Étude exploratoire des conditions supportant l'engagement dans l'activité de programmation informatique

Mémoire

Alexandre Lepage

Maîtrise en technologie éducative - avec mémoire
Maître ès arts (M.A.)

Québec, Canada

Étude exploratoire des conditions supportant l'engagement dans l'activité de programmation informatique

Mémoire

Alexandre Lepage

Sous la direction de :

Lucie DeBlois, directrice de recherche

Margarida Romero, codirectrice de recherche

Résumé

De plus en plus de pays introduisent ou réintroduisent la programmation informatique dans les programmes de formation de la maternelle jusqu'à l'école secondaire. Dans ce contexte, ce projet de recherche vise à explorer les conditions qui supportent l'engagement lors de la pratique de la programmation informatique envisagée comme activité humaine. L'engagement est ici défini suivant à la fois des définitions issues des sciences de l'éducation et de l'informatique. Il se décline en trois types : comportemental, cognitif et affectif. Dix-huit (18) sujets ayant un minimum de cinq (5) ans d'expérience en programmation informatique ont accepté de participer à des entretiens individuels semi-dirigés au cours desquels ils étaient invités à discuter de la façon par laquelle ils ont appris à programmer, des contextes dans lesquels ils se sont engagés et de leurs perceptions de l'implication cognitive et affective en jeu. Les discours ont été codifiés à partir d'une démarche de recherche interprétative émergente afin d'éclairer la question de recherche suivante : Quelles sont les conditions supportant l'engagement dans l'activité de programmation informatique ? Les résultats nous amènent à proposer trois conditions supportant l'engagement : (1) les programmeurs et programmeuses développent un rapport au savoir dans lequel ils regardent un même problème depuis quatre postures épistémologiques en tension, (2) il y a convergence entre les motivations personnelles et organisationnelles, et (3) ils ont assez d'espace pour exprimer leur créativité.

Abstract

Computer programming is being introduced or reintroduced in K-12 around the world. In this context, this project aims at exploring what supports engagement in computer programming considered as a human activity. Engagement is here defined following both an educational definition of school engagement and a computer science definition of engagement, and it is divided into three types: behavioral, cognitive and affective. Eighteen (18) subjects each with at least five years of cumulated programming experience accepted to take part in semi-directed individual interviews. They were invited to discuss the way they learnt programming, the various contexts in which they engage themselves regarding programming, and their perception about cognitive and affective involvement. Parts of speech were then classified following an emergent research design to answer the following question: What conditions support engagement in computer programming? Results lead us to propose three conditions that support engagement: (1) programmers adopt a relationship to knowledge in which they must undertake different epistemological positions to develop an effective understanding of a problem, (2) there is a convergence between self and organizational motivations, and (3) they have enough space to express their creativity.

Table des matières

Résumé iii
Abstract iv
Table des matières v
Liste des figures vii
Remerciements ix
Introduction 1
1 Problématique 4
1.1 L'activité de programmation informatique 4
1.2 L'engagement dans la programmation informatique 6
2 Cadre théorique 10
2.1 La pensée informatique 10
2.2 Le rapport au savoir comme composante de la pensée informatique : des manifestations d'un engagement comportemental 12
2.3 L'abstraction comme composante de la pensée informatique : des manifestations d'un engagement cognitif 13
2.4 La marge créative comme composante de la pensée informatique : des manifestations d'un engagement affectif 18
2.5 L'apprentissage autostructuré ou hétérostructuré de la programmation informatique 20
2.6 Objectif de recherche et question de recherche 22
3 Méthode 23
3.1 Approche 23
3.2 Sujets et recrutement 23
3.3 Déroulement des entretiens 24
3.4 Le contenu des entretiens 25
3.5 La posture épistémologique du chercheur 26
3.6 Méthode d'analyse 26
3.7 Description des codes établis pour l'analyse 28
4 Résultats 32
4.1 Analyse individuelle du sujet 01 32
4.2 Analyse individuelle du sujet 02 36
4.3 Analyse individuelle du sujet 03 40
4.4 Analyse individuelle du sujet 04 44
4.5 Analyse individuelle de la sujet 05 48
4.6 Analyse individuelle du sujet 06 51
4.7 Analyse individuelle du sujet 07 56
4.8 Analyse individuelle de la sujet 08 60
4.9 Analyse individuelle du sujet 09 64
4.10 Analyse individuelle du sujet 10 68
4.11 Analyse individuelle du sujet 11 70
4.12 Analyse individuelle du sujet 12 74
4.13 Analyse individuelle du sujet 13 78
4.14 Analyse individuelle du sujet 14 81
4.15 Analyse individuelle du sujet 15 84
4.16 Analyse individuelle du sujet 16 87
4.17 Analyse individuelle du sujet 17 91
4.18 Analyse individuelle du sujet 18 94
5 Discussion 97

5.1 Réponse à la sous-question de recherche portant sur les manifestations de l'engagement comportemental 97
5.2 Réponse à la sous-question de recherche portant sur les manifestations de l'engagement cognitif 103
5.3 Réponse à la sous-question de recherche portant sur les manifestations de l'engagement affectif 107
5.4 Synthèse de la discussion 112
6 Conclusion 119
6.1 Rappel de la problématique et de la méthode 119
6.2 Rappel des résultats 119
6.3 Limites de la présente étude 120
6.4 Perspectives futures 121
7 Bibliographie 122
Annexes 126
Annexe 1 : Guide d'entretien 126
Annexe 2 : Courriel de recrutement 130
Annexe 3 : Formulaire de données complémentaires 131
Annexe 4 : Glossaire non exhaustif des termes informatiques utilisés 132

Liste des figures

Figure 1. Notre définition de la programmation informatique 6
Figure 2. L'engagement en programmation informatique 9
Figure 3. Trois axes pour décrire la pratique de la programmation informatique 101
Figure 4. La marge créative en programmation informatique 106
Figure 5. La programmation informatique comme activité de médiation ontologique 116

L'habitude est tendance à une fin sans volonté et sans conscience.

FÉLIX RAVAISSON

Remerciements

Merci à Margarida Romero qui a accepté de diriger mon travail de recherche dès nos premiers échanges. Je la remercie pour la confiance qu'elle m'a accordée. Je la remercie de m'avoir si habilement guidé vers la rigueur du travail scientifique, et surtout pour m'avoir amené à remettre en question de fausses certitudes que j'entretenais. Merci de m'avoir amené à un niveau de réflexion critique plus élevé qui m'a permis d'approcher ce travail de recherche avec beaucoup plus de nuances.

Merci à Lucie DeBlois qui a accepté de reprendre la direction de mon travail de recherche et dont la méthode de travail a su m'aider à mener à terme ce projet. Je la remercie pour l'efficacité et la précision de ses interventions qui ont su m'éviter bien des détours. Son apport au cadre théorique et à la méthode de la présente recherche est précieux et a contribué à ouvrir bien des avenues pour l'interprétation des résultats.

Merci aux 19 personnes qui se sont portées volontaires pour participer à cette étude et dont les réflexions ont permis de construire une solide base empirique. Je les remercie d'avoir accepté de réfléchir sur le vif à mes nombreuses relances dont les liens avec la programmation informatique n'étaient pas toujours évidents pour quiconque n'est pas mis au courant du cadre théorique.

