10. Écriture et performativité du code source informatique

Stéphane COUTURE[1]

Résumé : Ce texte présente quelques analyses préliminaires résultant d'une étude doctorale en cours concernant le « code source informatique ». Le code source est l'objet de la programmation informatique et peut être défini comme l'ensemble des instructions qui décrivent le fonctionnement d'un logiciel ou d'un système informatique. En utilisant la notion de performativité, initialement développée en linguistique pragmatique mais maintenant mobilisée en sociologie des sciences et des techniques et dans certains travaux d'anthropologie de l'écriture, le texte cherche à articuler la performativité du code informatique à son insertion dans un système d'autorisations et de chaînes d'écritures.

[1] Doctorant en communication à l'Université du Québec à Montréal (steph@stephcouture.info).
La recherche doctorale sur laquelle ce texte s'appuie a bénéficié d'un appui financier du Conseil de recherche en sciences humaines du Canada (CRSH). L'auteur remercie Serge Proulx et Geneviève Szczepanik pour leurs commentaires et relectures.

***

La compréhension des « effets » ou de l' « impact » des technologies fait depuis longtemps l'objet de préoccupations en sciences sociales. Longtemps appréhendé sous la perspective d'un déterminisme technique dur où « la technique », dans son essence, impacterait le social, le rôle des technologies a été nuancé à la fois par la perspective d'un constructivisme social ou encore par la proposition que les artefacts, dans leur design même, ont une capacité prescriptive ou « font de la politique », pour reprendre l'expression de Langdon Winner (2002). Dans ce dernier type d'analyse, la catégorie du code, et en particulier du « code informatique », a été mobilisée à plusieurs reprises. Le juriste américain Lawrence Lessig, dans son ouvrage *Code 2.0*, propose par exemple de considérer le code informatique à la manière d'un code juridique qui prescrirait ou limiterait certains comportements humains (Lessig, 2006). Ainsi, ceux et celles qui écrivent le code informatique, les « code writers », définiraient la nature de l'Internet. L'objectif de Lessig, dans ce livre, est d'insister sur la nécessité d'une plus grande régulation de la part des pouvoirs publics du travail d'écriture du code : « How the code regulates, who the code writers are, and who controls the code writers these are questions on which any practice of justice must focus in the age of cyberspace » (Lessig, 2006, p. 79).

Dans une autre perspective, la sociologie des sciences et des techniques, et en particulier la théorie de l'acteur-réseau, s'est tournée récemment vers la notion de performativité pour appréhender cet effet des artefacts, ce qu'ils « font » ou « font faire ». Introduite par Austin (1975) en linguistique pragmatique puis réutilisée notablement par Butler (1997, 2004), la notion de performativité a d'abord été utilisée pour analyser des situations où « dire, c'est faire », comme l'indique le titre de la traduction française de l'ouvrage fondateur de Austin. Les nouvelles relectures de la notion de performativité pour appréhender la dimension matérielle permettent d'éviter une approche déterministe en

articulant l'effet des artefacts à un travail continuel et collectif de performation. Elles permettent également d'éviter le risque du « tout linguistique » ou du « tout culture », souvent reproché aux théories du performatif[2] de Austin ou Butler.

En utilisant la notion de performativité, ce texte cherche à articuler la force performative du code informatique – ou sa force « juridique », pour employer la métaphore de Lessig – à l'activité collective d'écriture du code. L'analyse s'appuie sur mon enquête doctorale qui concerne le « code source » informatique et qui prend pour terrains deux logiciels libres ou à code source ouvert (*open source*) développés publiquement et collectivement sur Internet et orientés vers la conception de sites web interactifs : les logiciels symfony et SPIP. L'analyse présentée dans ce texte se concentre cependant sur le seul cas du logiciel symfony. Après avoir présenté différents travaux qui ont déjà abordé implicitement ou explicitement cette performativité du code informatique (Latour, 1992; Mackenzie, 2005; Arns, 2005), je proposerai ma propre analyse en m'inspirant de certains travaux en anthropologie de l'écriture qui s'attardent notamment au travail et aux matérialités donnant aux écrits leurs forces performatives (Fraenkel, 2006 ; Denis et Pontille, 2010a).

