAC) : https://cnll.fr/news/etude-open-source-2019/.

ou moins spécifique ?), mais aussi en tant que flux (doit-on prévoir aussi une évolution dans le temps de celle-ci ?), les critères de choix et d'évaluation des solutions ne seront pas les mêmes. C'est ce que la littérature et la pratique appellent « coûts totaux de possession » (ou « TCO »). Selon Shaikh et Cornford (2011), le TCO du logiciel pour l'ensemble de son cycle de vie est la combinaison de 5 types de coûts, qui sont liés à 5 phases : (1) le coût d'exploration (définition du besoin, recherche, évaluation et POC), (2) le coût d'acquisition (prix de la licence, adaptation aux besoins et intégration technologique), (3) le coût d'intégration (dans les usages ; migration, formation et processus), (4) le coût d'usage (support interne / externe, maintenance et, notamment, le coût de la panne (Walterbusch *et al.*, 2013), mises à jour techniques et fonctionnelles, passage à l'échelle) et (5) le coût de retrait ou de sortie (lié aux technologies et aux ressources humaines). Plus le projet ou la fonctionnalité portent sur une ressource stratégique de l'entreprise, qui doit durer dans le temps, plus la dimension dynamique a de l'importance (Arrègle, 1996 ; plutôt les phases 4 et 5, donc). Plus il y a d'adaptation du logiciel au besoin de l'entreprise, ou de l'entreprise au logiciel, plus les coûts de changement augmentent (plutôt les phases 2 et 3 du TCO). On retrouve ainsi les capacités dynamiques définies par Teece *et al.* (1997), au cœur des modèles économiques du numérique (Teece, 2018). L'utilisateur (entreprise) sera prêt à investir plus ou moins de temps et d'argent selon l'importance stratégique de l'actif, ce qu'illustre le tableau 3, en annexe.

Besoins de personnalisation et de contrôle dans le temps (besoin dynamique) mettent en lumière quatre grands modèles stratégiques de besoins. Si la ressource est perçue comme peu évolutive et peu stratégique, c'est une « commodité » qu'on cherchera à acquérir au meilleur coût (Prix). Si elle demande à être fortement adaptée aux besoins de l'entreprise, mais qu'une fois adaptée les évolutions sont perçues comme limitées, on recherchera une solution adaptable au meilleur coût, ce que nous appelons « Lean », et que Horn (2004) appelle « sur-mesure de masse ». La ressource peut avoir une dimension dynamique importante, du fait des évolutions technologiques par exemple, mais aussi des évolutions issues des besoins internes. Si le besoin n'est pas trop spécifique, on cherchera une solution basée sur des standards ouverts, permettant de garantir dans le temps les compatibilités technologiques, et échapper au maximum au lock-in (« Lock-out »). Si le besoin est en même temps spécifique et dynamique dans le temps, la solution devra être adaptable et évolutive. L'utilisateur vérifiera qu'il pourra contrôler celle-ci (solution « Contrôle »).

Cela veut aussi dire que lorsqu'on parle d'une solution logicielle, on parle, le plus souvent, d'un code logiciel, souvent vu comme un produit, complété par des activités de service pour choisir le produit, l'adapter et l'intégrer dans l'entreprise et dans son système d'information.

3.2. Création de valeur : articuler produit et service dans le temps

Une solution informatique est un mix de produit et de service (Horn, 2004 ; Cusumano, 2004). Avec l'arrivée des progiciels dans les années 1980, la partie produit est devenue visible, mais la partie service reste majoritaire en valeur (Campbell-Kelly et Garcia-Swartz, 2015), surtout chez les clients professionnels, notamment les grandes organisations (Yost, 2017). Les propositions de valeur associées à chaque segment de client se distinguent alors sur la composition du mix : une base « produit » plus ou moins centrale, car permettant des économies d'échelle sur la partie standard du besoin, et des services d'adaptation et de maintien de la solution en condition opérationnelle.

Ainsi, et pour illustrer notre propos, dans l'industrie des progiciels de gestion intégrés (PGI, ERP en anglais), certains consultants (ou entreprises) seront spécialisés dans une branche d'activité qu'ils sauront traduire en besoins fonctionnels, quand d'autres auront une connaissance fine de l'adaptation, du paramétrage par exemple, de l'outil pour répondre aux besoins fonctionnels. Ces compétences peuvent être intégrées en partie dans l'entreprise, et le fait qu'elles soient en partie maîtrisées facilite l'implémentation du PGI. Lee et Lee (2004) parlent des « actifs technologiques informationnels »* de l'entreprise.

Dans le modèle classique, cet écosystème est organisé autour d'un éditeur-distributeur, dont le métier est de « gérer une technologie (sur le long terme) »* (Cusumano, 2004, p. 4), et de distribuer ce produit, et pas seulement à fournir un « produit à succès unique »* (*ibid.*). Cependant, d'autres compétences sont aussi nécessaires pour que ce produit soit intégré par le client, quand c'est une ressource dynamique ou spécifique. Comme Lee et Lee (2004), nous parlerons des *activités d'assistance*, plus tournées vers le métier (de l'utilisateur) et faisant l'aller-retour avec l'outil (aide à la définition des besoins, spécification fonctionnelle, aide à l'usage, formation), et des activités liées à l'outil technique lui-même, ou *activités d'adaptation* (spécifications techniques, mise en œuvre, développement à façon), à côté des *activités d'assurance* de bon fonctionnement de la solution choisie à court et à moyen terme.

En synthèse (figure 1), nous soulignerons deux points. Premièrement, la valeur logicielle est toujours une construction

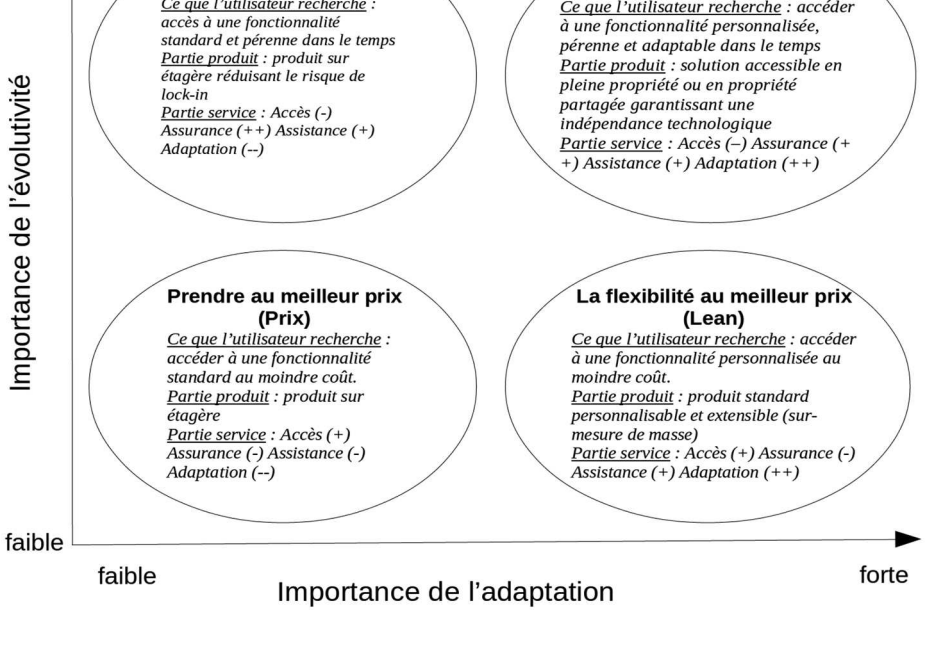


Figure 1 : Segmentation clients par rapport aux caractéristiques de leur besoin en terme de solution logicielle et mix produit-service.

de plusieurs éléments, une solution (le logiciel qui est développé, et le droit d'accès à cette solution) et des services qui permettent de mettre en œuvre la solution chez l'utilisateur ou le client (des services d'adaptation, d'assistance, d'assurance). Deuxièmement, la composition de ce bouquet dépend aussi des compétences des clients, dans une industrie qui reste une industrie de technologie.

Il faut alors comprendre quel(s) avantage(s) un utilisateur peut trouver à utiliser une solution libre (logiciel libre éventuellement complété par des services commerciaux ou produits en interne).

3.3. Avantages et inconvénients du logiciel libre

3.3.1. Les analyses génériques du logiciel libre

Les études sur les qualités (et les défauts) des logiciels libres s'appuient sur les éléments de création de valeur des solutions logicielles (Money *et al.*, 2012) : sécurité, facilité d'évolution, maintenabilité, testabilité, compréhensibilité, interopérabilité, en plus du prix. D'après ces auteurs, l'ouverture du code facilite la prise en compte des retours utilisateurs (la correction de bugs, par exemple). Franke et von Hippel (2003) montrent aussi comment la structuration d'un projet libre (le serveur HTTP Apache) construit une « boîte à outils d'innovation »* permettant à des utilisateurs ayant des compétences variées d'adapter le produit à leurs besoins. Le TCO des logiciels libres serait donc plus faible, à cause de la disparition des frais de licence et d'un meilleur respect des standards qui facilite l'interopérabilité et l'adaptation, donc l'intégration dans le système d'information (Almeida *et al.*, 2011).

Cependant, l'implication des utilisateurs est coûteuse et réservée aux utilisateurs les plus compétents. Les entreprises n'anticipent pas toujours les coûts d'intégration de logiciels libres (Morgan et Finnegan, 2014), qui sont pourtant classiques : « le coût du support et des services surpasse les économies initiales / Mauvaise documentation : difficulté à maintenir et à étendre les projets avec une documentation médiocre / Problèmes de compatibilité avec la technologie, les compétences et les tâches actuelles : toutes les parties du système doivent être compatibles. Il est souvent nécessaire de passer des applications logicielles habituelles à des applications offrant une plus grande compatibilité avec les logiciels libres. Par conséquent, il est nécessaire d'investir beaucoup de temps et d'argent dans le développement et la formation du personnel »* (*ibid.*, Table 3, p. 232). Dès lors, Jullien et Zimmermann (2006) en concluent que les logiciels libres ne sont pas pour des utilisateurs naïfs, même si, à cause des effets de rendements croissants d'adoption, les projets libres peuvent avoir intérêt à étendre leur base d'utilisateurs. Le risque étant que les besoins définis par ces primo-adopteurs soient très éloignés des besoins du marché de masse (Goldenberg *et al.*, 2002).

Au-delà des fonctionnalités, la santé du projet libre doit être évaluée avant d'adopter le produit. Plusieurs travaux ont proposé des métriques pour ce faire. Cependant, il n'existe pas, à ce jour, un modèle universel[7].

[7] Open Business Readiness Rating (OpenBRR), Qualification and Selection of Open Source software (QSOS, publiée par Atos Origin), font partie des premières tentatives de création d'une méthodologie standardisée d'évaluation des projets libres (Petrinja *et al.*, 2010). Ces méthodologies ont été complétées par des recherches visant à automatiser les mesures de qualité, incluant le logiciel et son écosystème, comme le QualiPSo OpenSource Maturity Model (Izquierdo-Cortazar *et al.*, 2010). Ce programme est toujours un travail en cours, comme en témoigne le projet « Community Health Analytics open-source Software » (Chaoss) soutenu par la Linux Foundation.

Ainsi, si Jansen (2014) propose une synthèse de la littérature et un programme pour construire de telles métriques, elle ne s'intéresse qu'aux projets déjà structurés. Pourtant, Crowston et Scozzi (2002) ont souligné très tôt que ces paramètres dépendaient de la maturité du projet, en matière de qualité et de quantité de code, bien sûr, mais aussi de taille de l'équipe de développement, ou du nombre des utilisateurs (Cheruy *et al.*, 2017).