Merci à mon conjoint, Alexandre Marois, pour ses encouragements depuis le début alors même que ce projet de maîtrise n'était qu'une idée. Je le remercie pour le regard critique qu'il a porté sur certaines esquisses. Merci d'avoir relu les 8 000 versions de ma problématique, chaque fois avec la même patience. Les discussions que nous avons eues m'ont permis de confronter mes idées et d'enrichir considérablement l'interprétation des résultats.

Merci finalement à mes collègues et amis qui ont formé autour de moi un environnement propice à la réalisation d'un travail sérieux, rigoureux. Leur présence a été déterminante dans la poursuite de ce projet de recherche. Merci notamment à Benjamin Lille et Raoul Kamga, notre système de pointage a été fort utile à la poursuite de mon parcours. Merci à mes parents qui ont toujours valorisé mes nombreux projets.

Introduction

Depuis quelques années, un mouvement est en marche pour intégrer la programmation informatique aux activités scolaires, parfois à l'échelle des pays, d'autres fois sous forme d'initiatives locales. En 2015, seize pays européens intégraient l'apprentissage de la programmation informatique à leur programme de formation à différents niveaux, de façon optionnelle ou obligatoire (Balanskat et Engelhardt, 2015). Au Canada, le gouvernement fédéral annonçait en juin 2017 un projet de 50 millions de dollars visant à donner aux jeunes « l'occasion d'apprendre le codage et de développer d'autres compétences numériques » (Gouvernement du Canada, 2017, para. 1). L'éducation étant de compétence provinciale, cette somme sert à financer des organismes ou organisations offrant des activités parascolaires. Au Québec, le ministre de l'Éducation, du Loisir et du Sport a annoncé à plusieurs reprises son intention de développer l'usage pédagogique de la programmation informatique dans le réseau de l'éducation (Cloutier, 2018; Proulx, 2018). Par exemple, il annonçait en janvier 2018 la mise sur pied du projet pilote Robot 360 visant à amener des enseignants et enseignantes à utiliser la programmation informatique et la robotique comme outils pédagogiques (Lavoie, 2018). De façon similaire, l'Institut de recherche Brookfield et le ministère de l'Éducation de l'Ontario annonçaient conjointement en avril 2018 un projet pilote de développement de la littératie numérique, lequel passera par l'apprentissage de la programmation informatique chez huit cohortes de 30 élèves âgés entre 12 et 15 ans (Brookfield Institute, 2018). Fin mai 2018, le ministère de l'Éducation et de l'Enseignement supérieur du Québec dévoilait le Plan d'action numérique en éducation et en enseignement supérieur dont l'une des mesures vise à « accroître l'usage pédagogique de la programmation informatique » (Ministère de l'Éducation et de l'enseignement supérieur, 2018, p. 27).

Dans ce mouvement pour intégrer la programmation informatique à l'école, les expressions *programmation informatique* et *codage* sont parfois employées indistinctement. Or, en sciences informatiques, le codage correspond à la représentation de données sous différentes formes et non à l'activité de programmation informatique. Il « consiste à transiter par une "table de conversion" qui permet de passer (mapper) d'une représentation du symbole à coder à une autre représentation plus fiable, plus sécurisée, plus facile à transmettre » (Goupille, 2015, p. 51). En anglais, le terme *coding* peut désigner l'écriture de programmes informatiques, ce qui peut expliquer l'appropriation du terme *codage*, traduction approximative de *coding*, dans des domaines éloignés de l'informatique pour désigner la programmation informatique. Il sera ici question de programmation informatique et non de codage.

Les milieux éducatifs qui choisissent d'intégrer la programmation informatique font face à de nouveaux défis pédagogiques comme la formation des enseignants et enseignantes, la préparation des élèves, l'usage de la programmation dans le respect des programmes scolaires. Ce projet de recherche vise à éclairer les conditions

pouvant supporter l'engagement des apprenants et apprenantes dans les activités de programmation informatique à l'école. Il s'inscrit dans une volonté de mieux comprendre l'engagement nécessaire à la pratique de la programmation informatique afin d'aider à mettre en place des conditions favorisant la réalisation de son potentiel pédagogique. Avant de présenter ce projet de recherche, il convient toutefois d'apporter quelques précisions sur la rédaction du présent document.

Ce mémoire a été rédigé de façon épicène. Plusieurs options ont été considérées, notamment l'emploi de doublets abrégés. Toutefois, nous ne parvenions pas à nous habituer à la graphie *programmeur.euse.s* sur la simple base que ce terme est imprononçable et qu'il implique une opération intellectuelle supplémentaire d'interprétation au moment de la lecture. À défaut de trouver un compromis parfait, nous avons opté pour l'utilisation de doublets traditionnels, par exemple en formulant *programmeurs et programmeuses*, couplée à une utilisation optimale de termes épicènes ou ne référant pas à la personne, lorsque possible. La seule exception est la section regroupant les analyses individuelles où l'emploi du masculin générique a été préservé afin de rendre compte des idées des personnes telles qu'elles ont été exprimées (tous et toutes s'étant exprimé dans un masculin générique). Partout dans le texte, le lecteur ou la lectrice notera que nous avons préféré le terme *sujet* au terme *participant*, le premier étant épicène.

Dans les analyses individuelles des deux femmes qui ont participé à l'étude, nous avons pris la liberté de féminiser le terme *sujet* en lui adjoignant des articles féminins, à défaut de quoi l'alternance entre le masculin de l'analyse et le féminin des propos rapportés serait apparue incohérente. Le problème de l'emploi d'un doublet traditionnel rédigé *participants et participantes* n'était pas tant la lourdeur mais plutôt les contorsions sémantiques que cela aurait impliquées, par exemple dans la phrase suivante : « Des différences sont observées entre les participants et participantes » (ici, l'emploi du doublet traditionnel peut suggérer qu'il y a des différences entre les hommes et les femmes, alors qu'il est pourtant employé pour désigner le groupe de tous les participants et participantes sans égard à leur genre). La formule retenue, l'usage des doublets traditionnels, est imparfaite en raison de la lourdeur qu'elle ajoute au texte et nous en sommes conscient. Nous nous contenterons de nous prononcer de façon très personnelle en faveur d'une éventuelle réforme de la langue française, laquelle ne remplit plus sa fonction sémiotique en ce qui a trait à l'emploi systématique du genre. Ainsi, nous sommes aux prises, comme bien d'autres en cette époque, avec des idées que la langue ne nous permet pas d'exprimer sans distraire par de lourds et nombreux ajouts.

Ce mémoire a été rédigé en utilisant le *nous* de modestie. Ce choix a été fait par respect des conventions et afin de rappeler que ce travail, bien que rédigé par une personne, est le fruit de bien des apports directs ou indirects qu'il est honnête de reconnaître.

Afin d'établir le questionnement qui guidera ce projet de recherche, nous avons convenu de nous intéresser d'abord au concept de programmation informatique en juxtaposant des définitions reçues. Par la suite, nous nous positionnerons sur le concept d'engagement et la signification qu'il peut prendre en programmation informatique. Cela nous conduira à établir un cadre théorique, puis à arrêter une question de recherche afin de mettre en place une méthode. Suivront les chapitres concernant les analyses des données récoltées puis leur interprétation. La conclusion du mémoire fera la synthèse de cet exercice de recherche.