Introduction : la performativité du code informatique

La notion de performativité a déjà été utilisée quelque fois pour appréhender l'action du code informatique. Dans *Aramis ou l'amour des techniques*, Bruno Latour réfère par exemple implicitement à l'ouvrage de Austin (*Quand dire c'est faire*) lorsqu'il affirme que les programmes informatiques et les puces « font ce qu'ils disent » :

[2] Cet argument est présenté de façon plus approfondie par Jérôme Denis (2006), en introduction d'un dossier sur les nouveaux visages de la performativité. Un numéro de Réseaux a également été consacré à la question de la performativité des artefacts (Licoppe, 2010).

> Les programmes s'écrivent, les puces se gravent comme des eaux-fortes, ou se photographient comme des plans. Et pourtant ils font ce qu'ils disent ? Mais oui, car tous, textes et choses, ils agissent. Ce sont des programmes d'action dont le scripteur délègue la réalisation tantôt à des électrons, tantôt à des signes, tantôt à des habitudes, tantôt à des neurones » (Latour, 1992, p. 182)

On retrouve explicitement l'idée d'une performativité du code informatique dans au moins deux articles. Dans un court article en art médiatique, Inke Arns propose d'utiliser la notion de performativité pour appréhender les conséquences politiques, esthétiques et sociales du code informatique : « This notion — borrowed from speech act theory — not only involves the ability to generate in a technical context, but also encompasses the implications and repercussions of code in terms of aesthetics, politics and society. » (Arns, 2005, p. 1). Sur le plan politique, l'auteure réfère à Lessig en écrivant que le code informatique devient la loi, en ce sens que sa « performativité codée » (« coded performativity ») a la capacité de mobiliser ou d'immobiliser les utilisateurs. La spécificité du code informatique, par rapport aux autres actes de langage décrits par Austin, est que les mots correspondent directement au faire : « it directly affects, and literally sets in motion, or even “kills”, a process » (Arns, 2005, p. 7). Citant Butler pour qui un énoncé n'est performatif que s'il produit un effet, Arns soutient que, sur le plan pragmatique, le code informatique n'est performatif que s'il produit effectivement un effet et s'il est exécutable. L'auteure nuance toutefois ce propos en indiquant que, dans le contexte de l'art logiciel, le code non-exécutable a aussi sa raison d'être et pourrait avoir une certaine force performative.

Se situant davantage dans le courant des *Cultural Studies,* Adrian Mackenzie (2005) mobilise la notion de performativité telle que développée par Butler (2004) pour analyser le code informatique. Pour Butler, si un énoncé performatif réussit, c'est parce qu'il « fait écho à des actions antérieures et *accumule la force de l'autorité à travers la répétition ou la citation d'un ensemble de pratiques antérieures qui*

font autorité » (Butler, 2004, p. 80[3]). S'appuyant sur cette approche, Mackenzie propose de considérer la programmation informatique comme une pratique continuelle de citation du code. Le code informatique tirerait sa force performative de la répétition de ce qu'il nomme l'*authorizing context* (que nous traduisons ici par « contexte d'autorisation ») qui inclurait un ensemble de pratiques, d'imaginaires et d'objets techniques davantage stabilisés. Mackenzie s'est attardé à l'analyse du cas de Linux, un système d'exploitation souvent qualifié de « clone » de Unix, un système d'exploitation plus ancien, développé dans les années 1970. Il remarque que le « Unix » dont Linux serait le clone n'est pas tant un logiciel précis aux frontières bien délimitées mais plutôt un ensemble de pratiques d'administration informatique, de programmation et de design logiciel quelques fois référées sous le terme de « philosophie Unix ». (Mackenzie, 2005, p. 84). Le nom LinuX, se terminant par la lettre X, est d'ailleurs une référence directe à cette philosophie UniX. L'auteur propose donc de considérer la « philosophie Unix » comme une partie du « contexte d'autorisation » qui sous-tend le développement de Linux[4].

Ces deux articles mettent de l'avant l'idée d'une performativité du code informatique, mais la prise en compte de l'activité d'*écrire* le code informatique et le caractère « écrit » de cet artefact restent relativement négligés dans ces analyses. Dans la suite de ce texte, j'aimerais proposer un développement de la performativité du code informatique qui s'attarde davantage au processus d'écriture et son lien avec la performativité.

Le code source informatique comme artefact écrit

[3] Les italiques sont dans le texte.

