Regard littéraire sur la programmation comprise comme une écriture

Frédéric Drouillon

Frédéric Drouillon

Doctorant au CREAD, EA n°3875
fdr@free.fr

Regard littéraire sur la programmation comprise comme une écriture

Résumé
Nous appréhendons la pratique de la programmation en accentuant des questions d'écriture, de langage, de culture et d'expression qui ne requièrent pas nécessairement de prérequis scientifiques. L'objectif n'est pas de traiter des sujets fondamentaux de la recherche en informatique mais simplement de voir, dans une perspective didactique, comment la programmation peut être abordée différemment selon la diversité des publics actuellement concernés par elle : enfants, collégiens, lycéens, étudiants de culture littéraire, artistique et scientifique ainsi qu'un large éventail de profils dans le domaine de la formation professionnelle. Notre premier propos consiste à extraire de la programmation ce qui peut la rapprocher d'un texte. Ensuite d'envisager une analyse de ce texte selon la notion théorique de « figures de pensée » empruntée à l'anthropologue Jack Goody. L'écriture du texte informatique, à l'image de celle du texte littéraire, se révèle être une activité d'auteur qui engage des aspects affectifs et émotionnels pertinents à prendre en compte lors des apprentissages de la programmation. Extraite de son contexte théorique, technique et scientifique, la pratique de la programmation, sans pour autant être facile, peut être proposée avec pragmatisme à de plus nombreux publics.

Mots clés : programmation, langage, code, texte, écriture, technologies de pensée, expression, art

1 Au commencement un texte

Bernard Chazelle (2011) présente la machine de Turing comme un petit train qui circule sur une voie ferrée illimitée et divisée en cases avec un chiffre (une donnée) par case. Selon les instructions qu'il reçoit le train peut

se déplacer d'une case à la fois, d'un côté ou de l'autre, et lire ou écrire. Alan Turing montre qu'il est possible de faire exécuter par cet automate tous les calculs possibles à partir de listes d'instructions élémentaires simples. Cette liste d'instructions qui permet l'accomplissement d'une tâche constitue un algorithme. L'idée vient alors d'écrire également cette liste sur la voie comme le sont les données, le texte final obtenu composant un programme. Ce modèle, qui inaugure un système textuel hors du commun, se trouve complété avec les concepts de fonction et d'application (lambda-calcul) donnés par Alonzo Church[1]. Ces concepts amorcent le développement de langages spécifiques : les langages de programmation.

1.1 L'algorithme comme scénario d'actions

Un algorithme se définit par une suite finie d'opérations organisée en vue d'accomplir une tâche. Trouver un algorithme revient à décomposer la réalisation finale de l'action en une succession d'étapes et de moments intermédiaires et nécessaires. Maurice Nivat, un des pionniers français de la recherche en informatique et algorithmique[2], le résume dans une définition concise : « un algorithme c'est une façon de faire ». Écrire un algorithme revient selon lui à scénariser une manière de s'y prendre pour arriver à un but. Dans cette optique l'algorithme apparaît dédoublé avec d'une part l'aspect descriptif d'une suite d'opérations et d'autre part son accomplissement effectif. En tant que description il exprime un déroulement de l'action ce qui le rapproche d'une narration et, en extrapolant un peu, ce qui semble lui donner la possibilité de raconter quelque chose.

Un algorithme peut traduire et représenter des actions prises dans la plupart des activités humaines et plus largement les algorithmes peuvent fournir des descriptions pour toutes sortes de « processus ». La définition générale d'un processus[3] est presque similaire à celle de l'algorithme : « Suite continue de faits, de phénomènes présentant une certaine unité ou

1 Pour une présentation rapide du lamba-calcul : <https://fr.wikipedia.org/wiki/Lambda-calcul>.

2 Maurice Nivat : <https://fr.wikipedia.org/wiki/Maurice_Nivat>.

3 Celle donnée ici provient du dictionnaire TLFI (Trésor de la langue française informatisée) sur le site CNRTL (Centre national de ressources textuelles et lexicales) : <http://www.cnrtl.fr/definition/processus>.