1 Problématique

1.1 L'activité de programmation informatique

Dans les années 1960, la programmation informatique était définie principalement par rapport aux sciences dans lesquelles elle s'inscrivait – notamment les sciences physiques. Par exemple, Hoare (1969) a défini la programmation informatique comme une science exacte, car il considère que les propriétés d'un programme informatique et toutes les suites de son exécution peuvent être déduites à partir du code qui le compose. Cette définition peut s'expliquer considérant que dans les années 1960, la recherche en informatique était orientée autour de l'optimisation des algorithmes, principalement sur la base de leur temps d'exécution (Hopcroft, 1987). La programmation informatique est toujours considérée par certains comme le regroupement d'instructions pouvant être interprétées ou compilées par un ordinateur (p. ex., Rochkind, 2004).

Toutefois, d'autres définitions tendent à s'intéresser au rôle de l'humain dans cette activité. Hoc, Green, Samurçay et Gilmore (1990) ont suggéré que la programmation informatique est plus qu'un regroupement d'instructions et qu'elle représente un moyen d'expression de la pensée. Weinberg (1998) ajoute une dimension interpersonnelle à la programmation informatique en la définissant comme une activité sociale dont l'accomplissement requiert des interactions avec d'autres personnes, par exemple pour discuter des options possibles afin de corriger un bogue. Romero (2018) la définit comme une « activité réflexive » (para. 8), laquelle « va au-delà de l'écriture d'instructions en langage informatique, car avant d'écrire du code, il faut analyser une situation, puis s'engager dans une démarche de conception » (para. 7).

C'est ainsi que certaines de ces définitions (Hoare, 1969; Hopcroft, 1987; Rochkind, 2004) tendent à considérer la programmation informatique du point de vue de l'ordinateur alors que d'autres (Hoc et al., 1990; Romero, 2018) la considèrent aussi du point de vue de la personne qui pratique cette activité. À ce sujet, Green (1990) conclut un chapitre intitulé *The Nature of Programming* en affirmant que :

> Nous envisageons encore les programmes [informatiques] comme étant simplement destinés à la compilation [par un ordinateur]. Nous devrions les envisager aussi comme moyen de communication entre nous et les autres, et comme véhicule pour exprimer nos pensées à nous-mêmes. Ainsi, nous devrions considérer davantage la différence entre la lecture et l'écriture [de programmes informatiques], entre la capture des idées et l'affichage des idées. (p. 42, traduction libre)

Cela amène à considérer que les programmes informatiques peuvent être intelligibles par l'un ou l'autre de l'être humain ou de l'ordinateur, et parfois par les deux. En ce sens, Paloque-Bergès (2009) dit, au sujet du code informatique :

> En contexte informatique, le code est la première articulation conceptuelle du phénomène numérique : le niveau élémentaire est le code binaire, dont les deux signes, 0 et 1, convertissent les signaux électriques des voltages en signes électroniques et numériques. Le *bit* est l'unité de mesure désignant la quantité élémentaire d'information représentée en termes binaires. La programmation se passe à plusieurs niveaux. (p. 11)

Au sujet de ces différents niveaux de programmation, elle ajoute que le langage machine, de bas niveau, est « un ensemble d'instructions et de données qui sont directement compréhensibles par l'unité centrale de l'ordinateur » (p. 11). Selon elle, le code dit *code-source*, de haut niveau, est « écrit pour être déchiffré par la machine et par l'homme » (p. 11). La coexistence de ces différents niveaux de programmation, certains étant destinés à la compréhension par la machine et d'autres à une compréhension mutuelle entre la machine et l'être humain, peut expliquer pourquoi certaines définitions de la programmation informatique s'intéressent aux implications cognitives qu'elles peuvent avoir chez une personne alors que d'autres ne s'y intéressent pas.

Mendelsohn, Green et Brna (1990) considèrent qu'il y a, à l'égard du développement cognitif et de la programmation informatique, deux visions dominantes. L'une correspond à la programmation informatique comme moyen d'interagir avec le savoir d'une façon qui serait impossible autrement. L'autre correspond à la programmation comme un outil permettant de développer des habiletés cognitives pouvant être réutilisées dans d'autres domaines que l'informatique. La première vision prend son origine autour des travaux de Papert (1992) pour qui l'informatique, en plus de faciliter l'accès à la connaissance, modifie la nature de cette connaissance et le rapport qu'y développe l'humain. Papert donne pour exemple l'aspect dynamique de la connaissance lorsqu'un individu est en interaction avec un ordinateur. Aujourd'hui, la seconde vision, celle de la programmation informatique comme moyen de développer des habiletés cognitives, rassemble des publications s'appuyant sur un article de Wing (2006) qui considère la programmation informatique comme une activité des sciences informatiques parmi plusieurs, laquelle peut mener selon elle à la pensée informatique dont nous discuterons dans le cadre théorique. Cette vision de la programmation informatique s'apparente aussi à l'idée plus générale de Jonassen (2006) voulant que l'ordinateur soit un outil cognitif de modélisation : « Quand les apprenants utilisent les ordinateurs comme partenaires, ils lui délèguent certaines tâches non productives de mémorisation, ce qui leur permet de réfléchir de façon plus productive » (p. 21, traduction libre).

Pour Azevedo (2005), en plus d'être un outil cognitif, l'ordinateur serait aussi un outil métacognitif de régulation des apprentissages si son usage respecte les six caractéristiques suivantes : (1) les apprenants et apprenantes prennent des décisions quant aux objectifs d'apprentissage, (2) les apprenants et apprenantes prennent des décisions quant au contexte dans lequel il est utilisé, (3) l'ordinateur supporte l'autorégulation en amenant l'apprenant ou l'apprenante à se questionner, (4) l'ordinateur amène l'apprenant ou l'apprenante à utiliser des habiletés spécifiques à un domaine, (5) l'ordinateur est utilisé dans un contexte où les pairs, les enseignants, enseignantes et autres personnes agissent comme des agents de régulation, et (6) l'usage de processus

métacognitifs est essentiel pour réaliser un apprentissage. Suivant cette idée, la programmation informatique apparaît comme une activité rendue possible grâce à l'ordinateur, et son potentiel métacognitif dépend de ces caractéristiques.

Aux fins de la présente recherche et à partir des définitions reçues, nous concevons la programmation informatique comme une activité réflexive où l'engagement de la personne amène à la création de programmes informatiques originaux lesquels peuvent être envisagés à partir de deux angles : comme un rassemblement d'instructions destinées à la compilation ou à l'interprétation par un ordinateur, ou bien comme activité humaine de construction. La Figure 1 illustre cette définition. Envisagée comme activité, la programmation informatique est orientée vers la construction d'un programme informatique, une création numérique. Si la programmation informatique n'est pas considérée en tant qu'activité humaine, alors le programme informatique est envisagé seulement comme un regroupement d'instructions.