[4] Mackenzie rappelle que les pratiques de l'informatique qui forment ce « authorizing context » sont également articulées à des pratiques d'exclusion et de hiérarchisation, en particulier en ce qui concerne les rapports de sexe et de genre. Ainsi, dans son courriel qui annonçait la création de Linux, le créateur de ce logiciel faisait référence au temps mémorable où « les hommes étaient de vrais hommes qui programmaient leur propre gestionnaire de périphériques » (Mackenzie, 2005, p. 87). La performativité du code est donc également une performativité du genre.

Comme indiqué au début du texte, la présente analyse s'appuie sur mon étude doctorale qui prend comme terrains deux logiciels web : SPIP et symfony. Ces logiciels sont utilisés dans le fonctionnement de milliers de sites web tels que *Le Monde Diplomatique* et certains sites de La Poste française dans le cas de SPIP ou encore Daily Motion, Delicious ou Yahoo Bookmarks dans le cas de symfony. Tel qu'indiqué plus tôt, l'analyse se concentrera cependant ici sur le cas du logiciel symfony. L'enquête, d'inspiration ethnographique (Star, 1999), s'appuie sur l'analyse des traces de discussions disponibles sur les forums en ligne (mails, rapports de « bugs », commentaires et annotations au code source), sur une série d'entretiens semi-dirigés, de même que sur la participation ponctuelle, entre juin 2009 et mai 2010, à diverses conférences et rencontres organisées par les membres des projets étudiés.

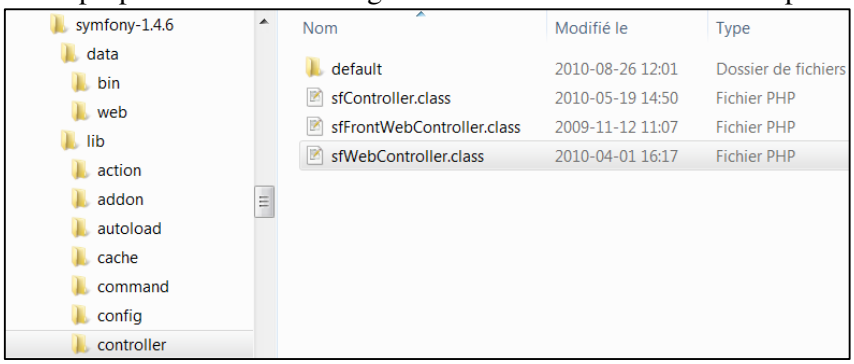
L'enquête doctorale concerne plus spécifiquement le « code source informatique ». Sans entrer ici dans les nuances, nous pouvons trivialement définir le code source informatique comme étant la forme écrite - et lisible humainement – du code informatique. Wikipédia définit par exemple le code source informatique de cette manière :

> Le code source (ou les sources voire le source) est un ensemble d'instructions écrites dans un langage de programmation informatique de haut niveau, compréhensible par un être humain entraîné, permettant d'obtenir un programme pour un ordinateur.[5]

Lorsqu'interrogés en entrevue sur la définition du « code source », les réponses des acteurs et actrices sont souvent très générales, métaphoriques. Je n'aborderai pas ici ces réponses générales, que je j'explore par ailleurs dans ma thèse. Toutefois, quand les questions se situent plus précisément sur ce qui constitue le code source des projets étudiés, l'objet référé est cette fois-ci plus concret. Ainsi, dans une entrevue, une actrice impliquée dans le projet symfony mentionne : « Le code source de symfony, c'est celui que je peux télécharger pour

[5] <http://fr.wikipedia.org/w/index.php?title=Code_source&oldid=52775534> (version du 30 avril 2010).

l'avoir chez moi et pour après l'utiliser » (sf05, 9 mars 2010), tandis qu'un autre acteur nous indique que « quand on télécharge symfony, on télécharge les sources de symfony » (sf07, 3 avril 2010). Ce « code source de symfony » référé par ces acteurs désigne en ce moment un ensemble d'environ 2 000 fichiers organisés en plus de 800 dossiers, et comprenant au total un peu plus de 300 000 lignes d'instruction, ou de code proprement dit. Les figures suivantes montrent des captures