On ne peut pas parler d'une proposition de valeur du logiciel libre, mais des propositions de valeur qui dépendent du nombre d'utilisateurs, de leur compétence et donc de leur investissement dans le projet (nombre de contributeurs), et dans une moindre mesure, de la licence utilisée (Santos *et al.*, 2013). Elles dépendent aussi de la maturité du projet (Crowston et Scozzi, 2002 ; Cheruy *et al.*, 2017).

3.3.2. Le logiciel libre en tant que création d'un écosystème de développement d'une solution

Ces auteurs, comme Koch (2011) et Jullien *et al.* (2019), distinguent trois phases dans un projet libre, qui font écho aux phases de développement d'un logiciel en général (Lehman, 1980), mais aussi à la question de l'action collective (Marwell et Oliver, 1993) sur la partie projet, et à la diffusion des technologies de Rogers (1983) sur la partie adoption. Elles permettent de préciser les types de valeur créés et les utilisateurs intéressés par la solution à chaque phase.

- ***Phase 1 : La phase de l'utilisateur-innovateur***

Le cœur de l'incitation à initier un projet libre est le principe de l'utilisateur-innovateur (Lakhani et von Hippel, 2003 ; von Hippel et von Krogh, 2003) : comme les utilisateurs bénéficient directement de l'innovation qu'ils produisent, ils sont incités à la produire et, comme ils peuvent s'attendre à des retours d'information ou des innovations cumulatives sur leur proposition, ils sont incités à la partager librement. Jullien et Roudaut (2012) ont montré la difficulté pour un projet ouvert à réussir lorsque les contributeurs ne sont pas ses utilisateurs.

L'utilisateur-innovateur n'est pas forcément un individu. Il peut s'agir d'une entreprise qui fait développer un logiciel pour ses besoins propres et demande une licence libre pour ne pas être dépendante de son fournisseur (voir l'étude de la stratégie « libre » du Département de la Défense des États-Unis par Le Texier et Versailles, 2009), ou qui recherche des collaborations pour partager les coûts de développement (comme Google avec Tensorflow et Facebook avec Pytorch, les deux solutions leaders d'apprentissage machine). Elle peut vouloir se séparer d'un logiciel qui devient trop coûteux à maintenir par rapport à son avantage stratégique (Van der Linden *et al.*, 2009), comme AOL avec le navigateur web Netscape (Ãgerfalk et Fitzgerald, 2007). Une troisième catégorie est apparue plus récemment : il s'agit de projets libres planifiés, généralement menés par un consortium d'entreprises, comme OpenStack (Teixeira *et al.*, 2016). L'objectif est alors de créer une norme industrielle, ou de mutualisation par la demande. On retrouve les différentes stratégies d'innovation ouverte de West et Gallagher (2006).

La phase 1 est donc une phase où les besoins utilisateurs s'inscrivent dans une évolution dynamique du logiciel ; la licence libre permet de s'assurer du contrôle du logiciel dans la durée et de participer à son développement pour s'assurer que ses besoins propres seront pris en compte, soit la stratégie Contrôle, mais aussi de mutualiser les coûts de développement, et de rechercher des innovations externes

qui améliorent les fonctionnalités. Cela se comprend, car la solution logicielle est un actif complémentaire (service de cloud computing pour OpenStack), adapté pour apporter un avantage concurrentiel (West et Gallagher, 2006 ; Jullien et Zimmermann, 2011b ; Teixeira *et al.*, 2016). À ce stade, le logiciel est souvent immature à l'image du demi-produit décrit par Millier (1997). Il convient au besoin des clients dits « à logique technique » qui sont prêts à collaborer pour développer un nouveau produit, par opposition aux clients « à logique achat » qui recherchent un produit sur étagère répondant à des besoins standardisés avec une prise de risque limitée (*ibid.*).

Si le logiciel n'est maintenu que par un acteur, la dépendance à cet acteur reste forte et la solution s'apparente plus à un logiciel gratuit qu'à un logiciel libre[8]. Autrement dit, un logiciel n'est libre qu'à partir du moment où son évolution est assurée par un consortium d'acteurs (individus ou entreprises) qui proposent et éventuellement participent à la gouvernance des évolutions. À partir du moment où le projet propose une base stable, il permet des demandes « Lock-out » : des solutions plus standards (Ven *et al.*, 2007), mais sur une base fonctionnelle qui reste limitée. Si le consortium grossit, on atteint la phase 2 d'une action collective.

- ***Phase 2 : Épanouissement ou délitement***

C'est la phase de percolation (Marwell et Oliver, 1993), ou de rendements croissants d'adoption (Arthur, 1994). Les entreprises et utilisateurs VH sont attirés par le potentiel élevé, mais pas encore pleinement réalisé du projet, leur permettant d'exercer un certain niveau de contrôle et de façonner l'avenir du projet (construire la fonctionnalité dont elles ont besoin). Les volontaires sont attirés par la visibilité et la réputation croissantes des contributions potentielles au projet.

Le défi est alors de mettre en place les structures de gouvernance et l'infrastructure modulaire essentielles 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 importante en nombre de travailler efficacement (Mockus *et al.*, 2002). Il s'agit de construire une « architecture de participation » (MacCormack *et al.*, 2006), qui facilite aussi l'entrée de nouveaux contributeurs, car ils peuvent se concentrer sur le code, le module qui les intéresse (Bessen, 2006). La formalisation de la gouvernance est nécessaire à la stabilité de l'écosystème, par exemple pour éviter les phénomènes de forks (Viseur, 2012). Les solutions peuvent être variées, selon la possession des droits sur le code du logiciel, mais aussi la marque : groupe projet, fondation ou propriété d'un éditeur open-source semblent être les trois structures possibles (Charleux et Mione, 2018). 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). Il est nécessaire de trouver le bon équilibre entre une organisation hiérarchique, source d'efficacité productive (Lee *et al.*, 2017), et l'ouverture, qui facilite l'intégration de nouveaux membres et donc l'innovation, dans un dilemme classique entre exploration et exploitation d'une solution technologique

[8] Ainsi, en théorie les utilisateurs KM pourraient être initiateurs, car ils peuvent maîtriser la partie expression de besoins. Mais ils vont avoir des problèmes de contrôle de la solution, et rester dépendants des fournisseurs qui développent la solution à cause de leur manque de compétences techniques. On est dans un projet sous licence ouverte, mais pas libre, au sens où il n'y a pas une collectivité de développeurs issus d'horizons différents qui cogèrent le développement.

(March, 1991)[9]. Si le projet ne relève pas ces défis, il s'atrophie et il y a de fortes chances qu'un projet concurrent s'impose. S'il les relève, il peut accueillir de nouveaux contributeurs, qui proposent de nouvelles fonctionnalités, attirant de nouveaux utilisateurs, qui rendent le projet plus visible et plus attrayant pour les développeurs et les fournisseurs de solutions complémentaires. C'est le principe des plates-formes multifaces (Rochet et Tirole, 2006).

Ensuite, la complexification même du projet génère de nouveaux besoins auxquels des entreprises peuvent répondre. Si le nombre de modules augmente, il faut choisir, puis suivre les évolutions. Il faut aussi intégrer les solutions dans les systèmes d'information des entreprises. En plus des besoins Contrôle, on peut alors voir se développer une demande pour le Lean, où la solution libre, organisée sous forme de modules, facilite l'adaptation d'une solution standard accompagnée d'adaptation des modules. Cette phase permet le développement d'offres d'intégration (accès et assurance sur des versions de logiciels et sur la maintenance en condition opérationnelle de ces logiciels), à côté de l'assistance (évaluation d'une solution), puis de l'adaptation aux besoins. Cela peut nécessiter la mise en place de systèmes de certification des offres commerciales, proches de ceux existant pour les éditeurs classiques, car ce sont des signaux de confiance que peuvent comprendre les utilisateurs KM et qui permettent le développement du marché des services complémentaires (Karpik, 1996). Ces certifications peuvent être délivrées par l'entreprise éditrice du logiciel, ou par la fondation qui le gère à l'image de ce que propose la Fondation Linux pour le logiciel libre Kubernetes de déploiement et de gestion de logiciel dans des plates-formes cloud. Germonprez *et al.* (2020) expliquent que c'est même devenu une des fonctions majeures de ces fondations.

Finalement, un projet logiciel, même libre, peut atteindre un niveau de maturité tel que les évolutions ralentissent, notamment car les besoins fonctionnels sont couverts.

- ***Phase 3 : Maturité et au-delà***

La maturité ne signifie pas absence d'évolution : il y a toujours des mises à jour techniques dans un projet logiciel. Cependant, le nombre de contributeurs nécessaires à cette maintenance, ainsi que l'intérêt intellectuel du projet diminuent. À cet égard, ces projets ouverts en ligne semblent, comme les organisations traditionnelles (Hannan et Freeman, 1984), être menacés par la tendance naturelle à l'inertie structurelle et par une difficulté croissante à s'adapter (O'Mahony et Ferraro, 2007). Cela peut attiser des phénomènes de fork si les contributeurs estiment que l'inertie nuit trop à la prise en compte de leurs besoins (Viseur, 2012), ou conduire à l'abandon du projet avec la fin des mises à jour techniques.

Ces solutions matures installées dans les organisations doivent être maintenues, et les entreprises utilisatrices ont toujours des besoins d'assurance réalisée en interne ou sous-traitée, soit la « tierce maintenance applicative » (TMA). Cela peut prolonger la vie du projet libre et les contributions. D'une certaine façon, si avec le logiciel libre il y a moins de « vendor lock-in », les utilisateurs ont toujours des risques de lock-in technologique (Arthur, 1989), qui engendrent des coûts de maintenance avec arbitrage entre faire et faire faire. De plus,

[9] En pratique, la gouvernance des projets libres s'inscrit dans un continuum de degrés d'ouverture entre ces deux cas. Laffan (2012) a proposé une mesure du degré d'ouverture de la gouvernance, l'Open Governance Index, qui a pour but de quantifier, sur une échelle de 0 à 100%, le degré d'ouverture d'un projet en termes de transparence, de prise de décision, de réutilisation et de structure communautaire.

l'utilisation de logiciels matures dans des plates-formes cloud permet de répondre à des besoins d'accès à une technologie entretenue à bas coût (« Prix ») en baissant les coûts d'accès, car les coûts de licence sont nuls (voir Partie 4 pour le côté offre). Enfin, il peut rester des besoins « Lean » : choix des composants, paramétrage de ceux-ci, voire développement de nouveaux composants pour des besoins spécifiques.

Par contre, les utilisateurs ayant des besoins « Contrôle » et « Lock-out », et particulièrement les utilisateurs VH, rechercheront des offres, des projets plus jeunes, qu'ils pourront plus facilement adapter à leurs besoins (contrôle) ou pour lesquels le lock-in technologique sera moins fort (Lock-out), initiant ainsi de nouveaux projets logiciels libres.

- ***Synthèse***

Nous avons montré que la possibilité pour les utilisateurs de contrôler l'évolution d'une solution explique le lancement d'un projet logiciel, quand l'indépendance vis-à-vis d'un acteur, et le partage des coûts de développement (développement collectif) expliquent le succès au-delà de l'initiative des utilisateurs-producteurs. Les besoins s'appuient essentiellement sur la gestion de cette dynamique et du logiciel-flux (adaptation et assurance), pour répondre aux besoins « Contrôle », puis « Lock-out ». La croissance de l'écosystème et du projet organisé de façon modulaire répond à des demandes d'adaptation (développement et paramétrage des modules) et d'assurance (« Lean »). Avec la maturité du projet et de la solution, des demandes d'accès à bas coût (Prix) peuvent se développer via la fourniture de la solution sur des plates-formes cloud. Autrement dit, la création de valeur des projets libres, les caractéristiques des utilisateurs intéressés par cette création de valeur et les stratégies open-sources associées évoluent aussi avec le projet, ce que nous résumons dans la figure 2.

Dans la partie suivante, nous proposons d'analyser quelles sont les activités et ressources clefs qui permettent à des modèles open-sources de répondre aux besoins complémentaires exprimés et surtout de capter une partie de la valeur créée.

