The role of aesthetics in understanding source code

Pierre Depaz

Le rôle de l'esthétique dans la compréhension du code source

Pierre Depaz
sous la direction d'Alexandre Gefen (Paris-3)
et Nick Montfort (MIT)

ED120 - THALIM

Septembre 2023

Introduction

En tant que texte dont le but même est de disparaître (transformé en changements de courant électrique), le code source est un objet hybride, autant description qu'action, aux interlocuteurs multiples. Il peut être communication entre humain et humain, entre humain et machine, ou entre machine et machine. Cependant, en tant que création humaine, il est possible de s'interroger sur les modalités de ses manifestations, notamment en termes d'expressivité. Si l'expressivité fonctionnelle d'un code source implémentant un algorithme est indéniable, l'expressivité artistique d'un texte de code source demeure élusive. Quelle est donc la place de l'esthétique, d'une manifestation formelle sensuellement plaisante, dans l'écriture ou la lecture du code source? Le code source, basé sur un système syntaxique similaire au language naturel, se révèle être un système de communication aussi bien d'humain à humain que d'humain à machine. Il

semble néanmoins être principalement destiné les humains, et secondairement aux les machines (Abelson et al., 1979). Une fois que la condition principale d'existence du code source (sa validité d'exécution) est satisfaite, une condition secondaire semble être non pas sa beauté mais sa compréhensibilité.

Se pose donc la question d'un texte dont les manifestations formelles sont vouées à disparaître, et dont la lecture n'est qu'un processus collatéral de son exécution. Donald Knuth, dans son oeuvre *The Art of Computer Programming* (Knuth, 1997), commence par établir que la programmation (l'écriture et la lecture du code source) est bel et bien un art. Cette déclaration, dans les premières pages du premier volume, n'est pourtant pas réitérée ni élaborée dans les autres volumes du monographe. Si écrire du code peut être un art, certains codes sources peuvent donc exhiber des propriétés esthétiques, mais ces dernières sont rarement explicitées par la littérature sur le sujet.

La problématique principale de ce projet de recherche est donc celle du *rôle de l'esthétique dans les compréhensions du code sources*. Il s'agit tout d'abord de mettre à jour les propriétés esthétiques propres au code source, et d'identifier comment celles-ci permettent de faire sens. En plus d'une approche empirique des codes sources en eux-mêmes, il s'agit de mettre à jour de quelles manières ces propriétés sont similaires à d'autres domaines de créations mobilisés par les programmeurs, notamment au domaine de la littérature, de l'architecture, des mathématiques et de l'ingénierie. En examinant ces relations, cette thèse s'inscrit donc dans un travail de recherche sur la dimension cognitive du phénomène esthétique.

Une approche rapide de l'état de l'art sur ce sujet révèle deux tendances séparées: d'une part, la littérature en informatique et ingénierie prend en compte l'évidence de l'existence d'une esthétique du code source, du point de vue la productivité de ceux et celles qui écrivent du code, et d'un point de vue cognitif de la compréhension des bases de code (Oram & Wilson, 2007; Cox & Fisher, 2009; Gabriel & Goldman, 2001; Martin, 2008; Deti-

enne, 2001; Weinberg, 1998). Néanmoins, ces études considèrent une esthétique comme un phénomène donné, aux conséquences plus ou moins mesurables, sans pour autant systématiser leur approche au niveau conceptuel.

D'autre part, les recherches en sciences humaines traitent de l'interaction entre esthétique et code, tout en restant majoritairement attachées à une conception abstraite et désincarnée du "code", élaborant une esthétique du digital sans pour autant rentrer dans les détails des codes sources eux-mêmes (e.g. il ne semble pas pertinent qu'ils soient écrits en Perl, ou Python, ou Ruby, etc.) (Cramer, 2019; Hayles, 2010; Mackenzie, 2006; Lévy, 1992). Il existe cependant un certain nombres d'ouvrages de sciences humaines approchant la question matérielle du code de manière directe, que ce soit au niveau culturel (Montfort et al., 2014), politique (Cox & McLean, 2013) ou sociologique (Paloque-Bergès, 2009) Deux travaux doctoraux se sont précédemment intéressés à l'esthétique du code source (Black, 2002; Pineiro, 2003), mais se limitent à des communautés d'écriture du code bien précises, sans examiner les codes sources eux-mêmes. C'est au sein de cette approche sur les manifestations du code source que cette recherche s'inscrit.

Méthodologie

L'approche principale de ce projet est premièrement empirique. Il s'agit d'examiner les codes sources eux-mêmes, ainsi que les discours de ceux et celles qui les écrivent et les lisent. Pour cela, je m'appuie sur les travaux de Kintsch et van Dijk et leurs études des stratégies de compréhension du discours (Kintsch & van Dijk, 1978), l'approche métaphorique considérée comme une stratégie de compréhension. Le code source étant un texte, autour duquel sont produits des discours.

Il a d'abord s'agit de faire émerger une variété d'individus écrivant et

lisant du code, que j'ai regroupé en quatre catégories (Hayes, 2015). Les *développeurs* sont les individus qui écrivent du code dans un contexte économique, à but fonctionel et soutenable à long-terme; ils travaillent sur des bases de code extensives et souvent en collaboration. Les *artistes* sont les individus écrivants du code dans un cadre expressif et artistique, tels que des *code poems*; leur code est destiné principalement à être lu par un public, et seulement de manière secondaire à être exécuté. Les *hackers* sont les individus qui écrivent du code en tant que solution technique unique, idiosyncratique et hautement contextuelle; leur code est destiné à être exécuté par une machine particulière, et de manière secondaire par des humains. Enfin, les *académiques* sont les individus qui écrivent du code afin d'illustrer des concepts de computation, représentant des abstractions plutôt que des usages concrets.

Ces quatres groupes ont des limites poreuses, et un code source peut, souvent, appartenir à plusieurs de ces catégories en même temps (une référence artistique dans un contexte commercial, un hack présenté comme oeuvre d'art, ou une approche particulière au sein d'une base de code commerciale, etc.).

La constitution du corpus de cette recherche s'est faite principalement par la consultation de ressources en ligne. En effet, la plupart des bases de code existe sur des sites d'aggrégations (GitHub, BitBucket, etc.) ou sous formes d'extraits (*snippets*) présentés et commentés dans des forums, sur des sites personnels, ou des sites de question/réponse (e.g. StackOverflow, Quora). Cette constitution du corpus inclue non seulement ces sources primaires (le code en soi), mais également des sources secondaires (le commentaire par un auteur, ou par un public, justifiant de l'aspect esthétique d'un code présenté). De cette manière, je considère la valeur esthétique d'un code source comme étroitement lié au jugement de groupe émis à son encontre.

Ma métholodologie consiste en une approche interprétative du corpus constitué, dans un double-mouvement. D'une part, les références et dis-

cours des individus écrivants du code sont pris en compte afin de constituter un ensemble de standards esthétiques: ces standards sont des manifestations concrètes du code permettant une facilitation de la compréhension du *propos* du code (propos prescriptif, ou effectif). D'autre part, les standards exhibés sont appliqués à d'autres bases de code afin de vérfier leur applicabilité. Ã travers la multiplicité de ces standards, il s'agit donc d'identifier les stratégies métaphoriques de compréhensions des lecteurs de code.

Cette examination se fait également à travers un cadre théorique partant de la philosophie esthétique et de la littérature afin de définir mon utilisation du terme *esthétique*. D'un point de vue littéraire, je m'appuie sur les travaux de Gérard Genette et sa distinction entre fiction et diction (Genette, 1993), considérant l'esthétique du code comme sa diction, tandis que la poétique serait sa fiction. Entre littérature, philosophie et sciences cognitives se trouvent les oeuvres de Paul Ricoeur (Ricoeur, 2003) et de George Lakoff (Lakoff & Johnson, 1980), reliant composition verbale et évocation d'images mentales. Enfin, cette manifestation en surface est reprise par le philosophe de l'art Nelson Goodman dans son analyse des languages de l'art (Goodman, 1976), et notamment dans son exploration entre systèmes syntactiques et communication, ainsi que son approche scientifique des phénomènes artistiques.