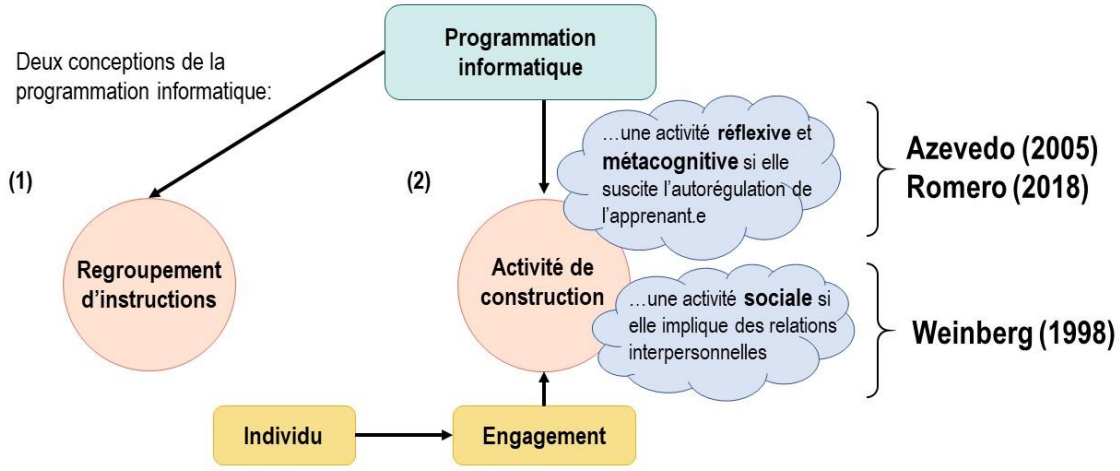

Figure 1. Notre définition de la programmation informatique

1.2 L'engagement dans la programmation informatique

La définition de la programmation informatique proposée par Romero (2018) fait intervenir le concept d'engagement, lequel est d'intérêt dans l'optique d'une intégration de la programmation informatique dans les activités scolaires. Pour Sheard, Carbone et Hurst (2010), le concept d'engagement réfère à la participation et l'implication d'une personne dans une activité d'apprentissage. Parent (2014) observe que le concept

d'engagement est souvent relié à celui de motivation, mais soutient que ces deux concepts doivent être distingués. L'engagement est différent de la motivation, car il « implique un plus grand ancrage dans l'action que la motivation » (p. 15). Les écrits convergent vers une classification de l'engagement en trois types : l'engagement comportemental, l'engagement affectif et l'engagement cognitif (Sheard, Carbone et Hurst, 2010). Pour Fredricks, Blumenfeld et Paris (2004), chercheurs et chercheuses qui ont distingué ces trois types d'engagement à l'origine, l'engagement comportemental s'observe principalement à partir du respect des règles et des normes de la classe, l'engagement affectif s'observe par les attitudes manifestées à l'égard de l'école, et, finalement, l'engagement cognitif s'observe par le recours à des stratégies personnelles d'autorégulation. Comme cette classification sert à comprendre l'engagement en contexte scolaire, il est possible qu'elle ne soit pas adaptée à l'étude de la programmation informatique au-delà des contextes scolaires.

Certaines études abordent l'engagement dans l'activité de programmation informatique, parfois sans en donner de définition, ce qui peut rendre leur comparaison ardue. Il ressort tout de même que l'engagement en programmation informatique peut varier d'une personne à l'autre ou d'un groupe à l'autre. Par exemple, Papavlasopoulou (2016) mène des travaux pour accroître l'engagement des filles dans des activités de programmation. À cet effet, elle relate le faible nombre de filles qui choisissent des carrières en lien avec les sciences de l'informatique. Kelleher (2009) s'est aussi intéressée au phénomène de l'engagement dans l'activité de programmation et a identifié des barrières qui peuvent expliquer que certaines personnes s'engagent moins que d'autres dans ces activités. Entre autres, ces barrières peuvent se manifester par le manque d'intérêt personnel pour la programmation informatique, l'échec à développer un sentiment de contrôle, ou l'incapacité à déterminer un outil de programmation approprié à la réalisation d'un objectif (Kelleher, 2009).

D'autres études se sont intéressées à l'engagement en programmation informatique auprès de programmeurs débutants. Par exemple, Bosch, D'Mello et Mills (2013) ont relevé que les premiers contacts avec la programmation informatique peuvent être vécus différemment entre les personnes : alors que certaines rapportent ressentir de l'engagement, d'autres ressentent de la confusion, de l'ennui ou de la frustration. Ces observations sur l'engagement en programmation informatique sont toutefois difficiles à juxtaposer étant donné que l'engagement est parfois entendu comme une émotion (Bosch et al., 2013), parfois comme un comportement observable (Kelleher, 2009). Ces deux visions peuvent s'approcher de deux des trois types d'engagement scolaire que nous avons évoqués, soient respectivement l'engagement affectif et l'engagement comportemental. Le rapprochement est toutefois imparfait considérant que les comportements observables ne peuvent être de l'ordre du respect des règles de la classe si l'activité de programmation informatique n'a pas lieu dans un contexte scolaire.

Naps et al. (2003) ont proposé une taxonomie de l'engagement dans des situations d'apprentissage impliquant l'utilisation d'outils de visualisation en sciences informatiques, par exemple pour la création ou la représentation d'algorithmes. Les outils de visualisation ont émergé à la fin des années 1980 et sont des logiciels permettant de créer des représentations graphiques de concepts de sciences informatiques (Naps et al., 2003). Ainsi, parmi d'autres usages, ils sont souvent utilisés pour accompagner l'apprentissage de la programmation informatique. Naps et ses collaborateurs défendent l'idée que ces outils ont peu de valeur pédagogique s'ils n'engagent pas l'apprenant dans une démarche d'apprentissage actif. Afin de fournir un cadre théorique pour étudier l'engagement de l'apprenant dans l'utilisation de ces outils, leur taxonomie comporte six catégories[1] : aucune visualisation, visualisation, interaction, modification, construction et présentation. Les auteurs insistent sur l'idée que ces catégories, excluant la première, bien qu'elles puissent témoigner d'une progression dans les niveaux d'engagement, peuvent se superposer. Ainsi, un apprenant peut utiliser un outil pour visualiser un algorithme tout en y apportant des modifications. Cette taxonomie a été largement reprise dans de nombreuses études s'intéressant à l'apprentissage des sciences informatiques. Par exemple, Yuan et al. (2010) l'ont appliquée à l'apprentissage de concepts de sécurité informatique, Fouh et al. (2014) l'ont utilisée pour concevoir des livres numériques interactifs portant sur l'algorithmique, et Kasurinen, Purmonen et Nikula (2008) l'ont employée pour concevoir un cours d'introduction à la programmation informatique. Comme cette taxonomie a été conçue spécifiquement pour étudier l'engagement dans l'utilisation d'outils de visualisation, la question de son applicabilité à l'ensemble de l'activité de programmation informatique demeure ouverte.

Nous considérerons l'engagement en programmation informatique comme un ensemble de comportements (engagement comportemental), de stratégies cognitives (engagement cognitif) et d'attitudes (engagement affectif) soutenus par les postures épistémologiques adoptées. Un tel emprunt au concept d'engagement scolaire implique toutefois de reconnaître que certaines manifestations de l'engagement ne peuvent pas s'observer dans des contextes qui ne sont pas scolaires (par exemple, le respect des règles de la classe comme manifestation d'engagement ne s'applique qu'à un contexte scolaire). La Figure 2 illustre notre définition de l'engagement aux fins de la présente recherche.