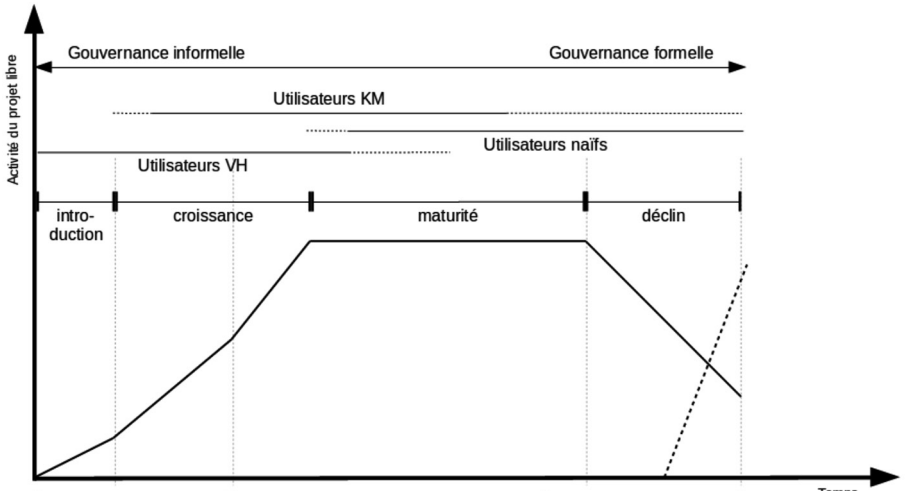
4. CAPTATION DE LA VALEUR : RESSOURCES ET ACTIVITÉS CLEFS

Teece (2018) insiste sur l'alignement interne et la cohérence des éléments du modèle d'affaires en matière de captation de valeur, ce qu'il appelle les modèles de revenus, incluant la tarification, les canaux et le type d'interaction avec la clientèle ciblée, et les modèles de coûts, incluant les activités, les ressources et les partenaires clefs, c'est-à-dire selon Dierickx et Cool (1989), les capacités difficilement échangeables ou imitables. Nous étudierons ce lien dans le cas général de l'informatique avant de le spécifier dans le cas open-source.

4.1. Quelles activités et quelles ressources pour quels métiers dans l'informatique ?

4.1.1. Activités marchandes en informatique

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, elles répondent aux besoins d'assistance.



N.B. La durée de chaque phase varie selon les logiciels. Certaines phases peuvent durer plusieurs années, voire plusieurs décennies (Linux).

Figure 2 : Cycle de vie et modèle économique d'un projet libre.

Les activités de gestion de projet portent sur la gestion d'une technologie (création d'un logiciel interne ou activité d'édition logicielle) ou la mise en œuvre d'un logiciel (adaptation d'un logiciel sur site « on-premise » ou activité d'intégration logicielle). 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 ; les capacités de gestion de projet technique, liées à la maintenance sur le long terme de la technologie, répondant aux besoins d'assurance. Lee et Lee (2004) parlent, dans le cas de l'implémentation de progiciels de gestion intégrée, de capital IT humain et de capital IT technique.

Il en résulte les trois capacités suivantes (Gabay, 2014) : les capacités spécifiques aux clients (MoA), les capacités de gestion de projet client (MoA vers MoE) et les capacités de gestion de projet technique (MoE). Enfin, pour répondre au besoin d'accès, la distribution d'une version d'un progiciel est une activité différente de la gestion du projet technique (MoE).

4.1.2. Actifs et capacités spécifiques

Pour les activités spécifiques au client, ce sont les contacts répétés avec les clients qui construisent l'actif spécifique (Dierickx et Cool, 1989) : la connaissance plus ou

moins tacite des besoins et profils de clients. Ces actifs sont basés sur des ressources humaines, mais aussi sur la capacité de ces entreprises à organiser la capitalisation des connaissances des salariés. Ainsi, les grands cabinets de conseil, comme Wavestone ou Accenture, sont souvent organisés autour de services sectoriels (banque-assurance, automobile, chimie, etc.) et/ou de fonctions (achat, RH, marketing, etc.)

Pour les capacités de gestion de projet client, l'expertise se développe grâce à des relations répétées avec une ou quelques technologies. Elle est éventuellement sanctionnée par des diplômes ou des partenariats privilégiés avec des éditeurs, qui seront alors des partenaires clefs, tandis que les individus experts qui portent la connaissance de la technologie dans l'entreprise sont des ressources clefs[10].

D'activité artisanale conditionnée au seul talent des programmeurs, le développement logiciel a progressivement évolué vers une industrialisation accrue (Duvall *et al.*, 2007) recourant à la gestion en continu des versions et des déploiements (par exemple l'approche DEVOPS), visant notamment à en réduire la dette technique (Kruchten *et al.*, 2012), c'est-à-dire l'accumulation d'erreurs de programmation nuisant à la maintenance et à l'extensibilité. Ces contraintes de qualité et d'automatisation ont été rendues encore plus nécessaires par le développement de l'informatique en mode SaaS (Plouin, 2016 ; Bezemer et Zaidman, 2010). Ainsi, l'activité de gestion de projet technique repose sur le contrôle, d'une part, de ressources humaines très spécialisées et difficilement substituables du fait de la maîtrise des technologies acquises par la formation et l'expérience (courbe d'apprentissage) et, d'autre part, de ressources technologiques (infrastructures de suivi de versions, infrastructures d'intégration continue, bibliothèques de tests automatisés, scripts de déploiement, de migration, d'intégration...). Une protection complémentaire peut provenir du contrôle de la marque associée au produit (Messerschmitt et Szyperski, 2001). Dans les trois cas, la marque symbolise ainsi la qualité perçue par le marché de l'entreprise et de son expertise, et leur visibilité peut être considérée comme un actif spécifique (Dierickx et Cool, 1989).

Finalement, l'activité de distribution s'est sans doute simplifiée avec l'arrivée d'Internet. La distribution en magasin de supports physiques est remplacée par les sites web, les gestionnaires de paquets et les magasins d'applications. Cependant, il faut toujours faire connaître le logiciel, le faire adopter par des utilisateurs ou des prescripteurs, en plus d'organiser la publication de versions et le retour utilisateur. Il est donc toujours nécessaire de s'appuyer sur des compétences commerciales incluant le packaging de la solution, la constitution d'un réseau de distribution et l'exploitation d'une marque forte (Messerschmitt et Szyperski, 2001).

Nous avons ainsi identifié les actifs spécifiques suivants, qui renvoient aux actifs décrits dans la littérature par Boissin (1999), complétée par Douard et Heitz (2003) sur les réseaux de l'organisation : les actifs humains recouvrent des ressources humaines clefs orientées soit vers le client (fourniture d'expertise), soit vers la technologie (fourniture de capacités techniques entretenues) ; les actifs cognitifs concernent la création de bases de connaissances (bugs et référentiels de besoins) ; les actifs technologiques recouvrent les codes sources créés et détenus par le prestataire ; les actifs organisationnels concernent les procédures et infrastructures permettant

[10] Voir, à nouveau, les entreprises de conseil en technologie, et par exemple la présentation que fait Accenture de ses compétences autour des solutions d'Oracle.

l'organisation du développement. Elles peuvent être orientées vers l'intérieur (développements spécifiques industrialisés) ou vers l'extérieur (logiciel flux) ; Les actifs réputationnels sont relatifs aux marques ; enfin, les actifs commerciaux concernent les distributeurs et, plus globalement, les partenaires contribuant à la diffusion des produits ou des services.

Si chaque activité peut exister de manière indépendante, les théories de l'économie des services (Gadray, 2003), notamment pour l'informatique (Jullien et Zimmermann, 2011b), expliquent les trois associations les plus courantes : celle entre l'ingénierie des exigences et la traduction en spécifications techniques (activités de conseil, métier ou technique), centrée sur l'assistance et l'adaptation ; celle entre l'ingénierie des exigences et l'édition, ou gestion dans la durée de la solution technique (activités de maîtrise d'œuvre), centrée sur l'adaptation et l'assurance ; enfin, celle entre l'édition et la distribution (activités de fournisseur de progiciels), centrée sur l'assurance et l'accès.

Pour la première association, il s'agit d'articuler des compétences humaines, ce qui renvoie à la littérature sur la gestion des connaissances[11]. Les rendements croissants d'adoption, essentiellement sous la forme d'effets d'apprentissage, permettent le développement d'actifs spécifiques. Les barrières à l'entrée sont assez faibles. L'activité est basée essentiellement sur des coûts variables (les ressources humaines) avec donc un coût marginal du client stable, voire légèrement croissant avec le risque de décroissance de la rentabilité à cause des coûts de coordination. Elle se prête surtout au paiement à la prestation avec des taux horaires variables suivant l'intensité de la concurrence et la réputation du prestataire.

Pour la seconde association, avec les procédures du génie logiciel et le management des connaissances (Ourique *et al.,* 2019), on trouve aussi le contrôle d'une plate-forme technique et la réutilisation à coût nul d'un composant (le logiciel), donc des économies d'échelle. Les coûts d'entrée, fixes, sont donc plus forts, car il faut contrôler une plate-forme technique suffisamment mature pour avoir une demande, mais le coût marginal d'un utilisateur ou client va décroissant. La rémunération se prête davantage à des formules fixes (abonnements ou rémunérations forfaitaires).

La troisième association combine les effets d'apprentissages aux économies d'échelle (sur la solution, mais aussi sur la diffusion des versions), et bénéficie, pour la validation de la compréhension des besoins, de l'intrication possible entre technique et marketing. Celle-ci est permise par les pratiques de gestion analytique de l'innovation (Ries, 2012) et leur traduction en développement logiciel (par exemple feature toggles, cf. Rahman *et al*., 2016), et par les évolutions technologiques (activités de DEVOPS notamment). On est sur la prestation d'accès à une capacité technique entretenue, à très forts coûts fixes, rémunérée forfaitairement (licences ou abonnements).

4.1.3. Captation de valeur (flux financiers)

L'analyse sous l'angle de création de valeur (Figure 1) permet de préciser les liens entre activités commerciales et besoins.

[11] Nous renvoyons, sur les questions de la gestion des rendements croissants d'apprentissage à l'abondante littérature en système d'information sur les systèmes de gestion des connaissances. On pourra consulter, par exemple, North et Kumta (2018) pour une introduction aux différentes dimensions de la question, et à Gonzalez (2016) sur les questions de capitalisation de connaissance dans l'industrie du service (une question classique de la littérature en GRH, voir, par exemple, Morris, 2001).

Répondre au besoin Contrôle nécessite des tâches d'adaptation principalement, ensuite d'assurance et, plus marginalement, d'assistance avec une activité de conseil liée à la mise en place ou même au choix technologique. Les solutions recherchées garantissant une indépendance technologique, les activités de maîtrise d'œuvre sont primordiales, avec la fourniture de capacités humaines pour l'adaptation de la solution et éventuellement la fourniture de capacités techniques entretenues dans la partie assurance.

Il en est de même pour les demandes Lock-out, pour les mêmes raisons, même si la demande porte vers des solutions plus standards, moins coûteuses, donc plutôt vers la fourniture de capacités techniques entretenues. Les solutions recherchées peuvent être compétitives en prix avec éventuellement une partie d'assistance sur le choix de la technologie, via la fourniture de capacités humaines pour les activités de conseil.

La satisfaction du besoin Prix se dirige vers des offres accès et assurance qui sont concurrentielles en prix, donc vers les solutions qui proposent les économies d'échelles les plus fortes, i.e. vers les fournisseurs de progiciels qui proposent la fourniture de capacités techniques entretenues ou la monétisation de l'accès.