Développement

Premièrement, il n'y a pas "un" code source, abstrait et désincarné, mais bien une multiplicité de codes sources, existant au sein de pratiques et de groups différents. Ces codes sources possèdent tous une possibilité de manifestation esthétique, manifestations soumise à la manifestation et au transfert de *connaissances*. En ce que le code source possède tant un son aspect prescriptif (ce que le code *doit* faire) qu'un aspect effectif (ce que

le code *fait*), les esthétiques du code source peuvent se décliner le long de cette axe, avec les poètes et les académiques se concentrant sur l'aspect prescriptif, évoquant des concepts à travers la machine, et les hackeurs se concentrant sur l'aspect effectif, évoquant les concepts de la machine.

Pour communiquer l'intention du programme à travers son implémentation textuelle et en rapport avec son domaine d'activité, les programmeurs disposent de différents possibilités. Il est possible de développer une certaine du problème vers à travers divers niveaux de métaphores linguistiques (rhétorique procédurelle (Bogost, 2008), double-codage (Cox & McLean, 2013), double-signification (Paloque-Bergès, 2009)) par un choix précis de vocabulaire. Le code est donc ici une double sorte de langue: langue machinique et langue humaine dont les tensions syntactiques peuvent créer une richesse sémantique.

Ensuite, ces mécanismes linguistiques sont complémentés par des choix de structures et de syntaxe faisant appel à des domaines d'architecture et d'ingénierie (Gabriel, 1998; Schummer et al., 2009), qui permettent alors de mieux appréhender une exécution du code source—exécution dont la vitesse à laquelle elle se produit la rend incompréhensible pour des humains. L'esthétique du code source permet alors de représenter les espaces d'évolution des flux d'informations lors de l'exécution d'un programme.

Finalement, ces standards esthétiques se regroupent autour de la notion d'élégance—c'est-à-dire de l'utilisation d'une syntaxe minimale pour une expressivité maximale. Particulièrement présente dans les groupes de hackeurs et des académiques, celle-ci se décline aussi le long d'un axe machine-concept. Les hackeurs ont tendance à n'utiliser que ce qui est nécéssaire pour effectuer une action particulière, bien visible dans les concours de *demoscenes* (Kudra, 2020), tandis que les académiques vont témoigner de la même approche, mais pour communiquer des concepts fondamentaux de la science informatique, découplée de la machine spécifique qui exécute le programme en question.

Idéaux discursifs

L'histoire de la programmation est celle d'une pratique née au lendemain de la Seconde Guerre mondiale, qui s'est étendue à des publics de plus en plus larges à l'aube du XXIe siècle. Tout au long de ce développement, divers paradigmes, plateformes et applications ont été impliqués dans la production de logiciels, donnant lieu à différentes communautés épistémiques (Cohendet et al., 2001) et communautés de pratique (Hayes, 2015), produisant à leur tour différents types de code source. Chacune d'entre elles écrit un code source avec des caractéristiques particulières, et avec des priorités différentes dans la manière dont les connaissances sont produites, stockées, échangées, transmises et récupérées. Nous délimitons ces communautés en quatre catégories : les développeurs de logiciels, les hackers, les scientifiques et les poètes. Même si leurs types de lecture et d'écriture du code source se recoupent souvent, nous mettons en évidence la diversité des modes d'écriture du code, notamment en termes d'origine - comment une telle pratique a-t-elle émergé ? -, de références - que considèrent-ils comme bons ? -, de finalités - pour quoi écrivent-ils ? - et d'exemples - à quoi ressemble leur code ?

Les développeurs de logiciels travaillent dans un domaine commercial dont l'objectif est de produire des logiciels complexes, fonctionnels et durables. Ils sont étroitement liés à l'industrie des technologies de l'information, ainsi qu'au développement du matériel et des langages de programmation. Également issus du milieu universitaire, les hackers écrivent du code qui tend à être impénétrable, intelligent et très efficace à court terme, en participant à des compétitions ludiques telles que le golf du code ou les concours d'obscurcissement. En contrepartie, les scientifiques se différencient selon qu'ils traitent le code comme moyen (valorisant reproducibilité, durabilité) ou comme fin (valorisant expressivité, capacité à représenter concisement des idées computationnelles complexes). Enfin, les artistes du logiciel et les poètes du code se concentrent sur le code

source en tant que matériau sémantique textuel et ambigu, cherchant à comprimer le sens en exploitant les liens entre les langages naturels et les langages machine.

Bien qu'aucune de ces communautés de pratique ne s'exclue mutuellement - un développeur de logiciels le jour peut *hacker* le week-end et participer à des événements de *code poetry* -, elles nous aident à comprendre comment les manifestations du code source dans les textes programmés et leur évaluation par les programmeurs peuvent avoir de multiples facettes. Cette apparente diversité n'en présente pas moins des similitudes. Les extraits de code présentés dans cette section montrent qu'il existe une tendance à préférer un groupe spécifique de qualités - lisibilité, concision, clarté, expressivité et fonctionnalité - même si différents types de pratiques mettent un accent différent sur chacun de ces aspects.

Après avoir passé en revue la diversité des pratiques et des textes programméss parmi ceux qui lisent et écrivent du code source, nous allons maintenant analyser les normes esthétiques les plus appréciées par ces différents groupes. Pour ce faire, nous utilisons un cadre d'analyse du discours pour l'étude empirique du corpus, suivi d'un examen des discours que les programmeurs déploient lorsqu'il s'agit d'expliciter leurs préférences esthétiques en matière de code source (Mullet, 2018). Nous verrons que, si les domaines esthétiques mobilisés pour justifier les normes esthétiques sont clairement distincts, nous pouvons néanmoins identifier des ensembles récurrents de valeurs esthétiques à l'aune desquelles la qualité du code source peut être mesurée.

La *propreté* est liée à l'expressivité: absence de tout symbole syntaxique et sémantique étranger, elle facilite l'identification du cœur du problème. Dans un texte de programme propre, les détails superflus disparaissent au niveau syntaxique pour permettre l'expressivité au niveau sémantique. Le corollaire de la clarté est l'obscurcissement, ou *obfuscation*. Il s'agit de l'acte, intentionnel ou non, de compliquer la compréhension de ce que fait un programme en égarant le lecteur par une combinaison de tech-

niques syntaxiques, un processus visible dans les compétitions de l'IOCCC (Mateas & Montfort, 2005). Dans son sens le plus large, l'obscurcissement est utilisé à des fins de production pratique: réduire la taille du code et empêcher la fuite d'informations propriétaires concernant le comportement d'un système.

La *simplicité* du code source est, elle, une forme de parcimonie et d'équilibre[1]. Cette exigence d'équilibre nous amène à distinguer deux types de simplicité : la simplicité syntaxique et la simplicité ontologique. La simplicité syntaxique mesure le nombre et la concision des jetons lexicaux et des mots-clés visibles. La simplicité ontologique, quant à elle, mesure le nombre de types d'entités impliqués dans la sémantique du texte programmé. Le code source peut être syntaxiquement simple parce qu'il rassemble des concepts complexes en un nombre limité de caractères, ou il peut être ontologiquement simple, en raison du nombre minimal de concepts informatiques impliqués. La simplicité syntaxique a également une conséquence plus immédiate sur l'expérience de lecture d'un texte de programme : l'un des problèmes auxquels les programmeurs sont confrontés est qu'il y a tout simplement trop de lignes de code pour que l'on puisse s'y retrouver, ce qui oblige à réduire le contenu à son minimum fonctionnel (Butler, 2012).