une certaine régularité dans leur déroulement […] Ensemble d'opérations successives, organisées en vue d'un résultat déterminé. »

Qu'un phénomène puisse s'exprimer sous la forme d'un algorithme ne signifie pas pour autant qu'il puisse toujours donner lieu à un programme. L'algorithme cependant, grâce très largement à l'informatique, ouvre des possibilités d'interpréter, de simuler, de se représenter et par là de comprendre, d'expliquer, d'interroger, de rendre lisibles et intelligibles nombre de phénomènes. Il permet également d'imaginer et d'agir sur le réel. Le système textuel initié par la machine de Turing a ouvert l'espace inouï d'une écriture renouvelée et comme Bernard Chazelle (2013) le souligne « l'algorithme n'est pas tant un objet utile, tel qu'une voiture, un parapluie, ou un cure-dents, qu'une façon différente de penser ».

1.2 Expression et jeu dans des langages spécialisés

Pour extraire un algorithme d'un processus ou d'une action il faut pouvoir les décrire et les analyser. On a donc besoin d'un langage. Le langage à utiliser dépend du domaine : un parent qui apprend à son enfant à s'habiller le fera dans sa langue naturelle. Action ou processus peuvent être décrits dans un langage naturel, ils peuvent être aussi décrits en utilisant un système formel de notation appartenant à une discipline scientifique ou artistique et, afin d'être interfacés avec des machines, ils peuvent être décrits dans des langages de programmation.

Du point de vue d'une culture artistique et littéraire un langage de programmation fusionne des figures de pensée (ce concept est précisé ci-dessous en 2.) avec des contraintes techniques d'une machine pour rendre possible l'écriture de programmes capables de la piloter. Le programmeur se retrouve un petit peu dans la situation d'un marionnettiste : via un langage de programmation il articule et établit un fonctionnement que la machine exécute en différé comme s'il s'agissait d'une marionnette. L'image est assez nette en robotique ; elle l'est un peu moins dans les domaines des réseaux.

Les langages de programmation définissent en général deux catégories de vocabulaire. En nous fondant sur le langage C nous trouvons d'une part un vocabulaire d'instructions primitives (les sauts, les branchements, les boucles) sur la base desquelles le programmeur va construire des instructions plus évoluées, notamment sous la forme de fonctions. D'autre part un

vocabulaire pour le stockage des informations c'est-à-dire pour les données. Il s'agit d'une multiplicité de types de variables qui déterminent des espaces mémoires afin d'enregistrer des valeurs sur lesquelles sont opérées les instructions. La dualité théorique initiale instructions-données du modèle de Alan Turing est palpable dans la plupart des langages de programmation.

Pour chacun de ces langages une grammaire définit et ordonne les possibilités d'écriture comme dans tout langage et ces vocabulaires primitifs sont régis par des syntaxes plus ou moins différentes d'un langage à l'autre. Ces syntaxes sont en général assez simples sans pour autant être aisées à manipuler. En particulier, les rigidités d'un langage de programmation dues à la machine sous-jacente diffèrent des difficultés que l'on peut rencontrer avec des langues naturelles et c'est souvent un aspect déstabilisant pour les apprenants des disciplines littéraires. Cependant tout programmeur apprend à jouer et à maîtriser les formalismes de ses langages de programmation jusqu'à finalement les intégrer, au point de les « oublier » lorsqu'il pense avec eux.

2 Des figures de pensée libres d'interprétation

2.1 Héritage technologique du texte littéraire

La syntaxe d'un texte littéraire supporte et incorpore des figures conceptuelles comme des listes, des tableaux ou des arborescences. Ces figures constituent des moyens de penser, une manière graphique de raisonner et de connaître, une technologie de l'intellect. Jack Goody (1979) fait apparaître que le rôle de ces figures à l'œuvre dans l'élaboration des pensées permet la distinction et la comparaison entre cultures orales et cultures écrites. Écrire ne consiste pas à « enregistrer la parole », écrire c'est aussi « se donner le moyen d'en découper et d'en abstraire les éléments, de classer les mots en listes et combiner les listes en tableaux ».