Finalement, la satisfaction du besoin Lean est celle qui peut passer par une sous-traitance des tâches les plus variées (accès, assurance, assistance et adaptation), donc un mélange de fourniture de capacité technique entretenue et de capacité humaine, mais à coût réduit. S'il y a adaptation, c'est sur des composants standards, soit avec un couple éditeur-distributeur et conseil, ce dernier paramétrant la solution de l'éditeur, soit sur une maîtrise d'œuvre d'un composant suffisamment ouvert pour être adaptable et suffisamment bon marché pour financer le service à façon.

Les activités d'assistance associées à la maîtrise d'ouvrage sont en définitive peu impactées par le logiciel libre, car les choix technologiques ne sont pas au cœur de la proposition de valeur. Bien sûr, les utilisateurs VH voire même KM fournissent au projet un retour en continu sur les orientations prises ou annoncées (Teigland *et al.*, 2014). Cependant, le même système de retour, par des forums, des clubs utilisateurs peut exister pour les éditeurs classiques (Jayanth *et al.*, 2011). Dès lors, le logiciel libre, qu'il soit flux ou stock, n'apparaît pas comme une ressource clef pour ces métiers. C'est dans la relation avec la technologie, et dans l'évolution technologique de la solution (logiciel flux), que des entreprises vont développer des actifs et activités spécifiques qui vont leur permettre de capter de la valeur.

4.2. Les modèles d'affaires open-sources

Comme expliqué dans la partie 3, les projets de logiciel libre suivent différentes phases. Selon les phases, les compétences à maîtriser pour utiliser un logiciel libre, le type de prestation et les besoins couverts varient. Nous discutons ici du lien entre besoins et prestation à chaque phase pour définir les modèles d'affaires open-sources.

4.2.1. Capacités et captation en phase 1

En phase 1, le logiciel est peu complet fonctionnellement, peu mature en matière de qualité structurelle et peu connu. Les utilisateurs, de type VH, peuvent choisir ce type de technologie pour satisfaire des besoins du type Contrôle. La sous-traitance porte donc sur le développement du logiciel (adaptation) en vue de son intégration dans le système d'information des clients (assistance). Le projet libre permet un

abaissement du TCO du fait de la disparition des frais de licence et de la possibilité de faire évoluer la solution en direction de ses besoins.

Si des utilisateurs VH s'associent pour partager les risques de développement d'un nouveau logiciel, ils peuvent choisir de confier à un tiers, prestataire privé ou fondation, l'animation de la mutualisation (cas du Department of Defense et d'AdaCore pour le prestataire, Apache HTTP et Apache Foundation pour la fondation). Ils peuvent aussi être rassemblés par un acteur au sein de clubs de manière à organiser la discussion des besoins et de leur financement (cas d'XWiki). Finalement, le rapport de force entre, d'une part des utilisateurs entreprises ayant les moyens de développer des compétences VH et de choisir de faire développer ou adapter un logiciel à leurs besoins, et, d'autre part, un prestataire de développement spécialisé dans des activités de programmation permet aux utilisateurs VH de contrôler le logiciel.

Le prestataire engagé dans le développement du logiciel monétise sa connaissance approfondie de la solution (De Laat, 2005) et des développements libres (Fitzgerald, 2006 ; Stol *et al.*, 2014). Celle-ci est positivement liée au nombre de prestations vendues, mais reste une activité de fourniture de capacités humaines. Cette capacité à discuter avec les utilisateurs VH et à proposer des développements mutualisés est probablement indispensable pour assurer le succès initial du projet libre, mais aussi de l'entreprise. Les développeurs constituent donc la ressource clef. Ils doivent être polyvalents en ce sens qu'ils doivent être capables de développer des solutions techniques à partir de spécifications des clients, mais aussi de les intégrer dans une architecture logicielle cohérente. La polyvalence de ces développeurs pourra faciliter les évolutions ultérieures du prestataire (Narayanan *et al.*, 2009).

Dans la phase 1, le modèle d'affaires d'un prestataire libre est très proche d'un modèle de personnalisation. Cependant, dans un modèle classique on est plutôt dans une relation bilatérale de développement à façon. Dans un modèle open-source, il s'agit d'animer aussi un groupe privé d'utilisateurs-développeurs qui mutualisent des ressources de développement, soit en finançant le prestataire, soit en participant au développement, soit en combinant les deux. Nous parlerons donc de *personnalisation open-source*.

4.2.2. Capacités et captation en fin de phase 1

L'ouverture du projet au-delà du club fermé d'utilisateurs marque la transition de la phase 1 vers la phase 2. Pour que cette transition soit possible, la gouvernance (Viseur et Charleux, 2019) et l'infrastructure technico-organisationnelle doivent évoluer pour organiser la distribution des tâches entre contributeurs au projet. Le projet évolue vers un modèle d'édition industrielle d'un logiciel. Deux types de compétences clefs se révèlent nécessaires : l'ingénierie des exigences et, de plus en plus, la gestion dans la durée de la solution technique.

L'activité d'édition du logiciel peut être sous-traitée à un acteur privé ou à une fondation (Riehle et Berschneider, 2012 ; Charleux et Mione, 2018). Dans des cas plus rares, des acteurs réussissent à se coordonner, pour la partie infrastructure du moins, grâce à des plates-formes technologiques comme GitHub ou GitLab. Pour sa pérennité, l'éditeur doit convaincre la base d'utilisateurs de le financer, ce qui n'est pas toujours le cas (nous pensons en particulier aux applications pour le grand public). Ce sont donc toujours les utilisateurs-développeurs VH qui arbitrent entre faire ou faire faire. Cependant, plus il y aura d'utilisateurs et de contributions, plus les

activités de suivi peuvent être coûteuses, et plus on pourra avoir des activités d'assurance mutualisées, comme l'organisation de système de version, voire de distributions garanties. La fourniture de capacités techniques entretenues, la capacité technique étant le logiciel flux, gagne en importance. Les développeurs cœurs sont les mieux placés pour apporter des garanties sur le fonctionnement et éventuellement sur les évolutions. Les services d'expertises peuvent être intégrés dans une organisation pour mieux les coordonner, et gérer les aspects contractuels de la solution, voire commencer à organiser des niveaux de service et de retour utilisateurs. C'est pourquoi les développeurs cœurs, à l'origine du projet, participent souvent à la création d'un prestataire privé à ce stade s'il n'existe pas, ou sont intégrés s'il existe. C'est le modèle d'*éditeur open-source*.

D'autres actifs spécifiques renforcent l'avantage concurrentiel de l'éditeur : la propriété sur la marque associée au nom du projet libre permet de profiter de sa notoriété (Schaarschmidt *et al.*, 2015), et la base de tests de non-régression permet de maîtriser, garantir et assurer l'évolution de la qualité du projet. En cette fin de phase 1, le logiciel atteint le statut de commodité capable de satisfaire les besoins les plus communs, donc les premiers besoins Lockout. Cependant, il est difficile de développer une proposition de valeur rentable, car le client, souvent encore du type VH, cherche des fonctionnalités standards et le logiciel est perçu comme une commodité sans développement à façon, qu'il peut installer et maintenir lui-même.

4.2.3. Capacités et captation en phase 2

La phase 2, avec l'arrivée de nouveaux utilisateurs-contributeurs, ouvre de nouvelles opportunités. Dans cette phase, les entreprises utilisateurs VH sont toujours attirées par le potentiel élevé, mais pas encore pleinement réalisé, du projet qui permet d'exercer un certain niveau de contrôle sur l'avenir du projet en vue de construire la fonctionnalité dont elles ont besoin. Le développement de modules, et donc de la couverture fonctionnelle, étend les usages et utilisateurs potentiels de la solution vers des utilisateurs KM. Comme expliqué sur la fin de la phase 1, la complexification même du projet génère de nouveaux besoins auxquels des entreprises peuvent répondre : en plus des besoins d'assurance, si le nombre de modules augmente, il faut choisir les bons modules, et suivre leurs évolutions. Il faut toujours intégrer les solutions dans les systèmes d'information des entreprises. En plus des besoins Contrôle, on peut voir se développer une demande pour le Lean, où la solution libre, organisée sous forme de modules, facilite l'adaptation d'une solution standard, sans coût de licence et qui peut être testée avant d'être adoptée. Apparaissent ainsi différentes structures d'offres, reposant sur le contrôle d'actifs et d'activités différents. Nous présenterons d'abord les évolutions de ce qui existait en phase 1 autour de la gestion de la solution et de l'édition, avant de discuter de la gestion des modules.

Si un éditeur privé issu de la phase 1 existe, il peut améliorer son avantage concurrentiel en phase 2. Cela est possible dès lors que la technologie évolue régulièrement (besoin d'assurance sur le logiciel flux) et que l'éditeur contrôle la fourniture de la capacité technique entretenue : le système de gestion des contributions et des versions, via le contrôle des développeurs au cœur du projet. La propriété de la marque associée au projet (son nom) permet de contrôler la version officielle et de profiter de la réputation du projet (Valimaki, 2003 ; Fitzgerald, 2006). D'autres actifs, coûteux à imiter car construits dans le temps (Dierickx et Cool, 1989 ; Demil

et Lecocq, 2010), procurent un avantage concurrentiel important, comme une base de tests de non-régression, car ils permettent de garantir la qualité de la distribution (assurance), à l'image de l'entreprise AdaCore (Brosgol, 2019). Par rapport à un éditeur classique, et même s'il ne faut pas oublier les coûts importants d'animation, l'éditeur open-source bénéficie de coûts de développements d'autant moins élevés que les utilisateurs VH sont actifs : mutualisation des efforts de développement, exploration de nouvelles pistes par les contributeurs et feedback à propos des orientations prises ou annoncées. Cependant, il ne bénéficie pas des revenus liés à la vente de licences d'accès du modèle classique.

Trois modèles d'affaires sont possibles, répondant à trois types de besoins.

Dans le cas de logiciels complexes, demandant une forte expertise technique, le modèle que nous avons présenté dans la fin de la phase 1 persiste : il reste animateur d'une communauté d'utilisateurs VH et vend son expertise supérieure du logiciel via des prestations de conseil sur l'utilisation, l'implémentation, voire l'adaptation du logiciel. C'est ce que fait AdaCore Technology par exemple, ou certains éditeurs de plates-formes, comme l'éditeur de l'ERP Odoo, vis-à-vis des prestataires périphériques qui installent les plates-formes ou qui développent des modules spécifiques. Les éditeurs de distributions, comme RedHat, peuvent aussi proposer des solutions adaptées à des grands comptes, notamment des services de déploiement et de maintenance de distribution à très grande échelle, ce qui est facilité par leur contrôle sur les outils de gestion de paquets. Dans tous les cas, la connaissance intime du cœur de la solution libre, et de ses évolutions, est l'actif clef. C'est toujours le modèle éditeur open-source.

Lorsque des besoins plus standards apparaissent, la certification de l'édition de versions standards peut intéresser des utilisateurs KM, en offrant des garanties supplémentaires vis-à-vis de l'offre libre. C'est par exemple le cas du service de distribution certifié Red Hat Linux par rapport à Fedora. Elle repose sur les économies d'échelle dues à l'intégration automatisée d'un large ensemble de modules, standards, en direction d'un public qui a moins besoin d'adaptation (Lock-out), mais qui a aussi une faible disposition à payer, ou qui a besoin d'une version stable du cœur du logiciel pour supporter les modules périphériques qui sont, eux, adaptés (besoins Lean). On peut alors parler d'éditeur-distributeur, ou d'*intégrateur open-source*. Celui-ci vend l'accès à une capacité technique entretenue, le logiciel flux, qu'il contrôle. La ressource clef réside toujours dans les éléments de production et d'assemblage de la solution : la plate-forme de génie logiciel et les développeurs qui s'y rattachent.