Inversement, la nature intellectuelle de la pratique d'un programmeur implique souvent des solutions idiosyncratiques, voire alambiquées, et *ad hoc*, qui permettent de résoudre rapidement un problème donné, mais qui peuvent également être difficiles à généraliser ou à comprendre sans une aide extérieure: *clever hacks*. Cette aide extérieure prend souvent la forme d'une explication, et n'est pas souvent appréciée de manière positive[2]. Des *clever hacks* sont souvent trouvés dans les exemples de code écrits par des

[1]Gibbons cite Ralph Waldo Emerson pour à ce sujet: *"Nous attribuons la beauté à ce qui est simple ; qui n'a pas de parties superflues ; qui répond exactement à sa fin ; qui est lié à toutes les choses ; qui est la moyenne de beaucoup d'extrêmes. (Gibbons, 2012)*

[2]Comme l'a souligné en ligne Mason Wheeler:*"Lorsque cela nécessite beaucoup d'explications, il ne s'agit pas d'un "beau code", mais d'un "hack intelligent"*. (Overflow, 2013)

hackers, et parfois tournée en dérision, car ils perturbent cet équilibre entre précision et généralité, ceux-ci tendant à exploiter les particularités de la connaissance du support (le code) plutôt que de l'objectif (le problème).

Liée à la simplicité par le biais de la nécessité et de la suffisance, la perception de l'élégance est également liée à un sentiment subjectif d'adéquation, de correspondance, d'être apte. Reprenant certaines des définitions de la simplicité que nous avons vues jusqu'à présent, Paul DiLascia, qui écrit dans le Microsoft Developer Network Magazine, illustre sa conception de l'élégance - en tant que combinaison de simplicité, d'efficacité et de brillance - avec la récursion. L'élégance est donc bien le "Saint-Graal" des normes esthétiques de programmation (Temkin, 2023).

Une approche complémentaire pour comprendre ce que les programmeurs veulent dire lorsqu'ils parlent d'un beau code consiste à aller au-delà des termes positifs utilisés pour le qualifier et à porter notre attention sur les qualificatifs négatifs. Par exemple, *code spaghetti* fait référence à une propriété du code source où la syntaxe est écrite de telle manière que l'ordre de lecture et de compréhension s'apparente à démêler une assiette de pâtes spaghetti. Bien que techniquement toujours linéaire dans l'exécution, cette linéarité perd ses avantages cognitifs en raison de son extrême convolution, qui ne permet pas de savoir clairement ce qui commence et ce qui finit, à la fois dans la déclaration et dans l'exécution du code source (Steele, 1977). Cette métaphore matérielle est utilisée de la même manière par Foote et Yoder, qui décrivent le code comme une "grosse boule de boue"[3].

S'il y a une cohérence dans la description des moyens d'écrire du beau code, par l'examen d'un champ lexical aux identifiants clairs, cette analyse

[3] *Une grosse boule de boue est une jungle de code spaghetti, structurée de manière désordonnée, tentaculaire, bâclée, avec du ruban adhésif et des fils de fer qui se chevauchent. Ces systèmes présentent des signes évidents de croissance non régulée et de réparations répétées et rapides. Les informations sont partagées sans discernement entre les éléments distants du système, souvent au point que presque toutes les informations importantes deviennent globales ou dupliquées.* (Foote & Yoder, 1997)

ouvre aussi d'autres voies de recherche. Tout d'abord, nous constatons qu'il existe une relation entre les manifestations formelles et le fardeau cognitif, et que l'esthétique contribue à alléger ce fardeau. Un beau code rend accessibles les idées qu'il contient, ainsi que le monde qu'il vise à traduire et sur lequel il opère. En outre, les adjectifs négatifs mentionnés en référence aux aspects formels du code (malodorant, boueux, enchevêtré) sont éminemment *matérialistes*, ce qui indique une tension intéressante entre les idées du code et la sensualité de sa manifestation.

Passant d'un niveau syntaxique à un niveau thématique, pour faire référence au cadre d'analyse du discours de Kintsch et Van Dijk (Dijk & Kintsch, 1983), nous allons maintenant nous pencher sur chacun de ces domaines et sur ce qu'ils nous apprennent sur le code source.

L'hypothèse ici est qu'un medium - tel que le code source - est un moyen d'expression, et que différents supports peuvent soutenir différentes qualités d'expression ; en outre, une analyse comparative peut être productive car elle révèle les chevauchements entre ces supports. Puisqu'il semble y avoir des façons spécifiques de considérer le code comme beau, ces domaines adjacents, et les parties spécifiques de ces domaines qui créent cette contingence, préparent notre travail de définition de normes esthétiques spécifiques au code source.

Pour ce faire, nous examinerons les trois domaines les plus souvent évoqués par les programmeurs lorsqu'ils commentent les qualités sensuelles ou les expériences esthétiques suscitées par le code source: la littérature, les mathématiques et l'architecture. Bien qu'il existe des parallèles entre la programmation et la peinture (Graham, 2003) ou la programmation et la musique (McLean, 2004), ceux-ci se réfèrent au peintre ou au musicien en tant qu'individu, plutôt qu'au médium spécifique, et il n'y a, à notre connaissance, aucun compte-rendu du code comme de la sculpture, du film ou de la danse, par exemple.

En se référant au code en tant que texte, sa nature linguistique est mise en évidence, ainsi que sa relation problématique avec les langues na-

turelles - problématique dans la mesure où son ambiguïté peut jouer contre son désir d'être clair et compris, ou peut jouer en faveur de sous-entendus poétiques. Les normes exprimées ici concernent la syntaxe spécifique utilisée pour écrire la programmation, sa relation avec le langage naturel et son potentiel d'expressivité.

Compte tenu de la nature formelle du code source, les métaphores scientifiques assimilent le code source à la possibilité de présenter des propriétés similaires à celles des preuves et des théorèmes mathématiques, où l'élégance de la preuve n'est pas un couplage étroit avec le théorème à prouver, mais où une preuve élégante peut (et, selon certains, devrait) éclairer le lecteur sur des vérités plus profondes. Inversement, ces références scientifiques incluent également l'ingénierie, dans laquelle l'applicabilité, la justesse et l'efficacité sont primordiales : la conception de l'élégance change alors de perspective, tendant à la sensation générale de la structure en question, plutôt qu'à ses formalismes spécifiques.

Ces références à l'ingénierie nous conduisent au dernier des domaines : l'architecture. Celle-ci est présentée comme pertinente à la fois d'un point de vue *top-down* (avec des langages de modélisation formels et des descriptions, entre autres) et d'un point de vue *bottom-up* (y compris les modèles de logiciels, la familiarité et l'adéquation dans un contexte donné). Ces similitudes entre les logiciels et l'architecture, à la fois dans la planification, la pratique et les perspectives, touchent un autre sujet : la place des connaissances formelles et informelles dans la construction, la maintenance et la transmission de ces structures (logicielles)."

Enfin, au-delà des références à ces métaphores, nous pouvons également souligner le fait que l'esthétique semble toujours liée à la nécessité de comprendre les textes programmés. Nous avons mis l'accent sur la compréhension lorsqu'il s'agit de normes esthétiques : qu'il s'agisse d'obscurcir ou d'éclairer, le processus d'acquisition d'un modèle mental d'un objet informatique donné demeure un facteur déterminant dans le jugement de valeur appliqué au code source.

Complexité des pratiques de programmation

L'esthétique du code source est donc principalement liée à la compréhension. Dans cette section, nous nous concentrerons sur la raison pour laquelle les logiciels impliquent une telle charge cognitive, avant de passer en revue les moyens - à la fois linguistiques et technologiques - que les programmeurs impliquent pour alléger cette charge.