Les technologies du texte littéraire étudiées par Jack Goody, tableaux, classement, liste et recette, sont toutes présentes dans le texte informatique et dans des versions très augmentées. On y trouve en plus des algorithmes de tri et de nombreuses variations sur les listes. Tout cet ensemble riche et diversifié de figures s'intègre dans ce que l'on appelle « les structures de données ».

Le principe de la recette s'utilise souvent comme métaphore pour illustrer ce qu'est un algorithme : une suite d'instructions à effectuer les unes après les autres comme dans une recette de cuisine. Mais le plus stupéfiant dans ce rapprochement tient à la manière de penser à partir d'une recette qui ressemble à la manière dont on peut jouer avec un algorithme : « [les recettes] sont des textes qui prescrivent une action, qui formulent un programme ; ils conduisent à une extension des connaissances dont disposent le spécialiste et même l'amateur ; ils invitent à ‹mettre à l'épreuve› les recettes (à les ‹tester›, à les ‹expérimenter›) et à procéder à une évaluation comparative des résultats. » (Goody, 1979).

C'est exactement ce que peut être amené à faire un programmeur avec un algorithme pour l'étudier, l'augmenter, le détourner ou l'adapter.

Dans les figures conceptuelles étudiées par Jack Goody il manque l'arborescence, manifeste cependant au VIIe siècle dans les travaux d'Isidore de Séville (devenu d'ailleurs pour cette raison saint patron des informaticiens en 2002). Des arborescences très sophistiquées, avec toutes sortes de formes et d'implémentations, sont intégrées à de nombreux textes informatiques.

2.2 Apport technologique du texte informatique

Sur le même registre littéraire et sur les pas de Jack Goody notre idée consiste à repérer dans le texte informatique des figures intellectuelles à partir desquelles peuvent s'élaborer et s'exprimer des pensées. En nous référant au langage C et à ses dérivés (C++, C#, Objective-C, Perl, Python, Java, php, Ruby, JavaScript, Groovy, Scala, etc.) nous suggérons, afin d'inaugurer une recherche sur des approches littéraires possibles de ces figures, cinq exemples du texte informatique assez caractéristiques et largement employés : le si-sinon, la notion de variables et de nombres, la boucle, la fonction et pour la programmation orientée objet (POO) le principe des classes et des objets. Très brièvement en voici quelques interprétations.

Si-Sinon

Ce formalisme d'apparence logique est solidaire du fonctionnement de la machine et mis au service de ce que l'on souhaite voir faire à la machine, mais il n'enferme pas les raisonnements à l'origine des scénarios projetés sur la machine. « Si le personnage qui entre est vert alors le joueur

s'envole… » aucune logique, juste une invention scénaristique, le si-sinon devient un instrument dont on joue. Il intervient dans le raisonnement mais il n'emprisonne pas le raisonnement.

Variables et nombres

Sous la forme de variables les nombres acquièrent des propriétés remarquables. Les variables traduisent des particularités, des marques, des signes, des attributs, des émotions, etc. toutes sortes de caractéristiques qui permettent par exemple, dans un scénario, de qualifier un personnage, une chose ou une action, les nombres en donnent les valeurs et les intensités. Peur, anxiété, courage, loyauté, jalousie, magie… tout se retrouve traduit numériquement dans des fourchettes de valeurs. Autant de paramètres que l'on peut ensuite apprécier, comparer, mixer et confronter selon les relations souhaitées dans la trame du programme : « s'il a suffisamment de courage, il peut avancer et engager le combat sinon il doit continuer sa quête par ailleurs ». Le nombre devient une matière première, la gouache initiale à laquelle on s'habitue pour imaginer et mettre en scène, il peut représenter quasiment n'importe quoi et se traduire en ce que l'on veut. Il peut être envisagé comme l'équivalent d'un adjectif ou d'un adverbe (« il frappe doucement/fort, il parle lentement/vite »).

Boucle

Un texte algorithmique s'exécute de façon linéaire du début à la fin. La boucle permet de répéter une séquence d'instructions et donc de rompre la linéarité d'un texte en localisant le saut en arrière sur une portion de code facile ensuite à identifier. En général un programme obéit à une boucle principale qui prend fin lorsque l'utilisateur le demande. Elle permet de capturer toutes ses entrées (souris clavier, etc.) et constitue l'essence de l'interactivité.