Enfin, l'éditeur de la phase 1 peut renforcer son contrôle, notamment sa propriété intellectuelle sur le code, via des artifices juridiques tels que les accords de contribution permettant le partage de droits patrimoniaux (Poo-Caamaño et German, 2015). Il s'agit de générer des contributions d'utilisateurs pour diminuer les coûts de développement, tout en monétisant l'accès à la solution (Messerschmitt et Szyperski, 2001), au logiciel stock. Cela se traduit classiquement par des systèmes de double licence (Valimaki, 2003), lesquels peuvent s'apparenter à une forme de modèle freemium (Osterwalder et Pigneur, 2010). Cependant, dans un modèle freemium, on vend quelque chose en plus de la version gratuite, alors que là, la version refermée offre une marge de manœuvre moindre à l'utilisateur. Elle ne sera pas choisie par les clients Lock-out ou les clients Contrôle. S'il s'agit d'une version certifiée, c'est qu'il y a des besoins d'assurance sur l'accès à un logiciel flux, et d'autres stratégies permettent cela. Finalement, il semble que cette stratégie s'adresse à des

demandes Prix, donc sur des solutions standards, où il faut développer de fortes économies d'échelle. L'éditeur peut aussi arrêter de maintenir la version libre, dans le but d'obliger les utilisateurs, enfermés dans un lock-in technologique, à payer pour l'accès. Cependant, les utilisateurs-contributeurs peuvent aussi changer de solution, ou créer une solution parallèle, un fork (Nyman *et al.*, 2012 ; Viseur et Charleux, 2019), si le coût de la licence et du contrôle complet par l'éditeur n'est pas supportable (voir par exemple le fork Tryton du projet TinyERP, aujourd'hui Odoo). Il semble que ce modèle, même s'il a depuis longtemps été mis en avant dans la littérature (Kogut et Metiu, 2001), ne soit pas réellement soutenable. D'autres modèles comme le software as a service (SaaS, ou services cloud) sont aujourd'hui préférés pour facturer l'accès (cf. phase 3).

Pour répondre aux besoins Contrôle et Lean, il faut adapter la solution et des acteurs privés peuvent proposer le développement ou la modification de modules spécifiques.

Si les modules sont développés de façon libre, ils peuvent avoir les mêmes avantages pour un utilisateur que le cœur de la solution : pas de coût de licence, test possible, respect des standards, adaptation facilitée par l'ouverture du code. Dans ce cas, les compétences des prestataires porteront, pour commencer, sur l'adaptation de module, donc de la prestation. Leur avantage concurrentiel peut se construire de la même façon que pour l'animateur du projet en phase 1 : partant d'un avantage basé sur de l'expertise (capacité humaine), ils peuvent évoluer vers de la fourniture d'une capacité technique entretenue, s'ils arrivent à organiser les activités d'édition autour de modules, et donc de vente d'assurance sur des distributions de modules ou de versions spécifiques du logiciel. Ils deviendront alors des *éditeurs métiers open-sources*. Si les modules sont développés et vendus sous des licences privées classiques, l'éditeur-distributeur du module bénéficie du fait que le cœur de la solution reste libre (et gratuite), diminuant le TCO pour le client. Ces offres ne répondent pas aux besoins Contrôle ou Lock-out, mais à éventuellement à des besoins Lean (*éditeurs métiers classiques)*. Dans les deux cas, l'éditeur métier n'a pas besoin de développer une expertise sur le cœur de l'application (MacCormack *et al.*, 2006), l'aspect modulaire et ouvert permet de concentrer ses ressources sur le contrôle des parties périphériques du projet libre. Cependant, il faut toujours être en capacité de fournir des versions assurées d'un logiciel flux, par des activités de gestion de projet logiciel.

Les prestataires peuvent aussi rester sur des activités de conseil et d'adaptation à façon, en combinant des expertises métiers et des expertises techniques sur les modules, permettant de répondre aux besoins Contrôle (adaptation fine, sans lock-in), mais où on bénéficie moins d'économies d'échelle, tout au plus d'effets d'apprentissage. Nous les nommons les *conseillers solution-métiers open-source*.

On voit donc que la phase 2 est une phase où l'écosystème du projet libre se développe, permet l'émergence de différents modèles d'affaires open-sources, qui peuvent coexister, voire être complémentaires. Ces modèles capitalisent sur l'ouverture du projet et l'exploitation du logiciel flux. Osterloh et Rota (2007) rappellent que l'ouverture rencontrée dans le logiciel libre, comme dans l'innovation en général, tend à n'être qu'une situation transitoire permettant de faire évoluer plus rapidement la technologie dans un contexte d'incertitude. L'innovateur se réserve généralement dans un second temps les fruits de ses contributions. Lorsque la solution logicielle devient mature, que le projet entre en phase 3, les modèles d'affaires open-sources, qui étaient basés sur la gestion du

logiciel flux tendent à se recentrer sur la gestion d'un logiciel stock et à se rapprocher des modèles classiques.

4.2.4. Capacités et captation en phase 3

La maturité d'un projet libre ne signifie pas, au moins dans l'immédiat, la fin des besoins, même si les utilisateurs ayant des besoins Contrôle, et particulièrement les utilisateurs VH, rechercheront des offres, des projets plus jeunes, qu'ils pourront plus facilement adapter à leurs besoins, initiant ainsi de nouveaux projets logiciels libres. Les besoins de garantie d'accès à un logiciel flux (assurance) diminuent, car le logiciel arrive à maturité, sans disparaître, si le logiciel dépend de composants technologiques évoluant régulièrement. Fournir des ressources logicielles sans coût d'accès reste un avantage concurrentiel important. On peut toujours capter de l'accès et de l'assurance, mais chez des clients qui ont une faible disposition à payer pour la solution simple, qui va donc être intégrée dans des portefeuilles de solution. C'est ce que proposent les prestataires chargés, souvent pour un coût fixe, de gérer l'infrastructure d'un client, soit l'infogérance, ou les prestataires qui ont développé des offres d'accès à une fonctionnalité, soit la servitisation, via le SaaS.

L'infogérance s'est développée sur la tendance à l'externalisation des compétences non centrales, incluant l'IT (Jousset, 2005). Son intérêt est de permettre des économies d'échelle et de gamme sur la gestion et la maintenance de logiciels : d'échelle, car les mêmes solutions informatiques peuvent être suivies et maintenues pour différents clients ; de gamme, car, pour un même client, une même relation commerciale, on peut gérer plusieurs logiciels. L'infogérance classique repose sur une relation proche avec les éditeurs qui certifient leurs prestataires. Le logiciel libre, au-delà des coûts plus faibles d'accès, permet au prestataire de choisir son niveau d'investissement dans le suivi du logiciel en fonction de son importance pour les clients et de son évolution (Jullien et Zimmermann, 2011a) : s'appuyer sur des versions officielles, s'impliquer un peu plus dans le suivi des bugs et de leur correction, ou avoir un ou deux spécialistes du logiciel, donc des développeurs-contributeurs, pour un suivi du logiciel flux moins coûteux que lors de la phase 2. L'infogérance de logiciels libres peut diminuer le risque perçu de lock-in et en offrant une porte de sortie, même théorique, en cas d'insatisfaction vis-à-vis du fournisseur, cela peut toujours intéresser certains utilisateurs Lock-out, en plus d'utilisateurs Prix. L'*infogérance open-source* est proposée par les ENL, comme Smile ou Linagora. Des spécialistes de l'infogérance intègrent aussi des logiciels libres dans la bibliothèque des technologies/logiciels qu'ils suivent, à l'image de ce que propose Atos ou du rachat par IBM de RedHat.

La servitisation désigne le basculement de la vente d'un produit à la vente d'une solution offrant une valeur d'usage (Foss et Saebi, 2017). Elle correspond à une tendance de fond dans le secteur IT avec les offres cloud (Plouin, 2016) de logiciels en mode SaaS accessibles depuis un simple navigateur sans besoin d'installation au sein du système informatique du client. Du côté du prestataire, les actifs spécifiques sont dans ce cas des ressources humaines permettant l'optimisation technique de l'application (Bezemer et Zaidman, 2010), mais aussi l'administration système (Viseur, 2013). Dans le cas du SaaS, avec le développement d'une application réellement multitenant, l'entreprise bénéficie d'un flux récurrent d'abonnements unitaires faibles, mais aussi d'économies d'échelle importantes grâce à l'extrême standardisation du code source déployé et des machines gérées (Bezemer et Zaidman, 2010 ; Plouin, 2016). La spécificité

du logiciel libre vient encore de la possibilité de doser son niveau d'investissement dans le suivi du logiciel. Comme pour l'infogérance, les modèles d'affaires appuyés sur un logiciel libre peuvent être de deux types. On peut avoir des acteurs qui déploient des solutions SaaS basées un logiciel libre vers des clients KM, voire naïfs, avec des besoins Prix ou Lean, une tâche pour laquelle l'éditeur est idéalement positionné. C'est, par exemple, le cas pour les solutions d'ERP open-source comme Odoo, mais aussi d'infrastructures de gestion de projet comme GitHub ou GitLab. Nous parlerons alors de *servitisation open-source*. On peut aussi avoir des gestionnaires de plates-formes qui intègrent des solutions libres dans leur bibliothèque d'offre, comment peuvent le proposer Amazon (AWS), Microsoft (Azure), ou OVH.

Pour les mêmes raisons, on va assister aux mêmes phénomènes sur le développement et l'installation des modules métiers libres. Évidemment, leur passage en phase 3 peut être décalé par rapport au cœur du logiciel. Dans ce cas, les modèles vus en phase 2 peuvent perdurer.

Mais quand ces modules sont devenus eux aussi suffisamment stables, des prestataires ayant une forte expertise métier peuvent les utiliser pour développer des offres, voire proposer plusieurs technologies, libres ou non, selon les demandes des clients. Leur expertise réside davantage dans la spécification des besoins, comme la re-conception des processus de travail, et le paramétrage de solutions stables que dans la maintenance de tecnologies évolutives. À nouveau, selon le degré d'évolution de la solution, le prestataire pourra maintenir une activité de contribution au projet, pour être plus rapide sur l'adaptation, ou simplement suivre les versions. S'il s'appuie uniquement sur une offre libre, on pourra parler de *spécialiste métier open-source*, mais il sera en concurrence avec d'autres spécialistes métiers, sur des besoins métiers plutôt que sur les spécificités open-sources.

Avec un cœur de logiciel qui est stable, l'éditeur de modules métiers peut, à coût réduit, choisir de développer une distribution spécialisée de la solution, afin de capter l'ensemble des revenus issus des prestations d'assurance, voire d'installation/adaptation au SI du client. Cela relève d'une stratégie de verticalisation classique dans l'industrie des SI (Fink et Markovich, 2008). Il peut chercher à rendre ces modules agnostiques vis-à-vis de la plate-forme. Le projet libre est alors délaissé au profit des développements internes même s'il conserve l'intérêt d'assurer une relative indépendance vis-à-vis de la structure, éditeur ou fondation, produisant le logiciel. On peut même avoir un éditeur métier open-source indépendant des plates-formes (voir par exemple l'extension antispam Askimet, développée par Automattic, l'entreprise éditrice du CMS Wordpress, portée sur les CMS Drupal ou Joomla). La focalisation sur les modules peut être facilitée par l'ouverture récente des logiciels aux développeurs tiers, comme les places de marché d'extensions (Gottschalk *et al.*, 2019). Cependant, le modèle d'affaires ne change pas fondamentalement.

4.2.5. Synthèse des modèles d'affaires open-sources

Il ressort de cette analyse huit modèles d'affaires open-sources, qui correspondent à différentes propositions et captations et valeur, et à différents types d'activités et ressources spécifiques, regroupés par types d'actif. Le Tableau 2 en propose la synthèse.

Ces modèles ne sont pas étanches les uns des autres, certaines capacités permettent de passer plus facilement de l'un à l'autre. Ils n'apparaissent pas non plus au même moment de la vie du projet logiciel. Enfin, plus le projet/produit logiciel libre devient mature, plus il peut être utilisé par des

Tableau 2 : Modèles d'affaires open-sources : captation de valeur et ressources.