Cette exigence de compréhension, que ce soit de manière sérieuse, ludique ou poétique, est liée à l'une des caractéristiques essentielles des logiciels : ils doivent être *fonctionnels*. Comme nous l'avons mentionné dans notre discussion sur les différences entre le code source et le logiciel dans l'introduction, le code source est la description latente de ce que le logiciel fera en fin de compte. Par conséquent, un logiciel bogué ou dysfonctionnel aura toujours moins de valeur qu'un logiciel correct (Hill, 2016), quel que soit le degré d'esthétisme de sa source. Tout jugement de valeur concernant l'esthétique du code source serait subordonné au fait que le logiciel fonctionne correctement ou non, et ce jugement est nul si le logiciel ne fonctionne pas.

L'évaluation du bon fonctionnement d'un logiciel peut être décomposée en plusieurs sous-parties : savoir ce que le logiciel fait effectivement, ce qu'il est censé faire, être capable de dire si ces deux choses sont alignées, et comprendre comment il le fait. Après avoir décidé d'un point de référence pour évaluer la fonctionnalité du code source en question (comprendre ce qu'il devrait faire), il faut ensuite déterminer le comportement réel du code source en question une fois qu'il est exécuté (comprendre ce qu'il fait réellement). Il faut également comprendre comment le texte d'un programme le fait linguistiquement parlant, afin de le modifier.

De manière plus générale, les théories sur la manière dont les individus acquièrent la compréhension (comment ils en viennent à connaître les choses et à en développer des représentations opérationnelles et conceptuelles) ont été abordées d'un point de vue formel et d'un point de vue

contextuel. La tradition philosophique rationaliste et logique dont est issue l'informatique part du principe que le sens peut être rendu non ambigu par l'utilisation d'une notation spécifique. La compréhension explicite, en tant que lignée théorique de l'informatique, a ensuite été réalisée dans des situations concrètes par le biais des langages de programmation (Depaz, 2023).

Cependant, la spécification explicite du sens ne permet pas de traiter des tâches basiques que les humains considèrent comme évidentes. Cela nous amène à envisager une approche différente de la compréhension, dans laquelle celle-ci est acquise par des moyens contextuels et incarnés (Penny, 2019). En particulier, nous pouvons identifier la connaissance tacite comme reposant sur une composante sociale, ainsi que sur une composante somatique (Polanyi & Grene, 1969).

Le code source, en tant que système formel néanmoins dépendant du contexte, de l'intention et de la mise en œuvre, mobilise les deux approches de la compréhension. En raison de la nature pratique et ascendante de la programmation, les tentatives de formalisation se sont appuyées sur l'hypothèse selon laquelle les programmeurs experts disposent d'un certain type de connaissances tacites (Soloway et al., 1982; Soloway & Ehrlich, 1984).

Les logiciels impliquent, par le biais de langages de programmation, l'expression de modèles abstraits humains pour l'interprétation par la machine, qui à son tour est exécutée à une échelle de temps et d'espace qui est difficile à saisir pour les individus. Ces propriétés les rendent difficiles à comprendre, de la conception à l'application : dans le monde réel, les logiciels passent par un processus d'implémentation de concepts qui se perdent dans la traduction, interfacent le monde par des représentations discrètes et suivent l'exécution de ces représentations dans l'espace et le temps. Dans ce processus, le code source reste la représentation matérielle de toutes ces dynamiques et le seul point de contact entre l'action du programmeur et l'exécution de la machine et, en tant que tel, reste le lieu de la

compréhension.

D'une part, une manière de rendre les processus computationnels compréhensibles est le recours à la métaphore. Les métaphores jouent un rôle dans la compréhension en créant des liens entre les connaissances préexistantes et les connaissances actuelles, en suggérant des similitudes entre les deux afin de faciliter la construction de modèles mentaux du domaine cible. Les métaphores sont utilisées par les programmeurs à un niveau d'abstraction supérieur, les aidant à saisir des concepts (par exemple, la mémoire, les objets, les paquets, les réseaux) sans avoir à se préoccuper des détails.

Les programmeurs s'appuient également sur des outils logiciels spécifiques, afin de faciliter l'analyse et l'exploration des fichiers de code source, tout en exécutant des tâches banales qui ne devraient pas nécessiter une attention particulière de la part du programmeur, telles que l'établissement de liens ou le remaniement de dénomination (*refactoring*). L'utilisation de logiciels pour comprendre les logiciels est en effet paradoxale, mais participe néanmoins à la cognition étendue ; les moyens que nous utilisons pour raisonner sur les problèmes affectent, dans une certaine mesure, la qualité de ce raisonnement.

Les moyens mis en œuvre pour comprendre et appréhender les ordinateurs et les processus informatiques sont donc à la fois linguistiques et techniques. Linguistiques, parce que l'usage des ordinateurs est truffé de métaphores qui facilitent la compréhension de ce que sont et ce que font les entités présentées. Ces métaphores ne s'adressent pas seulement aux utilisateurs, mais sont également utilisées par les programmeurs eux-mêmes. Technique, parce que l'écriture et la lecture du code se sont historiquement appuyées de plus en plus sur des outils, tels que les langages de programmation et les EDI (environnements de développement intégrés), qui permettent aux programmeurs d'effectuer des tâches transparentes spécifiques au code source.

Dans la section suivante, nous poursuivons notre enquête sur les

moyens de compréhension, en nous éloignant des logiciels et en nous concentrant sur la façon dont les domaines esthétiques. Cela nous permettra de montrer comment l'esthétique en général, peut avoir pour fonction de rendre compréhensible l'imperceptible.

Esthétique et cognition

Jack Goody et Walter Ong ont montré dans leurs études anthropologiques que le principal moyen de communication des communautés étudiées affecte l'engagement de ces communautés vis-à-vis de concepts tels que la propriété, l'histoire et la gouvernance (Ong, 2012; Goody, 1986). Plus récemment, Edward Tufte et ses travaux sur la visualisation des données ont fait progresser cette ligne de recherche en se concentrant sur la traduction de données similaires d'un support textuel à un support graphique (Tufte, 2001). Plusieurs arguments ont ainsi été avancés en faveur de l'impact de l'apparence sur la structure, à la fois dans le code source et ailleurs. Nous entendons ici généraliser cette approche comparative entre plusieurs médiums, en examinant comment le code source se comporte expressivement comme un langage de l'art, à partir de la théorisation de Nelson Goodman de tels langages (Goodman, 1976).

Un système de symboles est basé sur des exigences qui pourraient indiquer que l'œuvre créée dans un tel système serait capable de susciter une expérience esthétique[4]. Un tel système devrait être composé de signes syntaxiquement et sémantiquement disjoints, syntaxiquement complets et sémantiquement denses. Cette classification permet de comparer la

[4] "*Peut-être devrions-nous commencer par examiner la pertinence esthétique des principales caractéristiques des différents processus symboliques impliqués dans l'expérience, et rechercher des aspects ou des symptômes, plutôt qu'un critère esthétique précis. Un symptôme n'est ni une condition nécessaire ni une condition suffisante pour l'expérience esthétique, mais tend simplement, en conjonction avec d'autres symptômes de ce type, à être présent dans l'expérience esthétique.*" (Goodman, 1976)

manière dont les différents systèmes de symbolisation utilisés dans l'art et la science expriment les concepts.

Le code source est écrit dans un système linguistique formel appelé langage de programmation. Un tel système linguistique est de nature numérique et satisfait donc au moins aux deux exigences de disjonction syntaxique (aucune marque ne peut être confondue avec une autre) et de différenciation (une marque ne correspond jamais qu'à ce symbole). Bien qu'il ne soit pas aussi syntaxiquement dense que la musique ou la peinture, il est néanmoins sans ambiguïté : le code source peut donc être considéré comme un langage artistique.

Monroe Beardsley souligne lui aussi la possibilité d'une expérience esthétique à rendre compréhensible, par le biais d'une *découverte active*, et il considère les métaphores comme un moyen d'y parvenir (Beardsley, 1970). Cette interaction entre l'intégration d'une métaphore dans nos structures mentales quotidiennes, l'émergence de la poésie dans le pensable et la création d'une tension pour qu'une telle émergence se produise, plaide en faveur d'au moins une des conséquences d'une expérience esthétique, et donc d'une de ses fonctions : donner un sens aux concepts complexes du monde.