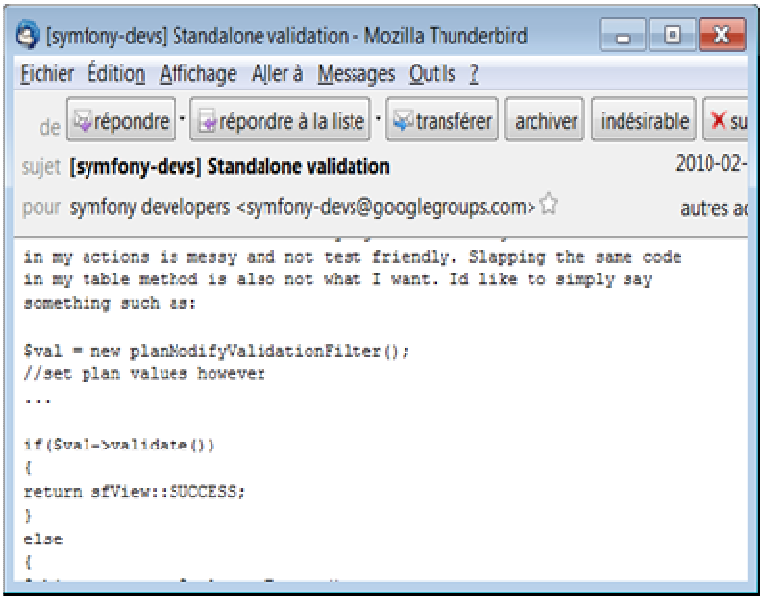
d'écran des dossiers et fichiers de symfony (10.1), ainsi que de quelques instructions (10.2) qui forment le code source de symfony.

```
29 class sfMySQLDatabase extends sfDatabase
30 {
31   /**
32    * Connects to the database.
33    *
34    * @throws <b>sfDatabaseException</b> If a connection could not be created
35    */
36   public function connect()
37   {
```

Figure 10.1. *Fichiers et dossiers du code source de symfony*

Figure 10.2. *Lignes du code source de symfony*

Il est toutefois important d'insister sur le fait que ce « code source » ne constitue que la version « stable » du code, disponible au téléchargement. Autour de ce « code source de symfony » existe une multitude d'espaces, associés à symfony, où des morceaux de code source sont publiés et discutés. Par exemple, le projet symfony répertorie environ 900 « *plugins* » dont on retrouve le code source sur le site web. Certaines parties du code source sont également constamment discutées sur des listes de discussion, un canal IRC[6], de même que sur de nombreux blogues et autres sites web. La figure 10.3 présente par exemple une capture d'écran d'un courriel où l'auteur propose une ébauche de code, en vue de recevoir des commentaires. La figure 10.4, quant à elle, présente une capture d'écran d'un répertoire de « *code snippets* », des petits morceaux de code qui peuvent être réutilisés dans la création d'autres applications.



[6] L'IRC (Internet Relay Chat) est un protocole de communication instantanée sur Internet qui sert principalement à la communication en groupe.

Figure 10.2. *Morceau de code source dans un courriel*

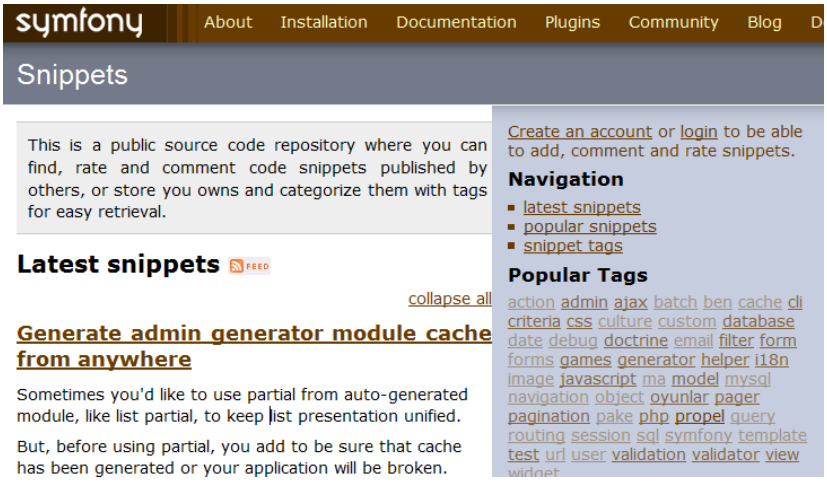


Figure 10.3. *Répertoire de « Code snippets »*

L'observation des différents espaces où est discuté et publié du code source donne l'impression d'une grande instabilité, voire d'une certaine fluidité ou d'une circulation du code source dans les différents espaces du projet étudié. Dès lors, la question qui nous préoccupe ici est de comprendre comment le code source, de symfony dans ce cas-ci – « celui que je peux télécharger » –, peut conserver la stabilité nécessaire à sa force performative.