	Personnalisation	**Éditeur**	**Intégrateur**	**Conseillers solution-métier**	**Éditeur métier**	**Spécialiste métier**	**Infogérance**	**Servitisation**
Phase(s)	1, 2	2,3	2,3	2,3	2, 3	2, 3	3	3
Besoin(s)	Contrôle	Lock-out, Lean	Lock-out, Lean	Lean, Contrôle	Lock-out, Lean	Lean	Lock-out, Prix	Prix
Captation de valeur								
Accès au logiciel stock (produit)	--	-	+	+	-	- -	-	+ +
Assurance (accès au logiciel flux)	-	+ +	+ +	+	++	- -	+	+
Assistance	+	+	+	+	+	+ +	+	-
Adaptation	+ +	- -	- -	+	-	+	-	- -
Actifs (ressources & capacités) clefs								
Humains (expertise)	++	-	-	+	-	++	+	- -
Humains (cte)	+	+ +	+ +	+	+ +	-	+	+
Cognitifs (base de connaissance)	+	+	+ +	+	+	-	+	+
Technologiques (code logiciel)	-	+ +	+ +	+	+	-	+	+
Organisationnels (infrastructure, si)	+	+ +	++	+	+	-	+	+ +
Réputationnels (marque)	+	+	+	+	+	+	+	+ +
Commerciaux (réseaux)	- -	+ +	++	+	+ +	+	+	+
Ressource projet logiciel libre								
Logiciel-stock	+	+	+	+	+	+	+	+
Logiciel-flux	+ +	+ +	+ +	+	-	-	+	+
Notoriété	-	+ +	+ +	+	-	- -	+	+

Légende : - - (très peu important), - (peu important), + (important), + + (très important) dans le modèle d'affaires open-source.

acteurs traditionnels des services informatiques comme une des solutions parmi celles qu'ils proposent.

5. CONCLUSION

5.1. Des modèles d'affaires open-sources contingents de la dynamique des projets libres

Grace au cadre formel du modèle économique, nous avons pu montrer que dans les modèles open-sources, la ressource spécifique à laquelle on doit avoir accès est un droit (de contribution, de gestion du projet...) Pour pouvoir l'exercer, il faut contribuer pour développer ses capacités d'absorption (Cohen et Levinthal, 1990), mais surtout pour être accepté par le projet, et sa « communauté » (Dahlander et Wallin, 2006). L'effort de contribution est coûteux, d'autant plus coûteux qu'on est au cœur du projet. Il permet aussi, comme expliqué par Jullien et Roudaut (2020), un meilleur contrôle des ressources rivales du projet (le droit de donner son avis sur l'orientation du projet, le droit de gérer un module), rivales, car les équipes sont maintenues petites (Mockus *et al.*, 2002), donc n'importe qui ne peut pas y avoir accès.

À chaque niveau de droit correspond une capacité, difficilement imitable, et une proposition de valeur monétisable : 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 permet 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 permet 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'intégrateur (assurance). Finalement, les capacités d'aliénation passent surtout par le contrôle d'actifs immatériels dynamiques : la marque (protégée par le droit de propriété intellectuelle, mais qui n'a de valeur que si le projet est reconnu et est dynamique), mais aussi des bases de données (de bugs, par exemple, qu'il faut constamment mettre à jour), protégées par le secret. Ceux-ci sont clefs dans les modèles basés sur l'assurance.

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 est organisé autour d'un écosystème et non plus autour d'une entreprise. De plus, 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). Quand la dynamique du projet s'épuise, les modèles d'affaires se rapprochent des modèles classiques et d'autres ressources clefs que la participation au projet deviennent primordiales, laissant la place à d'autres acteurs.

5.2. Des enseignements pour le praticien

Notre analyse montre trois choses. Premièrement, les modèles open-sources sont d'abord des modèles de prestation de services (assistance, adaptation autour de capacités humaines d'expertise) et d'opération de services (assurance et accès à une capacité technique entretenue). Deuxièmement, le choix du modèle de

coordination de la création de valeur (l'éditeur du projet), entre entreprise privée, fondation, ou plate-forme, dépend de qui est influent aux débuts du projet et mériterait d'être approfondi. Être influent nécessite des investissements importants qui vont être rentabilisés par le contrôle accru sur la feuille de route et les facilités d'accès au répertoire de code source. Ce sont des investissements en capacités humaines, notamment : déploiement de ressources ou recrutement de contributeurs influents, voire dans des cas extrêmes, fork pour redéfinir une gouvernance alignée sur les intérêts de l'entreprise, comme Dokeos, OpenBravo… Troisièmement, le choix de la structure chargée de la coordination du projet libre contraint les stratégies open-sources possibles du fait de l'alignement nécessaire entre modèle d'affaires et gouvernance du projet.

Les acteurs présents dès l'origine d'un projet sont les mieux placés pour capter la valeur dans chaque phase, notamment dans les stratégies d'éditeur open-source, car ils contrôlent le processus de développement. Il existe ainsi des parcours entre modèles d'une phase à une autre, une forme de dépendance au sentier (Teece *et al.*, 1997). Cependant, il s'agit toujours d'une position précaire, un monopole contestable, non par le prix, mais sur la position dans le projet, pour deux raisons D'une part, si un acteur est vu comme trop envahissant par la communauté, ou ne respectant pas assez les besoins des utilisateurs VH, ceux-ci peuvent se détourner de la technologie ou développer un projet parallèle, un fork (Viseur et Charleux, 2019 ; Viseur, 2012). Cette sorte de « main invisible » contribue à la durabilité du projet, en l'obligeant l'éditeur à écouter sa communauté (Nyman *et al.*, 2012). D'autre part, comme les compétences clefs ne sont pas tout à fait les mêmes à chaque étape, il faut que l'acteur ait les capacités financières et dynamiques de s'adapter.

L'itinéraire consistant à traverser les 3 phases en spécialisant progressivement son modèle d'affaires n'est pas non plus figé. D'abord, il n'y a pas nécessairement de prestataires dans l'écosystème d'un projet libre. C'est notamment le cas dès lors que le logiciel a un positionnement essentiellement grand public ou entraîne peu de besoins complémentaires. Ensuite, un prestataire peut sauter une phase, en forkant un projet ou en bénéficiant d'un investissement lui permettant de sauter une phase. Enfin, à chaque phase, de nouveaux acteurs possédant déjà des capacités métiers peuvent capter une partie de la valeur en s'impliquant dans le développement, et en diminuant ainsi l'avantage concurrentiel des firmes déjà présentes. Ce n'est pas forcément une mauvaise chose pour l'écosystème et son modèle économique, mais cela peut expliquer l'instabilité des acteurs privés qui gravitent autour d'un projet libre et de leur modèle d'affaires.

Les entreprises open-sources peuvent aussi participer à plusieurs projets libres, et déployer différents modèles d'affaires suivant les projets. C'est d'autant plus facile que les activités et ressources clefs sont proches. Certains modèles peuvent être complémentaires (éditeur du logiciel et éditeur de modules métiers, par exemple), car ils font appel aux mêmes types de compétences (gestion d'une capacité technique entretenue dans l'exemple), ou répondent à des besoins clients complémentaires.

5.3. Voies de recherche

En synthèse, même si les exemples mobilisés permettent d'illustrer ce travail essentiellement théorique, il devra faire l'objet d'une confrontation au terrain au travers d'études de cas longitudinales de prestataires. Ces dernières sont envisageables de deux manières. D'une part, le secteur open-source est riche de nombreuses entreprises

créées voici maintenant parfois plus de 10 ans (sur le marché francophone européen, citons par exemple Odoo, Dokeos, Linagora, Nexedi, Alterway). D'autre part, des entreprises continuent à se créer, permettant d'étudier la spécialisation progressive de leur(s) modèle(s) d'affaires et de documenter les questions, les incertitudes, qui se présentent au fur et à mesure de cette structuration.

RÉFÉRENCES

Acemoglu, D. et Pischke, J.-S. (1998). Why Do Firms Train? Theory and Evidence. *Quarterly Journal of Economics*, 113(1):79–119.

Adner, R. et Kapoor, R. (2010). Value creation in innovation ecosystems: How the structure of technological interdependence affects firm performance in new technology generations. *Strategic management journal,* 31(3):306–333.

Ãgerfalk, P. J. et Fitzgerald, B. T. (2007). Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy. *MIS Quarterly*, 32(2):385–409.

Almeida, F., Oliveira, J. et Cruz, J. (2011). Open standards and open source: enabling interoperability. *International Journal of Software Engineering & Applications* (IJSEA), 2(1):1–11.

Alvesson, M. (2000). Social indentity and the problem of loyalty in knowledge-intensive companies. *Journal of management studies,* 37(8):1101–1124.

Amit, R. et Zott, C. (2001). Value creation in e-business. *Strategic management journal*, 22(6-7):493–520.

Andersson, F., Freedman, M., Haltiwanger, J., Lane, J. et Shaw, K. (2009). Reaching for the Stars: Who Pays for Talent in Innovative Industries? *Economic Journal*, 119(538):F308–F332.

Arrègle, J.-L. (1996). Analyse resource based et identification des actifs stratégiques. *Revue française de gestion*, pages 25–36.

Arthur, W. B. (1989). Competing technologies, increasing returns and lock-in by historical events: The dynamics of allocations under increasing returns to scale. *Economic Journal,* 99:116–131.

Arthur, W. B. (1994). Increasing returns and Path dependance in the Economy. University of Michigan Press.

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

Bach, L., Cohendet, P., Pénin, J. et 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):59–72.

Baden-Fuller, C. et Mangematin, V. (2013). Business models: A challenging agenda. *Strategic Organization*, 11(4):418–427.

Baldwin, C. Y. et Clark, K. B. (2003). The architecture of cooperation: How code architecture mitigates free riding in the open source development model.

Bessen, J. (2006). Open source software: Free provision of complex public goods. In *The economics of open source software development* (pp. 57-81). Elsevier.

Bezemer, C.-P. et Zaidman, A. (2010). Multi-tenant saas applications: maintenance dream or nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution* (IWPSE), pages 88–92. ACM.

Boehm, B. et Abts, C. (1999). Cots integration: Plug and pray? *Computer*, 32(1):135–138.

Boissin, O. (1999). La construction des actifs spécifiques : une analyse critique de la théorie des coûts de transaction. *Revue d'économie industrielle*, 90(1):7–24.

Bonaccorsi, A., Giannangeli, S. et Rossi, C. (2006). Entry strategies under competing standards: Hybrid business models in the open source software industry. *Management Science Volume*, 52(7): 1085–1098.

Bonaccorsi, A. et Rossi, C. (2003). Why open source software can succeed. *Research Policy,* 32(7):1243–1258.

Brosgol, B. M. (2019). How to succeed in the software business while giving away the source

code: The AdaCore experience. *IEEE Software,* 36(6):17–22.

Campbell-Kelly, M. et Garcia-Swartz, D. D. (2015). From mainframes to smartphones: A history of the international computer industry. Harvard University Press.

Capiluppi, A., Lago, P. et Morisio, M. (2003). Characteristics of open source projects. In *Seventh European Conference onSoftware Maintenance and Reengineering, 2003. Proceedings*, pages 317–327. IEEE.

Carillo, K. et Okoli, C. (2011). Generating quality open content: A functional group perspective based on the time, interaction, and performance theory. *Information & Management,* 48(6):208–219.

Charleux, A. et Mione, A. (2018). Les business models de l'édition open source ; le cas des logiciels. Finance Contrôle Stratégie, (NS-1).

Cheruy, C., Robert, F. et Belbaly, N. (2017). Oss popularity: understanding the relationship between user-developer interaction, market potential and development stage. *Systèmes d'information et management,* 22(3):47–74.

Chesbrough, H., Lettl, C. et Ritter, T. (2018). Value creation and value capture in open innovation. *Journal of Product Innovation Management,* 35(6):930–938.