S'appuyant sur les travaux mentionnés ci-dessus, Catherine Elgin étudie la relation entre l'art et l'épistémologie, en examinant comment les symboles indéterminés sur le plan de l'interprétation font progresser la compréhension, et ce dans le contexte de l'indétermination interprétative. Étant donné que des systèmes de symboles syntaxiquement et sémantiquement denses sont utilisés dans les œuvres d'art, c'est cette multiplicité d'interprétations qui exige une attention cognitive soutenue à l'égard de l'œuvre d'art. Pour expliquer ces interprétations multiples, la métaphore est à nouveau présentée comme le dispositif clé pour expliquer la puissance épistémique de l'esthétique, basée sur une boucle de rétroaction interprétative de la part du spectateur. Pourtant, dans le contexte du code source, cette interprétation est toujours assombrie par son pendant ma-

chine - comment l'ordinateur interprète le programme.

Enfin, cette approche de la facilité cognitive trouve enfin un écho dans la vision que Gregory Chaitin, informaticien et mathématicien, propose de la compréhension comme compression. En considérant que la compréhension d'un sujet est corrélée à la charge cognitive moindre ressentie lors du raisonnement sur ce sujet, Chaitin développe un point de vue selon lequel un individu comprend mieux grâce à un modèle correctement ajusté - un modèle qui peut expliquer le plus avec le moins (Zenil, 2021). En ce sens, l'esthétique aide à comprimer les concepts, ce qui permet à l'individu d'en retenir davantage dans sa mémoire à court terme et d'en saisir une image plus complète.

Bien que ces autres types d'expériences restent valables lors de l'appréhension d'un tel objet, nous nous concentrons ici sur ce type d'expérience spécifique : l'approche cognitive de l'expérience esthétique. [5]. Nous allons maintenant voir comment ce type d'expérience peut se décliner dans les domaines esthétiques adjacents au code.

Les dispositifs littéraires syntaxiques permettent aux lecteurs de s'engager cognitivement avec un contenu particulier ; ils permettent la construction de modèles mentaux d'un récit particulier, grâce à un réseau de métaphores, d'allusions, d'interprétations ambiguës et de chronotopes (Bakhtin, 1981). Ces dispositifs littéraires s'appliquent également au code source, en particulier la façon dont l'utilisation de jetons de machine et l'interprétation humaine suggèrent une expérience esthétique par le biais de métaphores et de marqueurs particuliers nécessaires pour donner un sens au temps et à l'espace d'un programme informatique, qui diffèrent radicalement de ceux d'un texte imprimé. Cette compréhension d'un temps et d'un espace étrangers est en effet essentielle à la création

[5]Pour revenir à Goodman, il décrit une telle expérience comme impliquant: *de faire des discriminations délicates et de discerner des relations subtiles, d'identifier des systèmes de symboles et ce que ces caractères dénotent et exemplifient, d'interpréter des œuvres et de réorganiser le monde en termes d'œuvres d'art et d'œuvres en termes de monde.* (Goodman, 1976)

d'une carte mentale du monde de l'histoire (dans la fiction) ou du monde de référence (dans la non-fiction).

L'utilisation du terme "carte" implique également un type de territoire spécifique, rendu possible par le numérique. En tant qu'hybride entre la planéité de l'imprimé et la profondeur du code, Ryan et Murray - parmi beaucoup d'autres - identifient le récit numérique comme un récit hautement spatialisé (Murray, 1998). Selon Ryan, cette caractéristique n'est que le reflet de l'architecture interne du code source. Dans cette veine, nous nous tournons maintenant vers l'architecture en tant que discipline pour étudier comment l'environnement bâti permet quel type de compréhension, et comment ces possibilités peuvent se traduire dans l'espace des textes programmés.

L'architecture offre une certaine heuristique dans la recherche de caractéristiques esthétiques que le code peut présenter. Partant de l'idée naïve que la forme doit suivre la fonction, nous examinons comment la théorie des modèles d'Alexander (Alexander et al., 1977), et son influence significative sur la communauté des programmeurs (Gabriel, 1998), n'indique pas seulement un conditionnement explicite de la forme à sa fonction (auquel cas nous écririons tous du code Assembly fait à la main), mais plutôt une qualité insaisissable, mais présente, qui dépend à la fois du problème et du contexte.

Outre la fonction du programme en tant que composante essentielle du jugement esthétique, notre travail montre aussi que les textes des programmes peuvent présenter une qualité qui est consciente du contexte que l'auteur et le lecteur apportent avec eux, et du contexte qu'il leur fournit, ce qui le rend habitable. L'architecture logicielle et les modèles ne sont cependant pas explicitement loués pour leur beauté, peut-être parce qu'ils ne tiennent pas compte de ces contextes, étant donné qu'ils sont des abstractions de niveau supérieur ; cela implique que les solutions génériques sont rarement des solutions élégantes. Ainsi, il existe en architecture un lien indéniable entre le beau, l'artisanal et l'universel.

L'esthétique est aussi étroitement impliquée dans la considération des objets mathématiques, dans l'appréciation de leur représentation symbolique (Hardy, 2012; Poincaré, 1908), et comme heuristique vers une représentation à valeur positive (Sinclair, 2004). En particulier, la dichotomie entre les entités mathématiques (théorèmes) et leurs représentations (preuves) fait écho à la distinction que nous avons observée en programmation entre algorithme et mise en œuvre. Bien que les entités abstraites possèdent des qualités spécifiques qui sont appréciées positivement, c'est leur implémentation - c'est-à-dire leur manifestation textuelle - qui tend à être le lieu du jugement esthétique. L'esthétique complète également la notion plus courante de pensée rationnelle scientifique, d'un individu qui raisonne sur un problème d'une manière linéaire, étape par étape. Au contraire, nous avons vu que l'apparence, et le jugement de cette apparence, sert également de guide pour établir des énoncés mathématiques vrais et élégants.

En fin de compte, l'esthétique en mathématiques contribue à la représentation d'un objet mathématique, permettant ainsi d'accéder à la nature conceptuelle et aux implications de cet objet, tout en fournissant des heuristiques utiles pour établir un nouvel objet. Ce qu'il reste à faire, et qui sera abordé dans la section suivante, est de "*réifier cette métalogique comme un ensemble de règles, d'axiomes ou de pratiques.*" (Root-Bernstein, 2002), en établissant comment ces approches peuvent correspondre à l'esthétique du code source.

Esthétique spatiale des codes sources

Le logiciel est donc la représentation d'une idée dans des configurations matérielles spécifiques. Le support immédiat de cette représentation, du point de vue du programmeur, est le langage de programmation dans lequel l'idée est écrite. Jusqu'à présent, les langages de programmation ont été laissés de côté lorsqu'il s'est agi d'examiner quels aspects sen-

suels du code source permettaient d'obtenir ce qui pouvait être considéré comme un "beau" texte de programme. Pourtant, la relation entre la sémantique (structure profonde) et sa représentation syntaxique (structure de surface) est encadrée par les langages de programmation, car ils définissent l'organisation légale de la forme.

Les langages de programmation sont à la fois des outils et des environnements, et de surcroît éminemment *symboliques*, manipulant et façonnant la matière symbolique. L'examen de ces langages dans une perspective goodmanienne fournit une toile de fond permettant d'étudier leur pouvoir de communication et d'expression. La tension apparaît lorsqu'il s'agit des critères d'univocité, d'un point de vue humain. La composante des programmes la plus proche du langage naturel, à savoir les noms de variables et de fonctions, présente toujours un risque d'ambiguïté[6]. Nous considérons cette ambiguïté à la fois comme une opportunité productive pour la créativité et comme un obstacle à la fiabilité du programme.