Travail d'écriture et performativité des artefacts écrits

Pour analyser les liens entre performativité et travail de stabilisation du code source, j'aimerais référer à certains travaux en anthropologie de l'écriture qui s'attardent au travail collectif et aux mises en forme matérielles qui confèrent aux écrits leur force performative (Fraenkel, 2006 ; Denis et Pontille, 2010a). Denis et Pontille proposent par exemple de s'intéresser au travail de

maintenance dans l'analyse de la performativité de l'écrit (Denis et Pontille, 2010b). Analysant le cas de la signalétique dans le métro de Paris, les auteurs notent, en s'appuyant sur la théorie de l'acteur-réseau, que la performativité de ces panneaux repose en bonne partie sur la stabilité des relations qu'ils entretiennent au sein d'un réseau. En d'autres termes, pour qu'un panneau de signalisation soit effectivement performatif, il doit être reconnu en tant que tel, c'est-à-dire qu'il doit répondre à certaines normes graphiques, et être situé à des endroits significatifs dans le métro. Tous ces aspects demandent toutefois un certain travail de maintenance, qui se déroule au quotidien, en coulisse, et auquel nous devons nous attacher.

Béatrice Fraenkel, dans un article où elle critique le peu d'attention accordée par Austin au caractère écrit des actes juridiques, soutient quant à elle la nécessité de s'attarder aux mises en forme propres à l'écrit. Citant Latour (2004), Fraenkel note que les travaux empiriques sur la fabrique du droit montrent « qu'il existe une relation directe entre la force exemplaire de l'acte juridique et le réglage précis de ses formes textuelles, graphiques, matérielles » (Fraenkel, 2006). Elle remarque par exemple que lorsqu'Austin analyse le cas du testament, il considère sans ambiguïté que l'acte performatif se situe au moment de la lecture du testament, c'est-à-dire au moment où le testament s'exécute. Au contraire, Fraenkel soutient que l'acte juridique, pour avoir une quelconque force, doit être inséré dans « un système de chaînes d'écriture, de personnes habilitées et de signes de validation, éléments qui forment l'authenticité nécessaire à la performativité » (Fraenkel, 2006). Il ne s'agit donc pas seulement qu'un acte juridique soit écrit et publié pour avoir un pouvoir effectif, il faut encore qu'il soit authentifié par un ensemble de signes comme des sceaux, des tampons, des signatures. Ainsi, pour reprendre l'exemple du testament, le travail du ou de la notaire ne consiste pas seulement à recueillir les dernières volontés puis à exécuter ces dernières volontés par la lecture du testament (ce qui constitue le moment performatif pour Austin); le ou la notaire doit également et surtout veiller à donner une cohérence juridique à ces dernières paroles, en mettant en forme l'acte, en apposant les sceaux et les signatures appropriés. C'est en bonne partie

cette mise en forme, cette codification de l'acte juridique qui donne une force performative au testament, et non pas seulement sa lecture, ou son exécution. Fraenkel pose l'hypothèse que « tous les phénomènes de mise en forme jusqu'aux choix typographiques sont susceptibles de porter des significations et de participer à l'effectuation d'un énoncé performatif » (Fraenkel, 2006).

Signes de validation et autorisations d'écriture

Suivant Fraenkel, je voudrais à mon tour explorer la performativité du code en m'intéressant au travail d'écriture du code informatique et à son insertion dans un système de chaînes d'écriture, de signes de validation et de personnes habilitées. Il ne s'agit donc pas tant dans ce texte d'étudier les effets du code informatique, mais plutôt d'appréhender les modalités d'accomplissement de cette performativité du code, ce qu'Austin appellerait les « conditions de félicité »[7]. La réflexion de Fraenkel est par ailleurs intéressante en regard de mon analyse car elle prend pour objet les actes juridiques, ce qui permet de rejoindre la métaphore de Lessig du code informatique comme forme de loi, introduite au début du texte.