[1] Nous avons traduit nous-même les six catégories de la taxonomie de Naps et al. (2003). En anglais, elles sont respectivement *no viewing*, *viewing*, *responding*, *changing*, *constructing* et *presenting* (Naps et al., 2003, p. 142).

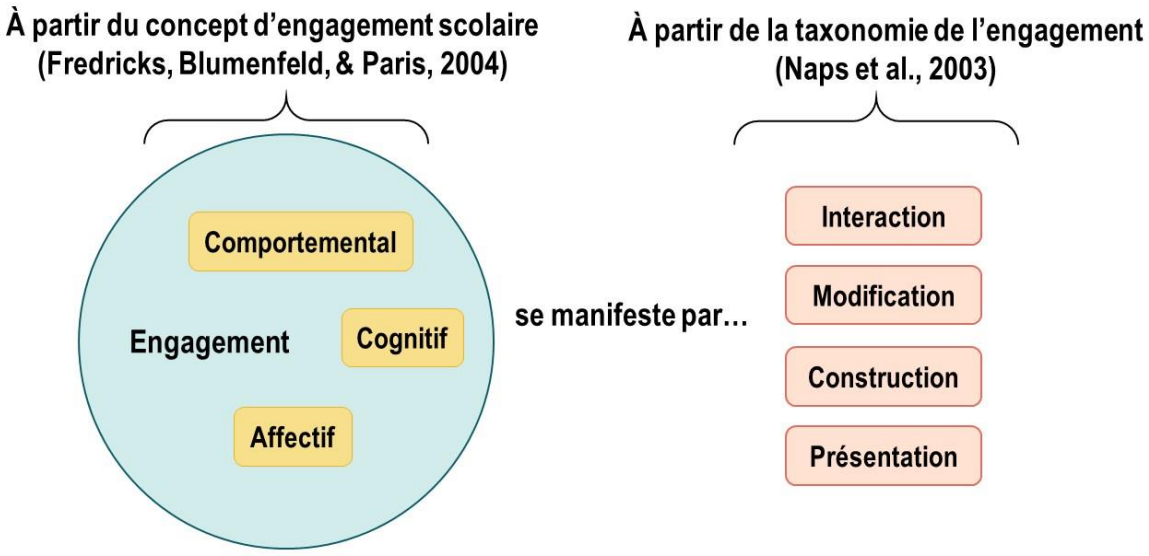

Figure 2. L'engagement en programmation informatique

Suivant notre définition de la programmation informatique, nous envisageons que cet engagement doit être accompagné de la création de programmes informatiques qui respectent deux caractéristiques : (1) ils sont originaux en ce qu'ils ne sont pas exclusivement reproduits d'autres programmes informatiques et (2) ils sont considérés comme des créations numériques et sont imprégnées de la personne ou du groupe qui les a créés. À partir de la taxonomie de l'engagement de Naps et al. (2003), nous considérons ici que l'engagement peut s'observer à partir de ce qu'une personne fait avec un programme informatique : Répond-elle à des questions sur son fonctionnement (interaction) ? Y apporte-t-elle des changements (modification) ? Le construit-elle (construction) ? Le présente-t-elle à d'autres personnes (présentation) ?

En vue d'une intégration de la programmation aux activités scolaires, il apparaît utile de comprendre ce qui supporte l'engagement dans la programmation informatique. Toutefois, pour ce faire, il nous faut circonscrire le cadre théorique qui permettra de repérer des manifestations d'engagement comportemental, cognitif et affectif. Une telle recherche permettra d'ouvrir de nouvelles perspectives pour éclairer le mouvement d'intégration de la programmation informatique à l'école primaire et secondaire, et d'outiller les acteurs des systèmes éducatifs.

2 Cadre théorique

2.1 La pensée informatique

Le concept de pensée informatique a été proposé par Wing (2006) comme un mode de raisonnement présent dans toutes les sphères de la vie et qui n'est pas spécifique aux sciences informatiques. Wing (2008) propose qu'au cœur de la pensée informatique se trouve le concept d'abstraction qui consiste, selon elle, à décider quels sont les détails importants à la résolution d'un problème et quels sont ceux qui peuvent être ignorés.

Papert (1980) a utilisé l'expression « computational thinking » (p. 182) dans son ouvrage *Mindstorms*, mais n'en a pas proposé de définition formelle : « Their visions of how to integrate computational thinking in everyday life was insufficiently developed » (p. 182) ». Barba (2016) considère que le concept de pensée informatique tel que présenté par Wing et abondamment repris ne correspond pas à la vision qu'en avait Papert lorsqu'elle affirme que la pensée informatique consiste à réfléchir comme un ingénieur informatique. En effet, Barba fait remarquer que Papert s'intéressait davantage aux possibilités offertes par le média qu'est l'ordinateur en termes de création de connaissances, plutôt qu'à la résolution de problèmes. Ce constat fait intervenir la question du rapport au savoir non seulement en programmation informatique, mais aussi de façon plus générale dans tous les usages de l'informatique.

Bien que Wing (2008) propose que l'abstraction soit au cœur de la pensée informatique, il demeure que celle-ci peut impliquer à la fois des activités abstraites et des activités concrètes. À cet effet, Papert s'est positionné à plusieurs reprises contre l'antagonisme entre l'abstrait et le concret (Papert, 1980, 1992; Turkle et Papert, 1990). Pour Papert (1980), une partie de ce qui est considéré comme abstrait en informatique est attribuable à la culture environnante. Il donne l'exemple des boucles en programmation : pour lui, cela n'a plus rien d'abstrait dès lors que l'on y est exposé. Cette idée se rapproche de Langevin (1934) pour qui « le concret, c'est de l'abstrait rendu familier par l'usage » (p. 45). Turkle et Papert (1990) plaident en faveur d'une revalorisation des activités concrètes qui selon eux sont tout aussi mobilisées que des activités dites abstraites dans l'usage de l'ordinateur.

Bundy (2007) écrivait au sujet de la pensée informatique qu'elle change notre façon de penser :

> Bien sûr, de nos jours nous avons tous des ordinateurs sur nos bureaux. Nous les utilisons pour les courriels, pour la navigation Web, le traitement de texte, les jeux vidéos, etc. Mais la révolution de la pensée informatique va beaucoup plus loin que cela; elle change la façon par laquelle nous pensons. (p. 1, traduction libre)

Cet extrait est cohérent avec la façon par laquelle Papert envisageait la pensée informatique, c'est-à-dire comme une transformation du rapport au savoir. Bundy (2007) donne des exemples d'usages de l'informatique ayant modifié la façon de penser de chercheurs :

> Les modèles de calcul à haute performance ont, pour la première fois, permis à des géologues de rendre compte de la complexité [de la géologie] et d'obtenir une meilleure compréhension géologique. (Bundy, 2007, p. 1, traduction libre)

Il apparaît ici qu'un tel usage de l'informatique implique un rapport au savoir où les personnes se sont donné ou ont compris avoir la permission de créer. Elles ont ainsi modifié ou adapté leur posture épistémologique. Si un tel usage de l'informatique est rendu possible par une attitude de création, alors cette attitude pourrait être posée comme une des conditions pouvant supporter l'engagement. Les exemples de Bundy nous amènent toutefois à remettre en question la modification du fonctionnement de la pensée qu'il évoque : si la pensée, au sens de la cognition, est envisagée comme un ensemble de processus adaptatifs, alors il n'est pas clair si l'informatique en induit de nouveaux ou si elle amène simplement à l'application de processus connus à des situations inédites.