Si les langages de programmation sont des systèmes de symboles esthétiques, ils peuvent permettre l'expression, avant tout de concepts informatiques. C'est dans le traitement de concepts particulièrement complexes que les langages de programmation se distinguent également en termes de valeur. Les différences dans la conception et l'utilisation des langages de programmation reviennent donc à des différences de style[7].

L'utilisation concrète des langages de programmation s'opère à un autre niveau de formalité : si les paradigmes de programmation sont des stratégies spécifiées par les concepteurs du langage, ils sont également com-

[6] Par exemple, `int numberOfFlowers` fait-il référence au nombre actuel de fleurs en mémoire ? Au nombre total de fleurs potentielles ? À un type spécifique de nombre dont la dénomination est celle d'une fleur ?

[7] Comme le dit Niklaus Wirth : *"Pour beaucoup, les arguments stylistiques peuvent sembler hors de propos dans un environnement technique, parce qu'ils semblent n'être qu'une question de goût. Je m'oppose à ce point de vue et affirme au contraire que les éléments stylistiques sont les parties les plus visibles d'une langue. Ils reflètent très directement l'esprit du concepteur et se reflètent dans chaque programme écrit."* (Wirth, 2003)

plétés par les tactiques des développeurs de logiciels. Ces pratiques se cristallisent, par exemple, dans l'écriture idiomatique. En linguistique traditionnelle, l'idiomaticité désigne la manière dont une langue donnée est utilisée, par opposition à ses alternatives possibles, syntaxiquement correctes et sémantiquement équivalentes. Elle renvoie donc à la manière dont une langue est une construction sociale, expérientielle, reposant sur une communication intersubjective (Voloshinov & Bachtin, 1986). Cette composante sociale des langages de programmation est donc liée à la manière dont on écrit "correctement" un langage. En ce sens, les communautés de langages de programmation s'apparentent à des clubs d'amateurs, avec leurs noms, leurs mascottes, leurs conférences et leurs blagues internes. Écrire dans un langage particulier peut être dû à des exigences externes, mais aussi à des préférences personnelles[8]. Un idiome dans un langage de programmation dépend donc de l'interprétation sociale des paradigmes de programmation formels. Une telle interprétation se manifeste également dans les documents créés et détenus par la communauté.

Nous avons vu comment les langages de programmation peuvent être soumis à un jugement esthétique, mais ces critères esthétiques ne sont là que pour soutenir l'écriture d'un bon code (c'est-à-dire fonctionnel et beau). Ce support existe via des choix de conception (abstraction, orthogonalité, simplicité) (Stansifer, 1994), mais aussi via les usages pratiques des langages de programmation, notamment en termes d'idiomaticité et de décoration syntaxique, permettant à certains langages d'être plus lisibles que d'autres. Comme tous les outils, c'est leur utilisation (adroite) qui importe, plutôt que leur conception, et ce sont les problèmes qu'ils servent à traiter,

[8] *"Je pense qu'un langage de programmation devrait avoir pour philosophie d'aider notre pensée, et donc Ruby se concentre sur la productivité et le plaisir de programmer. D'autres langages de programmation, par exemple, se concentrent plutôt sur la simplicité, la performance ou quelque chose comme ça. Chaque langage de programmation a une philosophie et une conception différentes. Si vous vous sentez à l'aise avec la philosophie de Ruby, cela signifie que Ruby est votre langage.* (Matsumoto, 2019)

et la manière dont ils sont traités, qui déterminent en fin de compte si un texte de programme dans ce langage présentera ou non des caractéristiques esthétiques.

Ce concept d'adéquation est également lié à l'honnêteté matérielle. Le fait qu'un programmeur tende à identifier sa pratique à l'artisanat implique qu'il travaille avec des outils et des matériaux. Les langages de programmation étant leurs outils, et l'informatique leur matériau, on peut étendre le concept d'honnêteté matérielle au code source (Sennett, 2009). Dans ce cas, travailler avec et dans le respect du matériel et des outils disponibles est une preuve d'excellence dans la communauté des praticiens, et aboutit à un artefact qui est en harmonie et bien adapté à l'environnement technique qui lui a permis d'exister. Un code source écrit conformément aux principes et aux possibilités de son langage de programmation est donc plus susceptible de recevoir un jugement esthétique positif. Par ailleurs, l'écriture idiomatique s'accompagne d'une caractéristique indépendante du langage, mais dépendante du groupe: le style de programmation (Depaz, 2022).

Les discours des programmeurs de notre corpus ne contiennent pas de critères unidimensionnels, mais plutôt des critères qui peuvent être appliqués à plusieurs niveaux de lecture. Certains ont tendance à se référer davantage à la conception globale du code examiné, tandis que d'autres se concentrent sur les caractéristiques formelles spécifiques présentées par un jeton donné ou des successions de jetons dans un extrait de code source. Pour répondre à cette variabilité, nous empruntons à John Cayley la distinction entre structures, syntaxes et vocabulaires (Cayley, 2012). Le cadre de Cayley nous permettra de prendre en compte un aspect essentiel du code source : celui des échelles auxquelles le jugement esthétique opère. Des comptes rendus psychologiques de la compréhension du code source aux utilisations de l'espace dans l'esthétique spécifique à un domaine en ou l'une des spécificités du code source est les multiples dimensions de sa structure profonde cachée derrière la disposition bidimensionnelle d'un

fichier texte, et la nécessité pour les programmeurs de naviguer dans un tel espace.

La structure est définie par l'emplacement relatif d'une déclaration particulière dans le contexte plus large du texte programmé, ainsi que par le regroupement de déclarations particulières les unes par rapport aux autres et par rapport aux autres groupes de déclarations dans le texte programmé, qu'il s'agisse d'un même fichier, d'une série de fichiers ou d'un réseau tentaculaire de dossiers et de fichiers. Cela inclut également les questions de formatage, d'indentation et de mise en place de la charpente en tant qu'arrangements formels purement basés sur des modèles, comme on le voit dans la mesure où ils affectent l'ensemble du texte programmé.

La syntaxe concerne la disposition locale des jetons au sein d'une déclaration ou d'un bloc de déclarations, y compris le flux de contrôle, les déclarations d'itérateurs, les déclarations de fonctions, etc. La syntaxe comprend également les choix spécifiques au langage - les idiomes - et généralement le type de déclarations nécessaires pour représenter au mieux la tâche requise.

Enfin, le vocabulaire fait référence aux éléments du code source définis par l'utilisateur, sous la forme de noms de variables, de fonctions, de classes et d'interfaces. Le vocabulaire du code source est constitué à la fois de mots-clés réservés (que l'ordinateur "comprend" parce qu'ils sont explicitement mentionnés par les concepteurs du langage) et de mots-clés définis par l'utilisateur, c'est-à-dire les mots uniques que les auteurs définissent eux-mêmes et qui ne sont pas connus d'un lecteur qui connaîtrait déjà les mots-clés du langage. Contrairement aux deux catégories précédentes, c'est donc la seule où l'auteur peut inventer de nouveaux mots, et c'est celle qui se rapproche le plus des métaphores de la littérature traditionnelle.

La spécificité du code source est qu'il agit comme une interface technolinguistique entre deux créateurs de sens : l'homme et la machine. Alors que la machine a une définition très précise et opérationnelle du

sens, les programmeurs ont tendance à mobiliser différentes modalités afin de donner un sens au système qui leur est présenté par le biais de cette interface textuelle. Parmi ces modalités, on peut citer le recours aux techniques littéraires (sous forme de métaphores), à l'architecture (sous forme d'organisation structurelle basée sur des modèles), aux mathématiques (sous forme d'élégance symbolique) et à l'artisanat (sous forme d'adéquation matérielle et de fiabilité).