L'un des signes de validation, à première vue assez évident, concerne l'endroit où « je peux télécharger » le code source de symfony. La capture d'écran suivante montre la page web en question :

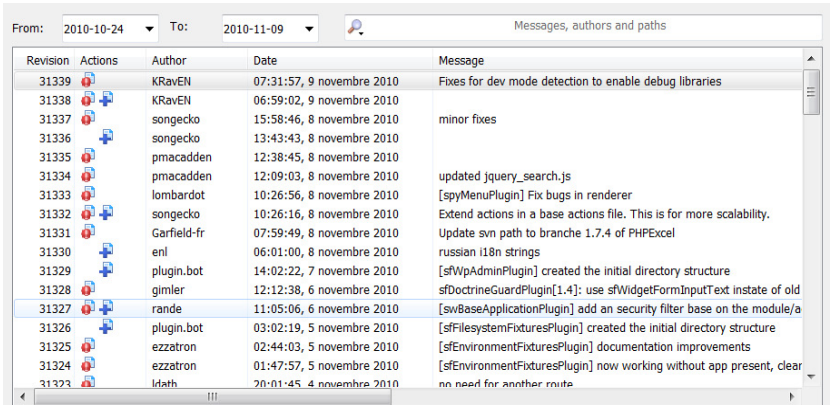
[7] Notons que certains auteurs (Denis 2006; Denis et Pontille, à paraître) prennent leur distance de l'idée même de « conditions de félicité » en soulignant la difficulté de Austin et d'autres théoriciens, à prendre en compte le caractère situé et collectif de l'accomplissement de la performativité. Ces auteurs parlent plutôt de « modalités d'accomplissement des performatifs » (Denis, 2006) ou plus simplement de « conditions de la performativité » (Denis et Pontille, 2010b), en insistant sur le caractère dynamique de ces conditions ou de ces modalités.

Figure 10.4. *Page de téléchargement de symfony*

La page de téléchargement agit bien sûr ici en tant que ressource pour télécharger symfony, mais elle agit également en tant qu'important signe de validation indiquant qu'il s'agit effectivement du code de symfony. L'autre élément intéressant de cette capture d'écran est la mention que le code source de symfony peut également être obtenu à partir du dépôt Subversion (*Subversion repository*), un système de contrôle des versions du code source. Dans symfony, le code source doit être manipulé à l'aide d'un système de gestion de versions, qui permet en outre de contrôler les droits en écriture et de récupérer des versions intermédiaires. Ainsi, plusieurs modifications successives au code source peuvent être apportées chaque jour et une contribution au code source de symfony passe par plusieurs niveaux d'autorisation et de validation.

Figure 10.5. *Historique des modifications au code source*

Au sein de symfony, seul-es les membres de l'« équipe de cœur » (*core team*), composée d'une dizaine de personnes, disposent des autorisations informatiques suffisantes pour écrire dans ce code source, « celui que je peux télécharger », pour reprendre les mots cités plus tôt. Ce code source, géré par le système de contrôle de versions, pourrait être appelé le « code source autorisé », en ce sens qu'il s'agit du code source autorisé par l'équipe de cœur à figurer sur la page de téléchargement. Plus précisément, il s'agit uniquement de la version spécifique du code source sous contrôle du système de gestion des versions, version qui a été de plus décrétée à un jour et une heure donnés par l'équipe de cœur, comme étant une version « stable ». L'écriture « autorisée » du code source étant restreinte, toute



proposition de modification au code source doit donc passer par l'intermédiaire d'un acteur humain disposant des droits suffisants en écriture. Le correctif, appelé dans les projets étudiés un « patche », est un cas exemplaire des autorisations d'écriture dans le code source informatique. La figure suivante montre une proposition d'un tel correctif au code source de symfony :

```
//SfPropelBehaviorSymfonyBehaviors.php
Replace:
31: if (\$ret = call_user_func(\$callable, \$this, \$con))
With:
31: if (!call_user_func(\$callable, \$this, \$con))
```

***Figure 10.6**. Correctif proposé au code source de symfony*

Ce correctif est proposé par un individu n'ayant pas les autorisations requises pour modifier le code source autorisé, en utilisant le système de suivi des bogues du projet, autre élément de l'infrastructure dans la chaîne d'écriture du code source. Le correctif est l'objet de 17 interventions sur une période de trois mois, interventions qui permettent d'améliorer le correctif et de le tester. Au terme de cette discussion, c'est un autre acteur, qui n'a pas participé à la discussion, mais qui possède les autorisations d'écriture requises, qui modifie finalement le code source autorisé pour intégrer la contribution. Cet acte d'écriture s'effectue par la commande informatique « commit » dans le logiciel symfony, un acte que seules certaines personnes sont autorisées à effectuer[8]. Cette courte description permet de comprendre la manière dont ce correctif, pour performer dans symfony, doit d'abord être validé par des personnes autorisées.