Voogt, Fisser, Good, Mishra et Yadav (2015) ont relevé qu'il existe des tensions quant aux limites du concept de pensée informatique. Ils soulignent que d'un côté, certains le réduisent à la programmation informatique, alors qu'à l'autre bout du spectre, certains y intègrent des attitudes comme la tolérance à l'ambiguïté et la propension à collaborer. Pour Romero, Lepage et Lille (2017), la pensée informatique correspond à un ensemble de stratégies cognitives et métacognitives mises au service de l'identification, de la représentation, de la programmation et de l'évaluation de systèmes complexes. Selby et Woollard (2013) ont analysé plusieurs définitions reçues de la pensée informatique et ont identifié des composantes vers lesquelles il y a convergence : l'abstraction, la décomposition, l'évaluation, la généralisation et la pensée algorithmique. Pour certains de ces éléments dont l'abstraction, ils identifient un consensus parmi les définitions qu'ils ont analysées.

Ceci nous amène à trois prémisses : (1) certains écrits suggèrent que la pratique de la programmation informatique peut induire ou nécessiter une transformation du rapport au savoir, (2) il y a convergence autour de l'idée que l'abstraction soit parmi les composantes centrales de la pensée informatique, et (3) le potentiel pédagogique de la programmation informatique est accru si elle est pratiquée dans des situations laissant une marge créative à l'individu. Nous entendons cette marge créative comme la latitude que se donne une personne pour créer lorsqu'elle travaille sur un problème de programmation informatique. Cette marge peut être décrite sur la base des caractéristiques du problème; un problème ouvert et mal défini donnant une marge créative plus grande. Les trois prochaines sections visent à élaborer ces prémisses en lien avec, respectivement, l'engagement comportemental, l'engagement cognitif et l'engagement affectif.

2.2 Le rapport au savoir comme composante de la pensée informatique : des manifestations d'un engagement comportemental

Nous considérons ici que le rapport au savoir d'un individu peut aider à caractériser les situations dans lesquelles il ou elle pratique la programmation (engagement comportemental). L'origine et la conception des savoirs sont ici abordées via la discipline de l'épistémologie. Dans le domaine de l'éducation, Fourez (2004) parle de l'épistémologie comme de « la discipline qui étudie la façon dont on connaît » (p. 9). Selon lui, l'épistémologie ne peut être dissociée des personnes; en ce sens, il affirme que « chacun a son itinéraire en épistémologie » (p. 10). Pour Fourez, l'épistémologie se positionne aussi comme l'étude de la construction sociale des sciences. À cet égard, le sens de la vérité et la valeur qui y est accordée sont centraux dans la définition de la posture épistémologique des individus.

En philosophie, l'épistémologie se manifeste par des courants abordant la connaissance humaine de différentes façons. Par exemple, Descartes (2000/1637) a élaboré ce qui est aujourd'hui considéré comme une posture rationaliste de la connaissance, c'est-à-dire que la seule connaissance véritable est celle issue de la raison au moyen d'une démarche minutieuse qu'il décrit dans son *Discours de la méthode*. Kant (1944/1781) établit une distinction entre les formes « a priori » (p. 32) et « a posteriori » (p. 32) de la connaissance, les premières étant issues de l'esprit et les secondes étant issues des sens, il dit ces dernières empiriques. On attribuera généralement à Auguste Compte la conceptualisation du positivisme, lequel propose l'existence de vérités exactes que les sciences peuvent découvrir au moyen de la méthode scientifique (Pickering, 2011).

Dans sa théorie de l'épistémologie génétique, Piaget (1970) aborde un point de vue évolutif en affirmant que :

> […] la connaissance ne saurait être conçue comme prédéterminée ni dans les structures internes du sujet, puisqu'elles résultent d'une construction effective et continue, ni dans les caractères préexistants de l'objet, puisqu'ils ne sont connus que grâce à la médiation nécessaire de ces structures et que celles-ci les enrichissent en les encadrant (ne serait-ce qu'en les situant dans l'ensemble des possibles). (p. 7)

Cette médiation implique que le sujet est actif dans la construction de ses connaissances. À cet égard, Piaget est considéré comme l'un des pionniers du courant épistémologique constructiviste en éducation. Ce courant a eu des répercussions en éducation notamment au niveau des pratiques pédagogiques, lesquelles sont de plus en plus orientées vers la construction des savoirs par les apprenants et apprenantes plutôt que par leur transmission par l'enseignant ou l'enseignante. Au Québec, cette vision se matérialise dans le Programme de formation de l'école québécoise dans le cadre du renouveau pédagogique de la fin des années 1990 et du début des années 2000.

De telles conceptions du savoir et de la façon par laquelle l'humain y accède ou le construit s'observent aussi dans l'activité de programmation informatique. En ce sens, Turkle et Papert (1990) proposent le concept de pluralisme épistémologique pour décrire les implications épistémologiques de la pratique de la programmation informatique :

> La diversité des approches en programmation informatique suggère que même les éléments les plus simples d'informatique impliquent d'accepter la validité de multiples façons de connaître et penser, un pluralisme épistémologique. (p. 129, traduction libre)

À plusieurs reprises, Papert (1992) considère que l'usage de l'informatique en général a des répercussions sur l'épistémologie notamment en raison de la nature des savoirs qui sont manipulés par un individu – ces savoirs seraient dynamiques plutôt que statiques.

Plusieurs auteurs prétendent que la pensée informatique est transversale à plusieurs domaines de connaissances (p. ex. Bundy, 2007; Voogt et al., 2015; Wing, 2006). Sans admettre ou contredire cette prétention, nous aborderons la posture épistémologique adoptée par des programmeurs et programmeuses en nous intéressant aux ontologies dans lesquelles s'inscrit la programmation informatique. À cet égard, Guarino (1998) relève que le terme *ontologie* peut avoir différentes significations : au sens philosophique, une ontologie est « un système particulier de catégorisation qui traduit une certaine vision du monde » (p. 4, traduction libre), alors qu'en informatique, elle est « une hiérarchie de concepts reliés par des relations d'inclusion » (p. 4, traduction libre). Étant donné que nous nous sommes positionnés pour considérer la programmation informatique en tant qu'activité humaine, nous entendrons l'ontologie au sens philosophique précité. Suivant l'idée de pluralisme épistémologique que nous avons discutée, nous considérons que plusieurs postures épistémologiques peuvent coexister chez une même personne, chacune traduisant différentes visions du monde.

2.3 L'abstraction comme composante de la pensée informatique : des manifestations d'un engagement cognitif

Nous proposons ici un cadre théorique de l'abstraction qui reprendra à la fois des théories psychologiques et didactiques. Nous aborderons le concept d'abstraction réfléchissante (Piaget, 1977) en tentant de l'adapter à l'activité de programmation informatique. Par la suite, nous présenterons la théorie des champs conceptuels puis le modèle d'interprétation des activités cognitives de l'élève. Ces deux ajouts offriront un cadre théorique pour opérationnaliser l'étude de l'abstraction dans l'activité de programmation informatique.