En tant que manifestation formelle impliquant, dans le contexte d'un objet artisanal, un producteur et un récepteur, l'esthétique contribue à l'établissement d'espaces mentaux. Dans des domaines tels que les mathématiques ou la littérature, les espaces mentaux peuvent représenter des théorèmes ou des émotions ; dans le code source, cependant, ils acquièrent une dimension plus fonctionnelle. En tant que tels, ils communiquent également des combinaisons d'*états* et de *processus*.

L'expressivité des textes programmés repose donc sur plusieurs mécanismes esthétiques, reliés dans l'espace entre une compréhension métaphorique des humains et une compréhension fonctionnelle des machines. De la mise en page au double-sens à travers les variables et les noms de procédures (Paloque-Bergès, 2009), le double codage et l'intégration des types de données et du code fonctionnel dans un texte de programme (Cox & McLean, 2013) et une rhétorique des procédures dans leur forme écrite (Bogost, 2008), tous ces éléments activent la connexion entre les concepts de programmation et les concepts humains pour mettre l'impensable à la portée du pensable. Bien que ces techniques soient déployées différemment selon l'environnement socio-technique dans lequel le texte programmé est écrit et lu, elles contribuent néanmoins toutes à faciliter la navigation dans le texte programmé, que ce soit au même niveau d'abstraction à travers des parties du texte, ou à différents niveaux d'abstraction dans les mêmes endroits où la syntaxe fait abstraction des signifiants inutiles de l'informatique parallèle.

En fin de compte, ces manifestations esthétiques du code source

dans un texte de programme sont toutes étroitement liées à l'exécution de ce texte de programme. Notre étude comparative du code source avec l'architecture et les mathématiques nous a permis de constater que l'esthétique n'est pas sans rapport avec les idées de fonction ou d'objectif. Dans le cas de l'architecture, l'appréciation esthétique d'un bâtiment peut difficilement être faite de manière totalement indépendante de la fonction prévue du bâtiment, tandis que, dans le cas des mathématiques, l'esthétique est étroitement associée à une fonction épistémologique, et donc à une sorte de fonctionalité informationnelle.

Chaque texte programmé que nous avons examiné dans ce travail implique toujours la nécessité de fonctionner pour être correctement jugé à un niveau esthétique, mais la diversité des pratiques que nous avons soulignées semble également suggérer différentes conceptions des fonctions. En développant les différentes façons dont la fonction d'un texte de programme peut être considérée, nous défendons l'idée d'une double relation entre la fonction et l'esthétique dans le code source. Premièrement, la fonction d'un texte de programme fait partie intégrante du jugement esthétique de ce texte, à la fois parce que le statut du logiciel en tant qu'objet artisanal fait de sa fonction la structure profonde qui se manifeste à sa surface, et parce que les normes selon lesquelles elle est jugée dépendent des contextes socio-techniques dans lesquels le texte de programme est censé être utilisé. Deuxièmement, l'esthétique du code source n'est pas autotélique ; au contraire, la fonction de l'esthétique elle-même est de communiquer des informations invisibles au lecteur et à l'auteur.

Toute l'esthétique du code source est liée à une certaine conception de la fonction, impliquant la réalisation technique et l'existence interpersonnelle. Le jugement esthétique d'un texte de programme est, dans cette optique, le jugement des manifestations perceptibles dans les codes sources permettant la compréhension d'une réalisation technique selon des normes contextuelles. Ces manifestations ne sont donc pas seulement expressives (personnelles), mais avant tout communicatives (interperson-

nelles), visant à la transmission de concepts d'un individu par l'utilisation de la syntaxe de la machine à travers la double lentille de la sémantique homme-machine. En effet, le code qui n'est ni fonctionnel pour la machine, ni significatif pour un humain, a le moins de valeur possible pour les praticiens.

Dans l'écrasante majorité des cas de textes programmés, l'attente est de comprendre. Écrire un code esthétiquement agréable, c'est écrire un code qui engage cognitivement son lecteur, que ce soit explicitement pour les développeurs de logiciels, pédagogiquement pour les scientifiques, négativement pour les hackers et métaphoriquement pour les poètes. Cet engagement, à son tour, suppose une reconnaissance du lecteur de la part de l'auteur. La reconnaissance de l'existence de l'autre en tant que lecteur et co-auteur implique une reconnaissance en tant qu'autre généralisé, dans le sens où tout le monde peut théoriquement lire et modifier du code, mais aussi en tant qu'autre spécifié, dans le sens où l'autre possède un ensemble particulier de compétences, de connaissances, d'habitudes et de pratiques issues de la diversité des communautés de programmation. Cette position, entre le général et le particulier, montre la composante éthique d'une pratique esthétique : reconnaître à la fois la similitude et la différence dans l'autre, et communiquer avec un pair par le biais de systèmes de symboles spécifiques.

Conclusion

Les langages de programmation peuvent donc être considérés matériellement, comme *l'interaction entre les caractéristiques physiques d'un texte et ses stratégies de signification* (Hayles, 2004), qui dépendent à leur tour des dynamiques sociotechniques. En tant qu'interface avec l'ordinateur, les langages de programmation, sans trop déterminer la pratique des programmeurs ou le contenu de ce qui est programmé, influencent néanmoins

la manière dont cela peut être dit, par le biais d'idiosyncrasies et de dispositifs stylistiques. Nous avons ainsi établi le statut esthétique du code source en tant que support, et son existence entre technique et social, expressif et communicatif, individuel et collaboratif.

Nous avons ensuite présenté un cadre pour l'esthétique du code source, sous le double angle de la compression sémantique et de la navigation spatiale. Pour ce faire, nous sommes partis d'une approche par niveaux des points où les décisions esthétiques peuvent intervenir dans le code source, c'est-à-dire à travers la structure, la syntaxe et le vocabulaire. En élargissant cette approche, nous avons ensuite montré comment ces différents niveaux impliquent un engagement avec des couches sémantiques : entre le lecteur humain, le lecteur automatique et le domaine du problème. La minimisation de la syntaxe tout en représentant au mieux les différents concepts impliqués à ces différents niveaux aboutit à une compression sémantique. Un code source ayant une valeur esthétique est un code qui équilibre les techniques syntaxiques, l'organisation structurelle et les choix métaphoriques afin de communiquer l'intention socio-technique d'un artefact fonctionnel. À son tour, la compression sémantique soutient les changements d'échelles ou de perspectives que le programmeur engagé doit opérer lorsqu'il navigue dans son exploration non linéaire d'un texte de programme.

En fin de compte, nous avons montré que les représentations métaphoriques du code, représentations du code comme language, comme architecture, et comme matériau, se retrouvent dans l'expression d'une conception d'un espace sémantique dynamique. Cette conception spatiale infuse donc les propriétés esthétiques propres du code, en ce que la syntaxe et la structure du code source se concentrent principalement à la description et à la navigation d'espaces conceptuels, même si les différents contextes socio-techniques dans lesquels ces codes sources sont écrits vont eux-mêmes moduler la nature de ces espaces.

References

Abelson, H., Sussman, G. J., & Sussman, J. (1979). *Structure and Interpretation of Computer Programs - 2nd Edition*. Justin Kelly.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press.

Bakhtin, M. M. M. M. (1981). *The Dialogic Imagination : Four Essays*. Austin : University of Texas Press.

Beardsley, M. C. (1970). The Aesthetic Point of View. *Metaphilosophy*, *1*(1), 39–58.

Black, M. J. (2002). The art of code. *Dissertations available from ProQuest*, (pp. 1–228).

Bogost, I. (2008). The Rhetoric of Video Games. In K. Salen (Ed.) *The Ecology of Games: Connecting Youth, Games and Learning*. Cambridge, MA: The MIT Press.

Butler, B. (2012). On Programmer Archaeology.

Cayley, J. (2012). The Code is not the Text (Unless It Is the Text) › electronic book review.