Règles d'écriture et conventions de nommage

Afin de conserver une certaine cohérence dans le code source, un ensemble textuel assez large, plusieurs règles d'écriture sont spécifiées. Au sein de symfony, ces règles d'écriture sont définies explicitement sous la forme de « standards de programmation ». Ces règles concernent entre autres des éléments de style, comme la tabulation ou le nombre d'espaces entre différents éléments. Une partie des règles sont cependant décrites par les acteurs et actrices comme des « conventions de nommage », qui concernent par exemple la manière de nommer des identifiants, par exemple le nom des fichiers, ou encore le nom des variables ou des fonctions dans le code. L'extrait suivant issu de la page « How to contribute to symfony » expose une

[8] C'est donc cette dernière personne, celle qui effectue le « commit», qui est reconnue comme l'auteure de la contribution. Cet enjeu de reconnaissance peut paraître banal ou interne à première vue. Il faut toutefois noter qu'un nombre grandissant de sites web comme ohloh.net s'appuient désormais sur ces données pour mettre en valeur et comparer les acteurs et actrices du monde « open source ».

des ces règles concernant les « noms » de différentes composantes du code :

```
Use camelCase, not underscores, for variable, function
and method names:
   -  Good: function makeCoffee()
   -  Bad: function MakeCoffee()
   -  Bad: function make_coffee()
```

Figure 10.7. *Règle énonçant une convention de nommage dans symfony*

Si cette règle relève encore du domaine de la stylistique, les règles de « nommage » concernent cependant beaucoup d'autres aspects. Ainsi, l'une des règles importantes est de préfixer le nom des « classes », une composante du code, par les lettres « sf », qui signifient symfony, mais également Sensio Framework, Sensio étant le nom de l'entreprise qui mène le projet (figure 10.8). Une autre règle, plus générale, est d'utiliser uniquement des termes anglophones pour nommer les composantes du code.

```
$this->validatorSchema = new sfValidatorSchema();
$this->widgetSchema    = new sfWidgetFormSchema();
$this->errorSchema     = new sfValidatorErrorSchema($this
```

Figure 10.8*. Utilisation du préfixe « sf » pour nommer certains identifiants*

Si quelques-unes de ces règles sont explicites, en revanche, d'autres règles sont plus implicites. On retrouve sur une page concernant les standards dans symfony un énoncé selon lequel la « règle d'or » consiste à imiter le code existant (« Here's the golden rule: *Imitate the existing symfony code* »[9]). Une personne qui veut contribuer au code source doit donc s'assurer de conserver une certaine cohérence, ou homogénéité, dans le code source. Lors d'une entrevue, une des actrices du projet nous parlait de ces règles d'écriture, en particulier du

[9] En italiques dans le texte original.

fait d'écrire en anglais, comme une sorte de « contrat moral » *(sf05, 9 mars 2010),* dans le sens que cette règle n'est renforcée ni par la licence d'utilisation ni par une contrainte « technique ». En revanche, ces règles sont renforcées par le fait de voir son code intégré, ou non, à la chaîne d'écriture.

Conclusion

J'ai voulu montrer que le travail d'écriture du code source est étroitement articulé à un ensemble de règles et d'autorisations qui confèrent au code la stabilité sur laquelle sa performativité repose. En articulant l'analyse de Fraenkel aux propos de Lessig, nous pourrions dire que pour qu'un morceau de code source ait effectivement une force « juridique » – ou performative –, il doit pour cela être inséré dans un système de chaînes d'écriture qui sont elles-mêmes articulées à un ensemble de règles et d'autorisations. La comparaison du code source avec les écrits juridiques de Fraenkel a cependant des limites, dont la plus importante est sans doute le fait que le code source a la spécificité d'être destiné à être exécuté par une machine. Une seconde limite de la comparaison concerne le rôle des règles et des autorisations d'écriture. Dans le cas des actes juridiques, ces règles et autorisations sont des éléments qui font en quelque sorte partie de la matérialité intrinsèque à l'acte juridique et dont le rôle est d'assurer l'authenticité de l'acte écrit. Pour le code source, les règles et autorisations sont plutôt extrinsèques à l'artefact, et prennent part à un processus de validation de ce qui constitue le code source autorisé, « celui que je peux télécharger », qui sera éventuellement exécuté comme composante d'un logiciel (symfony dans ce cas-ci). Dans un certain sens, nous pourrions parler, pour le code informatique, d'une force performative plus ou moins grande. Dans la mesure où le code informatique, s'il est exécuté, va toujours effectivement « faire », la question n'est pas tant de savoir s'il s'exécutera ou non, mais plutôt de savoir quel effet aura cette exécution du code. Ainsi, un morceau de code source gagne en force performative s’il est intégré dans un système de chaînes d'écriture et est validé par des personnes habilitées. Cette approche rejoindrait celle de Mackenzie, pour qui le code