2.3.1 Le concept d'abstraction

Piaget (1977) distingue deux types d'abstractions : l'abstraction empirique et l'abstraction réfléchissante. L'abstraction empirique porte sur des objets physiques et ne porte pas sur les constructions intellectuelles des individus. La deuxième abstraction est dite réfléchissante. Contrairement à l'abstraction empirique, elle n'est pas réalisée sur les objets du monde physique, mais plutôt à partir des activités cognitives du sujet sur sa propre pensée.

L'abstraction empirique s'appuie sur les objets physiques plus que sur des constructions intellectuelles. DeBlois (1996) précise que Piaget (1977) « qualifie les abstractions d'empiriques lorsque l'enfant tire ses informations des propriétés des objets et de réfléchissantes lorsqu'il les extrait des coordinations de ses actions » (DeBlois, 1996, p. 78). L'abstraction réfléchissante exige le transfert d'une abstraction d'un palier donné vers un palier supérieur; elle est à tendance généralisatrice. Le déplacement d'une abstraction d'un palier vers un autre implique une sélection d'informations ou de caractéristiques et incidemment la perte d'informations ou de caractéristiques. Ce déplacement est nommé réfléchissement; il s'agit de la première étape du processus d'abstraction réfléchissante. Le réfléchissement risque ainsi de présenter une réduction de l'information. Par la suite, la construction ou la modification de ces représentations en des niveaux supérieurs se nomme *réflexion*. Ce sont les régulations, autant d'ajustements, qui permettront à l'individu d'ajuster ces représentations à des niveaux supérieurs. Pour mieux comprendre les relations conceptuelles qui émanent de ce processus d'abstraction réfléchissante, nous faisons appel à la théorie des champs conceptuels de Vergnaud (1990).

2.3.2 La construction de champs conceptuels par le concept de schème

Vergnaud (1990) a mobilisé le concept de schème dans sa théorie des champs conceptuels pour établir des relations entre différentes situations mathématiques. Bien qu'il construise et illustre la théorie à partir de la discipline mathématique, il la présente comme applicable à d'autres domaines pour autant que le travail d'élaboration des champs conceptuels soit fait. Selon Vergnaud, cette théorie a pour but de « fournir un cadre qui permette de comprendre les filiations et les ruptures entre les connaissances » (p. 135). L'étude de ces filiations se fait par l'identification d'un champ conceptuel autour d'un concept. Ce champ conceptuel se décline en trois composantes : (1) l'ensemble des situations de référence, (2) les formes langagières, non langagières et symboliques utilisées pour évoquer ou désigner le concept, et (3) les invariants opératoires.

L'ensemble des situations de référence correspond au répertoire dans lequel le concept peut ou devrait être mobilisé. Vergnaud (1990) donne l'exemple des structures additives qui peuvent intervenir dans les problèmes de réunion ou de complément d'ensemble, d'ajout ou de retrait d'éléments et de comparaison d'ensembles. Dans cet exemple, les formes langagières, non langagières ou symboliques sont les signes explicites (par

exemple les mots d'un énoncé) permettant de référer directement ou indirectement au concept d'addition arithmétique. Les invariants opératoires sont les moins facilement perceptibles car ils comportent une part d'implicite, contrairement aux deux composantes précédentes. Les invariants opératoires correspondent à des caractéristiques, ou à des propriétés, qui ne sont pas modifiées malgré les transformations du problème ou du concept. Par exemple, l'addition est commutative quelle que soit la grandeur des nombres en jeu. La recherche des invariants opératoires permet, selon la théorie des champs conceptuels, d'élaborer un concept en permettant l'émergence d'une compréhension abstraite chez un individu. Vergnaud (1990) regroupe dans les invariants opératoires ce qu'il nomme « concepts-en-acte » (p. 142) et « théorèmes-en-acte » (p. 142). Il s'agit de concepts ou théorèmes qui sont mobilisés de façon implicite et sans nécessairement qu'un individu soit capable de les expliciter. Cette distinction est d'intérêt pour l'apprentissage, car elle suggère qu'une compréhension abstraite passe par une forme de connaissances implicites, lesquels peuvent documenter ce processus de compréhension.

Il existe trois types d'invariants opératoires : les propositions, lesquelles peuvent être dites vraies ou fausses, les fonctions propositionnelles, lesquelles ne peuvent pas être dites vraies ou fausses, et les arguments. Les propositions correspondent aux théorèmes-en-acte évoqués précédemment. Ils se constatent dans l'action du sujet par l'application de règles fixes. Nous proposons ici l'exemple d'un élève qui compte à l'aide de ses doigts par bonds de cinq en alternant les mains une à une. Dans cet exemple, il existe au moins une proposition vraie appliquée par l'élève : une main comporte cinq doigts. Les fonctions propositionnelles ne peuvent à elles seules être dites vraies ou fausses, puisqu'elles dépendent d'arguments. Nous proposons un exemple de la fonction propositionnelle de relation « … plus grand que ... ». Seule, elle ne peut être dite vraie ou fausse. La véracité dépend des deux arguments qui sont fournis. La proposition « 2 plus grand que 4 » est vraie, alors que la proposition « 4 plus grand que 2 » est fausse. Les fonctions propositionnelles correspondent aux concepts-en-acte évoqués précédemment.

La théorie des champs conceptuels suggère donc que la compréhension abstraite d'un concept passe par l'exploration d'un champ conceptuel dans lequel se trouvent des invariants. Ces invariants sont présents peu importe les variables didactiques comme la grandeur des nombres ou le contexte des situations. Ces champs conceptuels alimentent l'organisation des problèmes à proposer aux élèves, ce qui amène le sujet à constituer des schèmes cognitifs tantôt adaptés à la situation, tantôt inadaptés. Un schème « n'est pas un stéréotype mais une fonction temporalisée à arguments, qui permet de générer des suites différentes d'actions et de prises d'information en fonction des valeurs des variables de situation » (Vergnaud, 1990, p. 142). Piaget précise que c'est par l'assimilation ou l'accommodation que l'individu en vient à adapter ses schèmes. La construction ou la modification de schèmes correspond à la réflexion, processus d'abstraction réfléchissante décrit par Piaget (1977).