Cette acquisition intellectuelle de la boucle dans la syntaxe même de l'écriture prend certainement part à la généralisation d'une pensée qui fonctionne par itérations successives et tâtonnements expérimentaux. Son principe se retrouve dans l'organisation circulaire et rétroactive de nombreuses situations de création, de conception et de réalisation des programmes (Drouillon, 2003).

Fonction

Une fonction permet de rassembler en une seule instruction, plusieurs instructions nécessaires à la réalisation d'une tâche. C'est un algorithme complet ou une partie d'un algorithme. Elle apparaît un peu comme un verbe. Elle

peut recevoir des valeurs en entrée et éventuellement retourner des valeurs. Dans un langage non objet une fonction peut être partagée par des données différentes. Pour que Blanche Neige croque une pomme nous écrirons une fonction « croquer » avec par exemple en entrée deux paramètres : une princesse et une pomme. Au moment où la fonction « croquer » est appelée pour être exécutée nous lui fournissons une princesse et une pomme spécifiques. Nous choisirons Blanche Neige ou n'importe quelle autre princesse, avec soit une pomme empoisonnée, soit une pomme enrichie de vitamines, ça dépend des circonstances et du rôle attribué à la pomme dans la fonction.

Classe et objets

Une classe réunit des variables et des fonctions afin de définir une entité complète dans le programme. Cette entité peut-être une chose, un personnage, une opération ou une action. À partir de la classe on décline des « objets » de la classe. La classe agit comme un moule qui permet de déterminer sur le même modèle plusieurs éléments semblables, mais dont les propriétés prennent des valeurs différentes. Blanche Neige serait probablement un objet d'une classe « Princesse » à partir de laquelle nous pourrions également obtenir une méchante reine et différentes autres personnalités, autant d'objets différents de la même classe. Banche Neige aurait une jalousie inoffensive de zéro, la méchante reine une jalousie féroce de cent sur cent.

Un grand intérêt de l'objet est qu'il suppose des mises en relation des objets entre eux. Elles s'effectuent sous forme d'échanges, avec des envois de messages pratiqués grâce aux paramètres en entrée des fonctions membres d'une classe, ou de connaissances mutuelles, avec des possibilités de partage d'adresses mémoire réalisées avec des pointeurs.

Le principe des classes et des objets fournit un autre moteur essentiel pour la pensée que l'on retrouve aussi à d'autres niveaux de l'écriture des programmes. Un objet peut être compris comme un petit programme autonome, sujet à des relations avec d'autres objets, sous-programmes comme lui, et le tout organisé dans un scénario général constitué par le programme dans son ensemble. Le programme général apparaît comme une sorte de réseau de programmes connectables les uns aux autres. Nous trouverions certainement une corrélation historique entre l'apparition des réseaux informatiques et l'évolution des langages vers la programmation orientée objet. Au départ ces propriétés d'échange et de partage entre des objets informatiques ne sont pas explicites dans un langage, elles naissent de l'appropriation des possibilités fournies par la syntaxe de l'écriture et s'imposent avec l'expérience.

Nous pourrions formuler l'hypothèse que progressivement ces échanges et partages entre objets, inhérents à la conception de nombreux programmes, se retrouvent aussi comme figures intellectuelles présentes dans la réflexion à d'autres niveaux : lorsqu'il est question d'échanges et de partages entre les acteurs d'un projet, dans son organisation, mais également dans l'imagination et la formulation de programmes qui interviennent dans les domaines de la communication, des échanges et des partages.

Les langages de programmation informatique, qui se comptent maintenant par milliers, sont à l'origine de nombreuses autres formes et figures qui interviennent dans la pensée. Il semblerait intéressant de repérer leurs influences non seulement sur la manière de programmer mais aussi plus largement sur les manières de penser ainsi que sur des comportements et des éléments de culture qui pourraient en découler.