Cohen, W. M. et Levinthal, D. A. (1990). Absorptive capacity, a new perspective of learning and innovation. *Administrative Science Quarterly,* 35:128–152.

Conner, K. R. (1991). A historical comparison of resource-based theory and five schools of thought within industrial organization economics: do we have a new theory of the firm? *Journal of management,* 17(1):121–154.

Crowston, K. et Howison, J. (2005). The social structure of Free and Open Source Software development. First Monday, 10(2).

Crowston, K. et Scozzi, B. (2002). Open source software projects as virtual organisations: competency rallying for software development. *IEE Proceedings-Software,* 149(1):3–17.

Cusumano, M. (2004). The Business of software: what every manager, programmer, and entrepreneur must know to thrive and survive in good times and bad. Free Press, New York.

Dahlander, L. et Magnusson, M. G. (2005). Relationships Between Open Source Software Companies and Communities: Observations from Nordic Firms. *Research Policy,* 34:481–493.

Dahlander, L. et Magnusson, M. G. (2008). How do firms make use of open source communities? *Long Range Planning,* 41 (2008):629–649.

Dahlander, L. et Wallin, M. W. (2006). A man on the inside: Unlocking communities as complementary assets. *Research Policy,* 35:1243–1259.

De Bandt, J. (1998). Les marchés de services informationnels : quelles garanties pour le client, consommateur ou partenaire ? *Revue d'économie industrielle*, 86, 4e trimestre:61–84.

De Laat, P. B. (2005). Copyright or copyleft? An analysis of property regimes for software development. *Research Policy,* 34(10):1511–1532.

Demil, B. et Lecocq, X. (2010). Business model evolution: in search of dynamic consistency. *Long range planning,* 43(2-3):227–246.

Dierickx, I. et Cool, K. (1989). Asset stock accumulation and sustainability of competitive advantage. *Management science,* 35(12): 1504–1511.

Douard, J.-P. et Heitz, M. (2003). Une lecture des réseaux d'entreprises : prise en compte des formes et des évolutions. *Revue française de gestion,* (5):23–41.

Duvall, P., Matyas, S. et Glover, A. (2007). Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley Signature Series. Pearson Education.

Ethiraj, S. K., Kale, P., Krishnan, M. S. et 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):25–45.

Fautrero, V. et Gueguen, G. (2012). Quand la domination du leader contribue au déclin. *Revue française de gestion*, (3):107–121.

Fink, L. et Markovich, S. (2008). Generic verticalization strategies in enterprise system markets: An exploratory framework. *Journal of Information Technology,* 23(4):281–296.

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly*, 30(3):587–598.

Foss, N. J. et Saebi, T. (2017). Fifteen years of research on business model innovation: how far have we come, and where should we go? *Journal of Management,* 43(1):200–227.

Franco-Bedoya, O., Ameller, D., Costal, D. et Franch, X. (2017). Open source software ecosystems: A systematic mapping. *Information and software technology*, 91:160–185.

Franke, N. et Von Hippel, E. (2003). Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software. *Research policy*, 32(7): 1199–1215.

Gabay, J. (2014). Maîtrise d'ouvrage des projets informatiques - 3e éd. : Guide pour le chef de projet MOA. Management des systèmes d'information. Dunod.

Gadray, J. (2003). Socio-économie des services. coll. Repères, La Découverte, Paris.

Gawer, A. et Cusumano, M. A. (2002). Platform Leadership. Harvard Business School Press, Boston, MA, USA.

Germonprez, M., Levy, M., Kendall, J. E. et Kendall, K. E. (2020). Tapestries of innovation: Structures of contemporary open source project engagements. *Journal of the Association for Information Systems*, 21(3):5.

Goldenberg, J., Libai, B. et Muller, E. (2002). Riding the saddle: How cross-market communications can create a major slump in sales. *Journal of Marketing*, 66(2):1–16.

Gonzalez, R. V. D. (2016). Knowledge retention in the service industry. *International Journal of Knowledge Management* (IJKM), 12(1):45–59.

Gottschalk, S., Rittmeier, F. et Engels, G. (2019). Business models of store-oriented software ecosystems: a variability modeling approach. In *International Symposium on Business Modeling and Software Design,* pages 153–169.

Grant, R. M. (1991). The resource-based theory of competitive advantage: implications for strategy formulation. *California Management Review,* 33(3):114–135.

Gueguen, G. et Torres, O. (2004). La dynamique concurrentielle des écosystèmes d'affaires. *Revue française de gestion*, (1):227–248.

Hannan, M. T. et Freeman, J. (1984). Structural inertia and organizational change. *American sociological review*, pages 149–164.

Hauge, Ø., Ayala, C. et Conradi, R. (2010). Adoption of open source software in software-intensive organizations - a systematic literature review. *Information and Software Technology,* 52:1133–1154.

Horn, F. (2004). L'économie des logiciels. Repères, la Découverte.

Howison, J. et Crowston, K. (2014). Collaboration Through Open Superposition: A Theory Of The Open Source Way. *MIS Quarterly*, 38(1):29–50.

Izquierdo-Cortazar, D., Gonzalez-Barahona, J. M., Dueñas, S. et Robles, G. (2010). Towards automated quality models for software development communities: The qualoss and flossmetrics case. In *7th IEEE International Conference on the Quality of Information and Communications Technology* (QUATIC), page 364–369, Porto, Portugal.

Jansen, S. (2014). Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, 56(11):1508–1519.

Jarke, M., Loucopoulos, P., Lyytinen, K., Mylopoulos, J. et Robinson, W. (2011). The brave new world of design requirements. *Information Systems,* 36(7):992–1008.

Jayanth, R., Jacob, V. S. et Radhakrishnan, S. (2011). Vendor and client interaction for requirements assessment in software development: Implications for feedback process. *Information Systems Research,* 22(2):289–305

Jousset, G. (2005). L'expérience de l'infogérance. *Entreprises et histoire*, (3):46–48.

Jullien, N. et Pénin, J. (2014). Innovation ouverte : vers la génération 2.0. In Denis, J.-P., Hafsi, T., Martinet, A. C. et Tannery, F., éditeurs: *Encyclopédie de la Stratégie.* Economica.

Jullien, N. et Roudaut, K. (2012). Can open source projects succeed when the producers are not users? lessons from the data processing field. *Management international*, 16:113–127.

Jullien, N. et Roudaut, K. (2020). Commun numérique de connaissance : définition et conditions d'existence. *Innovations*, 3(63):I80–XXV.

Jullien, N., Stol, K.-J. et Herbsleb, J. D. (2019). A preliminary theory for open-source ecosystem microeconomics. In Fitzgerald, B., Mockus, A. et Zhou, M., éditeurs : *Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability*, pages 49–68. Springer.

Jullien, N. et Zimmermann, J.-B. (2002). Le logiciel libre : une nouvelle approche de la propriété intellectuelle ? *Revue d'économie industrielle*, 99:159–178.

Jullien, N. et Zimmermann, J.-B. (2006). Peut-on envisager une écologie du logiciel libre favorable aux nuls? *Terminal*, 97-98:33–47.

Jullien, N. et Zimmermann, J.-B. (2009). Firms' contribution to open-source software and the dominant user's skill. *European Management Revue*, 6:130–139.

Jullien, N. et Zimmermann, J.-B. (2011a). Floss firms, users and communities: a viable match? *Journal of Innovation Economics,* 1(7):31–53.

Jullien, N. et Zimmermann, J.-B. (2011b). Floss in an industrial economics perspective. *Revue d'économie industrielle,* 136:39–64.

Karpik, L. (1996). Dispositifs de confiance et engagements crédibles. *Sociologie du travail*, 4/96:527–550.

Koch, S. (2011). Organisation of work in open source projects: expended effort and efficiency. *Revue d'économie industrielle*, 136:17–38.

Kogut, B. et Zander, U. (1992). Knowledge of the firm, combinative capabilities, and the replication of technology. *Organization science*, 3(3):383–397.

Kogut, B. M. et Metiu, A. (2001). Open source software development and distributed innovation. *Oxford Review of Economic Policy*, 17(2):248–264.

Krishnamurthy, S. (2005). Cave or community? an empirical examination of 100 mature open source projects (originally published in volume 7, number 6, june 2002). First Monday.

Kruchten, P., Nord, R. L. et Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21.

Laffan, L. (2012). A new way of measuring openness: The open governance index. *Technology Innovation Management Review*, 2(1).

Lakhani, K. et Von Hippel, E. (2003). How open source software works: Free user to user assistance. *Research Policy,* 32:923–943.

Lakhani, K. et Wolf, R. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In in feller *et al.* (2005), éditeurs : *Perspectives on free and open source software*, pages 3–22.

Lazear, E. et Oyer, P. (2013). Personnel Economics. In gibbons, R. et roberts, D. J., éditeurs : *Handbook of Organizational Economics.* Princeton University Press.

Le Texier, T. et Versailles, D. W. (2009). Open source software governance serving technological agility: the case of open source software within the dod. *International Journal of Open Source Software and Processes* (IJOSSP), 1(2):14–27.

Lecocq, X., Demil, B. et Ventura, J. (2010). Business models as a research program in strategic management: an appraisal based on Lakatos. *M@n@gement*, 13(4):214–225.

Lee, S., Baek, H. et Jahng, J. (2017). Governance strategies for open collaboration: Focusing on resource allocation in open source software development organizations. *International Journal of Information Management,* 37(5):431–437.

Lee, S. et Lee, H. (2004). The importance of change management after ERP implementation: an information capability perspective. *ICIS 2004 Proceedings*, page 76.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076.

Lepak, D. P., Smith, K. G. et Taylor, M. S. (2007). Value creation and value capture: A multilevel perspective. *Academy of management review*, 32(1):180–194.

Lerner, J. et Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics,* 50:197–234.

Lisein, O., Pichault, F. et Desmecht, J. (2009). Les business models des sociétés de services actives dans le secteur open source. *Systèmes d'information et management,* 14(2).

Maccormack, A., Rusnak, J. et Baldwin, C. Y. (2006). Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52(7):1015–1030.

March, J. G. (1991). Exploration and exploitation in organizational learning. *Organization science,* 2(1):71–87.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J. et Ghalsasi, A. (2011). Cloud computing – The business perspective. *Decision support systems*, 51(1):176–189.

Marwell, G. et Oliver, P. (1993). The Critical Mass in Collective Action: A Micro-Social Theory. Cambridge University Press, Cambridge.

Massa, L., Tucci, C. L. et Afuah, A. (2017). A critical assessment of business model research. *Academy of Management Annals*, 11(1):73–104.

Maucuer, R. et Renaud, A. (2019). Business Model Research: A Bibliometric Analysis of Origins and Trends. *M@n@gement*, 22(2):176–215.

Meissonier, R., Bourdon, I., Houze, E., Amabile, S. et Boudrandi, S. (2010). Understanding Open Source Developers' Motivations from their Participation. *Systèmes d'Information et Management* (French Journal of Management Information Systems), 15(2):71–97.

Messerschmitt, D. G. et Szyperski, C. (2001). Industrial and economic properties of software: Technology, processes, and value. Rapport technique MSR-TR-2001-11.

Millier, P. (1997). Stratégie et marketing de l'innovation technologique. Dunod.

Mockus, A., Fielding, R. T. et 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):309–346.

Money, L. P., Praseetha, S. et Mohankumar, D. (2012). Open source software: quality benefits, evaluation criteria and adoption methodologies. *Journal of Computations & Modelling,* 2(3):1–16.

Morgan, L. et Finnegan, P. (2014). Beyond free software: An exploration of the business value of strategic open source. *The Journal of Strategic Information Systems*, 23(3):226–238.

Morris, T. (2001). Asserting property rights: Knowledge codification in the professional service firm. *Human relations*, 54(7):819–838.

Mouakhar, K. et Benkeltoum, N. (2020). Capacité d'absorption des entreprises de l'open source : du modèle d'affaires à l'intention d'affaires. *Systèmes d'Information et Management.* 25(1), 47-88.