informatique tire sa force performative de la répétition, de la citation d'un *a priori*. Au contraire de Mackenzie toutefois, l'insistance sur la chaîne d'écriture, inspirée en cela des travaux en anthropologie de l'écriture, permet d'aborder le code source en tant qu'artefact écrit.

Dans le cadre de ce texte, nous avons exploré la manière par laquelle le code informatique conserve la stabilité et la cohérence nécessaires à sa performativité. La performativité dont nous avons traitée ici est cependant celle d'un dispositif technique – le code exécutable – et non pas tant celle du code source, comme artefact écrit. C'est en effet en tant qu'elle participe à la stabilisation du code (exécutable) – et donc du dispositif technique – que nous avons abordé la chaîne d'écriture du code source. Cependant, comme l'indique Inke Arns à propos de l'art numérique, même le code source non-exécutable fait sens, puisqu'il peut, en lui-même, être l'objet d'une production artistique. De la même manière, il est également important d'appréhender le code source en soi comme le produit d'une pratique d'écriture collective, similaire à celles que l'on retrouve aujourd'hui sur Internet, par exemple dans l'usage des wikis. Dans cette perspective, le code source informatique ne doit pas être appréhendé uniquement comme un artefact dans la conception d'un dispositif technique, mais également (et peut-être avant tout) comme un espace d'écriture et d'interaction entre humains, et entre humains et machines.

Bibliographie

ARNS, I., 2005, « Code as performative speech act », *Artnodes,* Juillet. En ligne <http://www.uoc.edu/artnodes/espai/eng/art/arns0505.pdf> Consulté le 14 juillet 2010.

AUSTIN, J., 1975, *How to do things with words*, 2e édition, Cambridge : Harvard University Press.

BUTLER, J., 1997, *Excitable Speech: A Politics of the Performative*. 1ère édition, New York, Routledge.

BUTLER, J., 2004, *Le pouvoir des mots : Politique du performatif,* Paris, Éditions Amsterdam.

DENIS, J. 2006, « Les nouveaux visages de la performativité », *Études de communication*, n° 29, p. 7-24.

DENIS, J., PONTILLE D., 2010a, *Petite sociologie de la signalétique : Les coulisses des panneaux du métro*, Paris, Presses de l'Ecole des mines.

DENIS, J, PONTILLE D., 2010b. « Performativité de l'écrit et travail de maintenance », *Réseaux*, vol. 163, n° 5, p. 105-130.

FRAENKEL, B., 2006, « Actes écrits, actes oraux : la performativité à l'épreuve de l'écriture », *Études de communication,* n° 29, p. 67-96.

LATOUR, B., 1992, *Aramis ou L'amour des techniques,* Paris, La Découverte.

LATOUR, B., 2004, *La fabrique du droit : Une ethnographie du Conseil d'État,* Paris, La Découverte.

LESSIG, L., 2006, *Code 2.0,* New York, Basic Books.

LICOPPE, C., 2010, « Présentation », *Réseaux*, vol. 163, n° 5, p. 9-10.

MACKENZIE, A., 2005, « The Performativity of Code: Software and Cultures of Circulation », *Theory Culture Society,* n° 22, pp. 71-92.

STAR, S. L., 1999, « The Ethnography of Infrastructure », *American Behavioral Scientist*, vol. 43, n° 3, p. 377-391.

WINNER, L., 2002, « Les artefacts font-ils de la politique? », *La baleine et le réacteur. À la recherche de limites au temps de la haute technologie*, p. 35-74, Paris, Descartes et cie.

Biographie

Stéphane Couture est doctorant en communication à l'Université du Québec à Montréal. Il est membre du Laboratoire de communication médiatisée par ordinateur (LabCMO) et membre étudiant du Centre interuniversitaire de recherche sur la science et la technologie (CIRST).