Cohendet, P., Creplet, F., & Dupouët, O. (2001). Organisational innovation, communities of practice and epistemic communities: The case of linux. In A. Kirman, & J.-B. Zimmermann (Eds.) *Economics with Heterogeneous Interacting Agents*, (pp. 303–326). Berlin, Heidelberg: Springer Berlin Heidelberg.

Cox, A., & Fisher, M. (2009). Programming Style: Influences, Factors, and Elements. In *2009 Second International Conferences on Advances in Computer-Human Interactions*, (pp. 82–89).

Cox, G., & McLean, C. A. (2013). *Speaking Code: Coding as Aesthetic and Political Expression*. MIT Press.

Cramer, F. (2019). *Exe.cut(up)able statements: Poetische Kalküle und Phantasmen des selbstausführenden Texts*. Wilhelm Fink.

Depaz, P. (2022). Discursive Strategies in Style Guides Negotiation on GitHub. *RESET. Recherches en sciences sociales sur Internet*, *11*(11).

Depaz, P. (2023). Stylistique de la recherche linguistique en IA: De LISP à GPT-3. In A. Gefen (Ed.) *Créativités Artificielles – La Littérature et l'art à l'heure de l'intelligence Artificielle*. Les Presses du Réel.

Detienne, F. (2001). *Software Design – Cognitive Aspect*. Springer Science & Business Media.

Dijk, T. A., & Kintsch, W. (1983). *Strategies of Discourse Comprehension*. New York, NY: Academic Press Ltd.

Foote, B., & Yoder, J. (1997). Big Ball of Mud.
URL `http://www.laputan.org/mud/mud.html\#BigBallOfMud`

Gabriel, R. P. (1998). *Patterns of Software: Tales from the Software Community*. Oxford University Press.

Gabriel, R. P., & Goldman, R. (2001). *Mob Software: The Erotic Life of Code*.

Genette, G. (1993). *Fiction & Diction*. Cornell University Press.

Gibbons, J. (2012). The beauty of simplicity. *Communications of the ACM*, *55*(4), 6–7.

Goodman, N. (1976). *Languages of Art*. Indianapolis, Ind.: Hackett Publishing Company, Inc., 2nd edition ed.

Goody, J. (1986). *The Logic of Writing and the Organization of Society*. Studies in Literacy, the Family, Culture and the State. Cambridge University Press.

Graham, P. (2003). Hackers and Painters.
URL http://www.paulgraham.com/hp.html

Hardy, G. H. (2012). *A Mathematician's Apology*. Canto Classics. Cambridge: Cambridge University Press.

Hayes, B. (2015). Cultures of Code. *American Scientist*, *103*(1).

Hayles, N. K. (2004). Print Is Flat, Code Is Deep: The Importance of Media-Specific Analysis. *Poetics Today*, *25*(1), 67–90.

Hayles, N. K. (2010). *My Mother Was a Computer: Digital Subjects and Literary Texts*. University of Chicago Press.

Hill, R. K. (2016). What Makes a Program Elegant?
URL https://cacm.acm.org/blogs/blog-cacm/208547-what-makes-a-program-elegant/fulltext

Kintsch, W., & van Dijk, T. A. (1978). Toward a model of text comprehension and production. *Psychological Review*, *85*(5), 363–394.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. USA: Addison Wesley Longman Publishing Co., Inc.

Kudra, A. (2020). AoC \textbackslashtextbackslashtextbar Art of Coding – The Demoscene as Intangible World Cultural Heritage. In C. Yackel, R. Bosch, E. Torrence, & K. Fenyvesi (Eds.) *Proceedings of Bridges 2020: Mathematics, Art, Music, Architecture, Education, Culture*, (pp. 479–480). Phoenix, Arizona: Tessellations Publishing.

Lakoff, G., & Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press.

Lévy, P. (1992). *De la programmation considérée comme un des beaux-arts*. Textes à l'appui. Anthropologie des sciences et des techniques. Paris: Éd. la Découverte.

Mackenzie, A. (2006). *Cutting Code: Software and Sociality*. Peter Lang.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education.

Mateas, M., & Montfort, N. (2005). A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics. *undefined*.

Matsumoto, Y. (2019). Yukihiro Matsumoto: "Ruby is designed for humans, not machines".
URL `https://evrone.com/blog/yukihiro-matsumoto-interview`

McLean, A. (2004). Hacking Perl in Nightclubs.
URL `https://perl.com/pub/2004/08/31/livecode.html/`

Montfort, N., Baudoin, P., Bell, J., Bogost, I., & Douglass, J. (2014). *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*. The MIT Press, illustrated edition ed.

Mullet, D. R. (2018). A General Critical Discourse Analysis Framework for Educational Research. *Journal of Advanced Academics*, *29*(2), 116–142.

Murray, J. H. (1998). *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. Cambridge, MA, USA: MIT Press.

Ong, W. J. (2012). *Orality and Literacy: 30th Anniversary Edition*. London: Routledge, 3 ed.

Oram, A., & Wilson, G. (Eds.) (2007). *Beautiful Code: Leading Programmers Explain How They Think*. Beijing ; Sebastapol, Calif: O'Reilly Media, 1st edition ed.

Overflow, S. (2013). How can you explain "beautiful code" to a non-programmer?

Paloque-Bergès, C. (2009). *Poétique des codes sur le réseau informatique*. Archives contemporaines.

Penny, S. (2019). *Making Sense: Cognition, Computing, Art and Embodiment*. MIT Press.

Pineiro, E. (2003). *The Aesthetics of Code : On Excellence in Instrumental Action*. Ph.D. thesis, KTH, Superseded Departments, Industrial Economics and Management.

Poincaré, H. (1908). *Science et méthode*. Paris: E. Flammarion.

Polanyi, M., & Grene, M. (1969). *Knowing and Being; Essays*. [Chicago] University of Chicago Press.

Ricoeur, P. (2003). *The Rule of Metaphor: The Creation of Meaning in Language*. Psychology Press.

Root-Bernstein, R. S. (2002). Aesthetic cognition. *International Studies in the Philosophy of Science*, *16*(1), 61–77.

Schummer, J., MacLennan, B., & Taylor, N. (2009). Aesthetic Values in Technology and Engineering Design. In A. Meijers (Ed.) *Philosophy of Technology and Engineering Sciences*, Handbook of the Philosophy of Science, (pp. 1031–1068). Amsterdam: North-Holland.

Sennett, R. (2009). *The Craftsman*. Yale University Press.

Sinclair, N. (2004). The Roles of the Aesthetic in Mathematical Inquiry. *Mathematical Thinking and Learning*, *6*(3), 261–284.

Soloway, E., & Ehrlich, K. (1984). Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering*, *SE-10*(5), 595–609.

Soloway, E., Ehrlich, K., & Bonar, J. (1982). Tapping into tacit programming knowledge. In *Proceedings of the 1982 Conference on Human Factors in Computing Systems*, (pp. 52–57).

Stansifer, R. (1994). *Study of Programming Languages, The*. Englewood Cliffs, N.J: Prentice Hall, 1st edition ed.

Steele, G. L. (1977). Macaroni is better than spaghetti. *ACM SIGPLAN Notices*, *12*(8), 60–66.

Temkin, D. (2023). The Less Humble Programmer. *Digital Humanities Quarterly*, *017*(2).

Tufte, E. R. (2001). *The Visual Display of Quantitative Information*. Graphics Press.

Voloshinov, V. N., & Bachtin, M. M. (1986). *Marxism and the Philosophy of Language*. Harvard University Press.

Weinberg, G. M. (1998). *The Psychology of Computer Programming*. Dorset House Pub.

Wirth, N. (2003). The Essence of Programming Languages. In L. Böszörményi, & P. Schojer (Eds.) *Modular Programming Languages*, Lecture Notes in Computer Science, (pp. 1–11). Berlin, Heidelberg: Springer.

Zenil, H. (2021). Compression is Comprehension, and the Unreasonable Effectiveness of Digital Computation in the Natural World.