Mouakhar, K. et Tellier, A. (2013). How combine market and non-market: an empirical taxonomy of OSSC' strategic behaviors. *Systèmes d'Information et Management,* 18(3).

Narayanan, S., Balasubramanian, S. et Swaminathan, J. M. (2009). A matter of balance: Specialization, task variety, and individual learning in a software maintenance environment. *Management science,* 55(11):1861–1876.

North, K. et Kumta, G. (2018). Knowledge management: Value creation through organizational learning. Springer.

Nyman, L., Mikkonen, T., Lindman, J. et Fougère, M. (2012). Perspectives on code forking and sustainability in open source software. In Hammouda, I., Lundell, B., Mikkonen, T. et Scacchi, W., éditeurs: IFIP International Conference on Open Source Systems, pages 274–279, Berlin, Heidelberg. Springer.

O'mahony, S. et Ferraro, F. (2007). The emergence of governance in an open source community. *Academy of Management Journal,* 50:1059–1106.

Osterloh, M. et Rota, S. (2007). Open source software development. Just another case of collective invention? *Research Policy,* 36(2):157–171.

Osterwalder, A. et Pigneur, Y. (2010). Business model generation: a handbook for visionaries, game changers, and challengers. John Wiley & Sons.

Ouriques, R. A. B., Wnuk, K., Gorschek, T. et Svensson, R. B. (2019). Knowledge management strategies and processes in agile software development: a systematic literature review. *International journal of software engineering and knowledge engineering*, 29(03):345–380.

Peaucelle, J.-L. (1997). Informatique rentable et mesure des gains. Hermès.

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

Petrinja, E., Sillitti, A. Et Succi, G. (2010). Comparing openbrr, qsos, and omm assessment models. In Ågerfalk, P., Boldyreff, C., González-Barahona, J. M., Madey, G. R. et Noll, J., éditeurs: *Open Source Software: New Horizons*, pages 224–238, Berlin, Heidelberg. Springer.

Plouin, G. (2016). Cloud computing, 4e ed: Sécurité, gouvernance du SI hybride et panorama du marché. InfoPro. Dunod.

Poo-Caamaño, G. et German, D. M. (2015). The right to a contribution: An exploratory survey on how organizations address it. In *IFIP International Conference on Open Source Systems*, pages 157–167. Springer.

Popp, K. M. (2019). Best Practices for Commercial Use of Open Source Software: Business Models. BoD–Books on Demand.

Porter, M. E. *et al.* (1996). What is strategy? *Harvard business review,* 74(6):61–78.

Rahman, M. T., Querel, L.-P., Rigby, P. C. et Adams, B. (2016). Feature toggles: practitioner practices and a case study. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 201–211. ACM.

Riehle, D. et Berschneider, S. (2012). A model of open source developer foundations. In Hammouda, I., Lundell, B., Mikkonen, T. et Scacchi, W., éditeurs: *Open Source Systems: Long-Term Sustainability*, pages 15–28, Berlin, Heidelberg. Springer.

Ries, E. (2012). Lean Startup: Adoptez l'innovation continue. Village Mondial. Pearson.

Rochet, J.-C. et Tirole, J. (2006). Two-sided markets: a progress report. *The RAND journal of economics*, 37(3):645–667.

Rogers, E. M. (1983). *Diffusion of Innovations*. New York: The Free Press, 3rd édition.

Romer, P. M. (1993). Implementing a national technology strategy with self-organizing industry investment boards. *Brookings Papers on Economic Activity. Microeconomics*, 1993(2):345–399.

Ross, J. W., Beath, C. M. et Goodhue, D. L. (1996). Develop long-term competitiveness through it assets. *Sloan management review,* 38(1):31–42.

Santos, C., Kuk, G., Kon, F. et Pearson, J. (2013). The attraction of contributors in free and open source software projects. *The Journal of Strategic Information Systems,* 22(1):26–45.

Schaarschmidt, M., Walsh, G. et von Kortzfleisch, H. F. (2015). How do firms influence open source software communities? a framework and empirical analysis of different governance modes. *Information and Organization*, 25(2):99–114.

Shah, S. K. (2006). Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science*, 52(2):1000–1014.

Shahrivar, S., Elahi, S., Hassanzadeh, A. et Montazer, G. (2018). A business model for commercial open source software: A systematic literature review. *Information and Software Technology,* 103:202–214.

Shaikh, M. et Cornford, T. (2011). Total cost of ownership of open source software: a report for the UK Cabinet Office supported by OpenForum Europe.

Spinellis, D. et Szyperski, C. (2004). How is open source affecting software development? *IEEE software*, 21(1):28.

Spithoven, A., Clarysse, B. et Knockaert, M. (2010). Building absorptive capacity to organise inbound open innovation in traditional industries. *Technovation*, 30(2):130–141.

Stol, K.-J., Avgeriou, P., Babar, M. A., Lucas, Y. et Fitzgerald, B. (2014). Key factors for adopting inner source. *ACM Transactions on Software Engineering and Methodology* (TOSEM), 23(2):18.

Teece, D., Pisano, G. et Shuen, A. (1997). Dynamic capabilities and strategic management. *Strategic Management Journal*, 18(7):509–533.

Teece, D. J. (2010). Business models, business strategy and innovation. *Long range planning,* 43(2-3):172–194.

Teece, D. J. (2018). Business models and dynamic capabilities. *Long Range Planning*, 51(1):40–49.

Teigland, R., Gangi, P. M. D., Flåten, B.-T., Giovacchini, E. et Pastorino, N. (2014). Balancing on a tightrope: Managing the boundaries of a firm-sponsored OSS community and its impact on innovation and absorptive capacity. *Information and Organization,* 24(1):25–47.

Teixeira, J., Mian, S. et Hytti, U. (2016). Cooperation among competitors in the open-source arena: The case of Openstack. In *International Conference on Information Systems*, Dublin, Ireland.

Tirole, J. (1989). *The Theory of Industrial Organization.* MIT Press, Cambridge, MA.

Valimaki, M. (2003). Dual licensing in open source software industry. *Systèmes d'Information et Management*, 8(1):63–75.

Van Den Bosch, F. A., Volberda, H. W. et De Boer, M. (1999). Coevolution of firm absorptive capacity and knowledge environment: Organizational forms and combinative capabilities. *Organization Science*, 10(5):551–568.

Van Der Linden, F., Lundell, B. et Marttiin, P. (2009). Commodification of industrial software: A case for open source. *IEEE software,* 26(4):77–83.

Ven, K., Verelst, J. et Mannaert, H. (2007). On the relationship between commoditization and open source software. In limayem, M., éditeur: *Proceedings of the 12th Conference of the Association of Information and Management*, Lausanne, Switzerland.

Vendome, C., Bavota, G., Di Penta, M., Linares-Vásquez, M., German, D. et Poshyvanyk, D. (2017). License usage and changes: a large-scale study on github. *Empirical Software Engineering,* 22(3):1537–1577.

Viseur, R. (2012). Forks impacts and motivations in free and open source projects. International *Journal of Advanced Computer Science and Applications*, 3(2):117–122.

Viseur, R. (2013). Identifying success factors for the mozilla project. In *IFIP International Conference on Open Source Systems* (pp. 45-60). Springer, Berlin, Heidelberg.

Viseur, R. et Charleux, A. (2019). Changement de gouvernance et communautés open source : le cas du logiciel Claroline. *Innovations*, (1):71–104.

Von Hippel, E. (1988). *The Sources of Innovation.* Oxford University Press, New York.

Von Hippel, E. et Von Krogh, G. (2003). Open source software and the "private-collective" innovation model: Issues for organization science. *Organization Science*, 14(2):209–223.

Von Krogh, G., Haefliger, S., Spaeth, S. et Wallin, M. W. (2012). Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *MIS Quarterly,* 36(2):649–676.

Walterbusch, M., Martens, B. et Teuteberg, F. (2013). Evaluating cloud computing services from a total cost of ownership perspective. *Management Research Review*, 36(6):613–638.

West, J. (2003). How open is open enough? Melding proprietary and open source platform strategies. *Research Policy,* 32 (7):1259–1285.

West, J. et Gallagher, S. (2006). Challenges of open innovation: the paradox of firm investment in open-source software. *R&D Management,* 36(3):319–331.

Ye, Y. et Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering*, pages 419–429. IEEE.

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

Young, R. (1999). Giving it away: How RedHat software stumbled across a new economic model and helped improve an industry. *Journal of Electronic Publishing*, 4(3).

Zahra, S. A. et George, G. (2002). Absorptive capacity: A review, reconceptualization, and extension. *Academy of Management Review*, 27(2):185–203.

Zott, C., Amit, R. et Massa, L. (2011). The business model: recent developments and future research. *Journal of Management*, 37(4):1019–1042.

ANNEXE. TABLEAU 3 : RAPPORT ENTRE TCO ET SEGMENTATION DES BESOINS.

Étapes du TCO		Prix	Lock-out	Lean	Contrôle
	Exploration	Investissement minimal dans la recherche. Test à coût nul ou faible.	Investissement minimal dans la recherche. Test à coût nul ou faible. Indépendance au fournisseur recherchée.	Travail de définition des besoins fonctionnels. Évaluation multi-critère. Horizon court terme / moyen terme.	Travail de définition des besoins fonctionnels. Évaluation multi-critère. Horizon court terme / moyen terme / long terme (contrainte de pérennité). Contrôle de la solution.
		➠ Choix limité aux solutions connues du marché.	➠ Choix limité aux solutions connues du marché, mais via standard, concurrence ou licence ouverte.	➠ Exploration de différentes solutions, critère important de l'adaptabilité fonctionnelle.	➠ Exploration de différentes solutions. Critères importants de l'adaptabilité fonctionnelle et de l'indépendance au fournisseur.
	Acquisition	Achat d'un produit. Pas d'adaptation.	Achat d'un produit. Pas d'adaptation.	Capacité d'adaptation en fonction des capacités financières.	Capacité d'adaptation plus poussée et fonction des capacités financières.
		➠ Recherche de coûts d'accès au produit faibles ou nuls.	➠ Recherche de coûts d'accès au produit, licence importante (non dépendance au fournisseur).	➠ Acquisition d'un progiciel + adaptation à coût maîtrisé (paramétrage, modules) => notion de sur-mesure de masse.	➠ Solution comme un processus autant qu'un produit. Progiciel avec modules spécifiques ou développement à façon. Licence importante (non dépendance au fournisseur).
	Intégration	Minimale.	Minimale.	Intégration à coût maîtrisé. Besoin de formation. Adaptation à la solution.	Intégration poussée. Solution adaptable aux besoins en permanence.
				➠ Formations standards recherchées.	➠ Formations pouvant être importantes.
	Utilisation	Pas de maintenance prévue.	Minimale. Mais avec assurance de bon fonctionnement. Veille en termes de sécurité / bugs (la panne doit être évitée absolument).	Pas forcément beaucoup d'évolution (faible dynamique des besoins), mais maintenance en condition (capacité technique entretenue).	Maintenance des extensions « maison ». Veille en matière de sécurité / bugs (la panne doit être évitée absolument) et d'évolution de la base technologique.
			➠ Intégration de contrats de maintenance avec SLA* ou service informatique dédiés.	➠ Intégration de contrats de maintenance éventuels a minima.	➠ Intégration de contrats de maintenance avec SLA* ou service informatique dédiés ➠ Compétences métier internalisées.
	Sortie ou modification (évolution & dépendance)	Pas anticipé (besoins standards). « One shot ».	Importance de la liberté. Critère anticipé.	Peu anticipé.	Idéal de parcours de plusieurs cycles avec la solution. Recherche de capitalisation.

* SLA : Service Level Agreement, ou niveau de garantie de service dans les contrats de services de maintenance sur le bon fonctionnement et les délais de réparation de panne.