3 Une activité d'auteur

Dans la pratique programmer n'apparaît pas comme un acte subalterne ou anodin : programmer c'est vivre quelque chose et s'y engager. Des rapprochements entre programmation et activité d'auteur au sens artistique et littéraire sont intéressants pour aborder l'expérience sensible de la pratique du programmeur et la comprendre. La programmation est depuis longtemps présente en art[4] mais l'émotionnel dans la programmation reste toujours très marginal comme objet d'étude ou de recherche. Le rapport de la personne à cette activité et son investissement demeurent *a priori* largement inexplorés. Nous pouvons cependant citer l'exemple du séminaire « codes sources » dont les organisateurs[5] se donnent pour but « de décrire ces œuvres de l'esprit [les programmes] comme des textes à part entière ».

Les personnes qui actuellement rendent compte des possibilités expressives et émotionnelles dans la programmation sont le plus souvent des personnes qui connaissent à la fois une pratique artistique et une pratique

4 Pour un historique précis des rapports entre art et programmation se référer à Lartigaud, 2011.

5 Mélès B. (CNRS, Archives Henri-Poincaré), Fournier-S'niehotta R. (CNAM), Tabourier L. (LIP6, UPMC/CNRS), séminaire code source depuis 2015 : <http://codesource.hypotheses.org/>.

de la programmation. L'intelligence des deux pratiques leur permet d'opérer des rapprochements et des comparaisons.

Vikram Chandra (2014) est un romancier qui a financé ses études littéraires en travaillant comme informaticien. Dans un ouvrage étonnant, *Geek sublime*, il montre comment ses expériences d'informaticien et de romancier s'enchevêtrent et s'entrecroisent. Il témoigne de connexions sensibles entre la programmation et des arts : « parce qu'ils [les programmeurs] créent des objets complexes, et ne se soucient pas seulement de leurs fonctions, mais aussi de leur beauté, ils agissent exactement comme les peintres et les sculpteurs ».

Paul Graham, un informaticien de formation littéraire ayant de surcroît étudié la peinture à l'académie des beaux-arts de Florence, met en évidence dans son livre (2004) des points communs entre ses expériences de la programmation et sa connaissance de la peinture. Dans cet ouvrage il s'intéresse aux « hackers » c'est-à-dire, selon la terminologie anglo-saxonne, les amateurs au sens fort, ceux qui aiment l'informatique et font de la programmation une pratique inventive. Il affirme : « en fait, de tous les différents types d'individus que j'ai rencontrés, les hackers et les peintres sont ceux qui se ressemblent le plus. Les uns et les autres ont ceci en commun : ce sont tous des créateurs. Comme les compositeurs, les architectes et les écrivains, ce qu'ils tentent de faire, c'est de créer des objets qui soient réussis ».

De nombreux aspects émotionnels propres à la musique se transposent également assez facilement ou se retrouvent en programmation informatique. Dans notre expérience de musicien compositeur et de développeur les deux pratiques s'éclairent mutuellement et c'est parfois inattendu. Par exemple un frisson similaire à celui que procure parfois la musique peut être provoqué en programmation. On peut être touché par une expérience très spécifique de beauté et de compréhension, ce qui témoigne d'aspects émotionnels et psychologiques qui entrent dans le rapport à la programmation. On peut lire Mihaly Csikszentmihalyi (2006) pour une réflexion sur le flow et la créativité qui intéresse la programmation.

David-Olivier Lartigaud constate l'apparition depuis les années 2000 d'une génération d'artistes qui opèrent un retour aux codes sources des programmes informatiques. Des artistes s'approprient un ou plusieurs langages de programmation pour en faire le fondement d'une recherche et d'une écriture. Ils s'engagent dans une expérience artistique de pratique de la programmation. La programmation pour eux ne relève pas de l'utilitaire, elle relève de l'écriture et de la composition. Cette intimité avec la

programmation suscite de leur part un regard décalé sur elle ainsi que sur les cultures scientifique et technique qui lui sont en général associées.

4 Une entrée pragmatique dans la programmation

L'expérience montre que la programmation informatique peut être ouverte à toutes sortes de personnes quels que soient leurs âges et leurs formations : des enfants, des collégiens, des lycéens ou des adultes de culture littéraires, scientifiques ou artistiques[6]. Leurs approches, leurs objectifs ainsi que le niveau des performances diffèrent mais cette diversité de publics révèle néanmoins un espace de la programmation qui n'est pas monolithique. On peut observer également que l'entrée dans la programmation n'est pas nécessairement assujettie à des prérequis scientifiques.

Cette large ouverture possible, similaire à celle que l'on retrouve par exemple en musique avec l'apprentissage des instruments de musique, se relie probablement au fait que la programmation s'acquiert essentiellement par la pratique. Effectivement, apprendre à programmer c'est un peu comme apprendre à jouer d'un instrument de musique. Dans ces domaines le savoir théorique est indissociable de l'expérimentation répétée à travers des réalisations diverses et variées et de différents niveaux. En programmation, avec le temps, la pratique seule permet d'acquérir une intuition spécifique qui permet plus facilement de repérer ou de poser des problèmes et de trouver des solutions ensuite.

4.1 Distinction entre langage et discours

Pour les apprentissages de la programmation il apparaît très utile de distinguer d'une part le langage comme technologie du discours et d'autre part le discours comme une production intentionnelle associée à l'exercice du langage. La partie langage donne des possibilités d'architecture et de

6 Nous nous référons ici à différents ateliers de programmation que nous avons animés en bibliothèques auprès d'enfants, de collégiens et de lycéens du CM2 à la terminale, auprès d'adultes amateurs de programmation, dans le cadre de cours à l'université d'Arras en licence de lettres modernes et également dans l'école supérieure d'informatique cs2i Bourgogne de Nevers, en licence et mastère d'informatique.

construction, la partie discours fait appel à des connaissances externes. L'élaboration et la réalisation d'un programme associent les deux.

Lors de l'apprentissage de la programmation les premières expérimentations d'un langage portent sur des discours simples. Moment clé pendant lequel l'apprenant se confronte à l'acquisition d'un langage doté d'une syntaxe pour lui assez rude et surtout porteuse de figures de pensées inusitées dans les langues naturelles. Ces nouveautés de syntaxes et de figures (quelques-unes décrites en 2.2) lui rendent inaccessible une trop haute complexité de discours. Quel que soit son âge il se trouve dans la situation d'un élève de CP et CE1 lors des premiers apprentissages de la lecture et de l'écriture : toute son énergie intellectuelle passe à intégrer codage et décodage ce qui rend l'accès au sens extrêmement laborieux. La complexité des discours s'ajoute progressivement, élaborée la plupart du temps avec des connaissances totalement extérieures au langage de programmation lui-même.

Ces autres connaissances nécessitent souvent des apprentissages extrinsèques. Par exemple maîtriser des modèles mathématiques pour des traitements de l'espace ou de l'image, des modèles linguistiques dans le cadre du traitement automatique de la langue, des problématiques de fouilles de données, de big data, de recherches ou d'activités professionnelles variées… Ces modèles pour être portés par un langage de programmation nécessitent des dispositions appropriées, ils doivent pouvoir *se traduire* dans un langage de programmation. Cette reconfiguration et ce portage d'un modèle externe en un discours opérationnel sur machine exigent de confronter connaissance et compréhension du sujet traité d'une part et maîtrise du langage de programmation d'autre part.

4.2 Comprendre n'est pas savoir faire

Pour les débutants confrontés à un langage de programmation, en général comprendre ne suffit pas. Il faut aussi savoir faire et dans ce domaine comprendre n'est pas savoir-faire (Drouillon, 2014).

Pour prendre une image, c'est un peu la même situation pour apprendre à jouer d'un instrument de musique. Si un professeur ou un tutoriel vidéo explique comment jouer de la guitare, vous pouvez comprendre les explications, mais vous n'êtes pas devenu guitariste ni compositeur de musique. Devenir guitariste suppose de nombreuses heures de travail sur

son instrument. Devenir compositeur et avoir des idées de morceaux de musique ou d'arrangements musicaux supposent également beaucoup de questionnements et de travail. C'est la même chose en programmation et conception informatique.

Écrire un programme nécessite un minimum d'expertise des outils fondamentaux, c'est-à-dire un minimum d'expérience et de temps consacré à l'activité. De notre point de vue, l'acquisition progressive d'une maîtrise des fondamentaux, même incomplète, permet seule de faire des hypothèses de réalisation parce qu'elle produit petit à petit cette compétence d'identifier et de poser les problèmes avec pertinence, ce qui permet d'inventer des solutions. En effet, trouver une solution pour résoudre un problème revient souvent à créer cette solution. Il s'agit d'un processus d'élaboration qui nécessite au plan de l'écriture de l'intuition, de la recherche et de la créativité.

4.3 Un apprentissage non linéaire

D'une façon générale, l'apprentissage n'est pas réductible à une mécanique simple. Pour l'apprenant la clarté se fait progressivement un peu comme un brouillard qui se lève. Au départ il ne distingue rien et petit à petit, au fur et à mesure que le brouillard se dissipe, des formes apparaissent de plus en plus précises et claires. Il arrive que certains se sentent perdus au départ comme devant un mur lisse sur lequel ils n'ont aucune prise, peut-être un peu la même impression que dans un pays étranger en entendant une langue que l'on ne maîtrise pas bien et que l'on s'efforce de comprendre.

Ces phénomènes s'expliquent probablement par l'absence initiale de référent à la pratique d'un langage de programmation. Tout se passe comme si le débutant était totalement étranger à l'espace de pensée ouvert par le langage de programmation. Pour lui le langage ne se réfère à rien et du coup il ne voit pas comment le « parler », il n'arrive pas à le relier avec du sens, avec ce qu'il souhaite faire. De ce fait l'apprentissage n'est pas frontal, direct, facile à planifier. Il ne s'agit pas pour l'apprenant d'apprendre des solutions, il s'agit pour lui d'apprendre à les trouver par lui-même. Pour l'apprenant, qui se trouve obligé d'entrer dans une nouvelle activité de penser, il est question d'une véritable conquête. Avec le temps, des expérimentations et de l'entraînement, des pôles de clarté, de compréhension et de savoir-faire s'affirment et se relient doucement entre eux.

5 Conclusion

La programmation informatique passe par des langages qui répondent à différentes sortes de grammaires (syntaxes) à partir desquelles nous pouvons élaborer du sens et exprimer des points de vue sur toutes sortes de sujets. La programmation, matière scientifique et technique, n'apparaît pas seulement sous le prisme de langages formels et scientifiques, elle apparaît également comme une écriture qui permet de placer l'imaginaire et la sensibilité émotionnelle au cœur de l'élaboration de multiples réalisations techniques. Cette écriture qui n'a rien de mécanique permet de s'exprimer et de produire des récits sous forme de scénarios d'actions algorithmiques aux prises avec l'imaginaire plus général des actions possibles. Elle peut être abordée de multiples façons. Elle peut notamment être appréhendée de façon littéraire, artistique et philosophique du point de vue de la maîtrise technique comme du sens parce qu'elle se travaille comme un art, comme un instrument de musique par exemple. Sa plasticité conceptuelle est considérable et elle autorise de nombreuses et originales associations entre arts, philosophie, sciences, technologie.

Références

Chazelle, B. (2013). *L'algorithmique et les sciences.* Fayard et Collège de France.

Csikszentmihalyi, M. (2005). *La créativité, psychologie de la découverte et de l'invention*. Robert Laffont.

Drouillon, F. (2014). *Du C au C++, de la programmation procédurale à l'objet.* ENI.

Drouillon, F. (2003). *Chimères et gargouilles informatiques.* Thèse de doctorat Paris 8, diffusion ANRT.

Goody, J. (1979). *La raison graphique.* Les éditions de minuit.

Graham, P. (2004). *Hackers & painters, big ideas from the computer age*. O'Reilly.

Chandra, V. (2013). *Geek sublime*, *Une vision esthétique, littéraire, mathématique et pleine d'autodérision du codage*. Robert Laffont.

Lartigaud, D. (dir.) (2011). *Art++*. Éditions HYX.