2.3.3 Observer le phénomène de l'abstraction réfléchissante

Observer le phénomène de l'abstraction réfléchissante peut être difficile étant donné qu'il s'agit d'une activité intellectuelle qu'un individu n'a pas nécessairement la capacité de rendre explicite, même s'il la réalise effectivement. DeBlois (2000) a proposé un modèle d'interprétation des activités cognitives de l'élève ayant pour objectif de guider l'intervention orthopédagogique auprès d'élèves en difficulté d'apprentissage en mathématiques. Le modèle se situe dans une perspective constructiviste de l'apprentissage; en somme il considère que l'élève construit des représentations plutôt qu'il ne les reçoit. Ce modèle vise à cerner l'origine de ces difficultés d'apprentissage en développant une compréhension du sens des procédures utilisées par l'élève lors de la résolution de problèmes mathématiques. Selon DeBlois (2003), comprendre le raisonnement de l'élève permet de mieux intervenir :

> Les différentes composantes identifiées au cours des recherches précédentes permettent de poser l'hypothèse selon laquelle une interprétation des activités cognitives des élèves, au moment où les élèves résolvent un problème en classe, donne des outils supplémentaires pour intervenir. (p. 179)

L'utilisation de ce modèle fait intervenir l'interprétation de productions des élèves (Bélanger, DeBlois et Freiman, 2014; DeBlois, 2003). Ainsi, ce modèle favorise la tenue d'une discussion avec l'élève pour identifier le sens des procédures mises en place lors de la réalisation de sa production. L'interprétation de ces procédures permet d'identifier des coordinations entre des représentations exprimées (ou non) par l'élève. Dans tous les cas, l'élève régule les coordinations entre ses représentations. La structuration de ses connaissances s'opère par une prise de conscience de ces coordinations, laquelle peut être soit partielle (et par conséquent liée aux caractéristiques des situations plutôt qu'aux concepts), soit généralisatrice. La compréhension du sens des procédures privilégiées par l'élève en difficulté peut donc amener à détecter deux types de structurations des connaissances : des structurations partielles qui mènent à la résolution effective de problèmes dans certains cas et à des échecs dans d'autres, et les structurations généralisables qui permettent la résolution adaptable à un ensemble de situations. Pour Piaget (1974), la prise de conscience joue un rôle central dans les interactions entre sujet et objets, et c'est à partir d'elle que se construit la connaissance. En ce sens, DeBlois (2015) observe que la prise de conscience « semble nécessaire à une évolution des procédures des élèves » (p. 173).

Il ressort donc du modèle d'interprétation des activités cognitives de l'élève que le développement d'une compréhension abstraite passe par des régulations qui se manifestent par des coordinations opérées par l'élève lui-même. La compréhension de ce processus passe par l'analyse des productions des élèves et par le questionnement rétroactif chez ces élèves quant aux procédures privilégiées. La question se pose alors : le modèle d'interprétation des activités cognitives des élèves peut-il s'appliquer à la programmation informatique de la même façon qu'à l'apprentissage des mathématiques ?

2.3.3.1 Observation en programmation informatique

Les recherches en didactique des mathématiques que nous avons évoquées précédemment prennent toutes en exemple des concepts ou problèmes mathématiques. Par exemple, DeBlois (1992, 1994, 1997, 2000) a recours à des situations de compléments d'ensemble et de réunion[2] faisant intervenir le concept de numération positionnelle.

Lister (2011) propose une analyse qualitative de certaines réponses obtenues dans une étude auprès de programmeurs débutants. Dans cette analyse, Lister s'appuie sur une vision néo-piagétienne du développement de l'abstraction qu'il définit ainsi :

> Alors que les types d'abstraction sont généralement les mêmes dans la théorie classique de Piaget et dans la théorie néo-piagétienne, la principale différence de cette dernière est que les personnes, peu importe leur âge, sont présumées progresser dans les formes de raisonnement abstrait au fur et à mesure que leur expertise augmente dans un domaine spécifique de problème. (Lister, 2011, sec. 1.1, traduction libre)

Les recherches de Piaget (1970) proposaient que le développement de la pensée formelle était précédé de plusieurs stades : le stade sensori-moteur, où la connaissance est issue de l'expérience des sens, le stade préopératoire, le stade des opérations concrètes, et finalement le stade des opérations formelles, où la connaissance est construite à partir d'opérations intellectuelles réalisées sur des représentations de l'individu. La vision néo-piagétienne adoptée par Lister reprend ces stades de développement, mais considère que le passage de l'un vers l'autre n'est pas le résultat du développement avec l'âge. Il serait plutôt spécifique à chaque domaine. Ainsi, un individu pourrait raisonner au niveau préopératoire dans certains domaines et au niveau formel dans d'autres. À cet effet, Lister donne un exemple :

> Ainsi, une personne qui est novice dans un domaine (par exemple, le jeu d'échecs), peut adopter des formes de raisonnement moins formelles que celles qu'elle afficherait dans un domaine où elle est experte (par exemple, le calcul). (Lister, 2011, sec. 1.1, traduction libre)

Il faut toutefois considérer, dans ce dernier cas, comment est défini le niveau préopératoire et comment est décrit le niveau formel. Pour Piaget, le niveau préopératoire exige de poser des actions sur des objets alors que le niveau formel permet d'agir à partir d'hypothèses. Dans l'étude de Lister, il était demandé aux sujets d'expliquer verbalement le rôle d'un programme informatique de quelques lignes. Si les explications verbales

[2] Pour le lecteur ou la lectrice non familier, une situation de complément d'ensemble est un problème mathématique où l'élève doit déterminer un nombre permettant de compléter un ensemble. Voici un exemple : un panier contient 12 pommes, quelqu'un en prend 3, combien en reste-t-il? La situation de réunion implique de réunir deux ensembles, voici un exemple : Marie a 3 poires et Jules a 3 pommes, ils les mettent dans le même panier, combien y a-t-il de fruits dans le panier ?

sont considérées comme une réflexion sur des hypothèses, nous pourrions attribuer le stade formel à cette explication. À partir des explications reçues, Lister suggère que nonobstant leur âge, les programmeurs novices peuvent s'en remettre à des raisonnements qui s'appuient sur des régulations, plutôt que sur des hypothèses, ce qui pourrait évoquer le niveau préopératoire décrit par Piaget.

Afin de soulever une possible explication des observations de Lister (2011), il nous apparaît pertinent de reprendre un des exemples qu'il propose pour l'éclairer à partir de la théorie des champs conceptuels de Vergnaud (1990). L'exemple de Lister correspond à un court programme informatique où une variable est incrémentée[3] à chaque tour d'une boucle. Nous posons l'hypothèse selon laquelle la compréhension de ce problème implique des coordinations entre l'assignation ou la répétition. Par exemple, pour comprendre l'assignation renouvelée de la variable à chaque tour de boucle, il doit y avoir coordination entre le concept de répétition (à l'aide d'une boucle) et celui d'assignation de valeur à une variable Ainsi, pour Lister une explication sans régulation correspondrait à un stade formel.

Ces apports théoriques amènent à considérer que l'abstraction est bien présente dans l'activité de programmation informatique. À cet égard, elle peut être considérée comme une manifestation d'engagement cognitif. Nous avons précédemment abordé l'engagement comportemental sous l'angle du rapport au savoir.

L'abstraction se présente comme un processus qui a lieu dans toutes les situations où un individu est amené à construire ou reconstruire des connaissances. À ce titre, la modification du rapport au savoir d'un individu passerait aussi par une prise de conscience des coordinations qu'il réalise en lien avec sa posture épistémologique personnelle. Nous relierons donc l'engagement comportemental à l'engagement cognitif ainsi : l'engagement comportemental d'une personne, observable par son rapport au savoir, peut être compris en s'intéressant aux connaissances épistémologiques qu'elle a construites. Ces connaissances ne sont pas toujours de l'ordre de l'explicite, ainsi peuvent-elles parfois être qualifiées de concepts-en-acte ou de théorèmes-en-acte.

2.4 La marge créative comme composante de la pensée informatique : des manifestations d'un engagement affectif

Plusieurs auteurs parlent de programmation créative en évoquant l'importance de la créativité dans l'activité de programmation informatique (p. ex., Papavlasopoulou, 2016; Romero, Davidson, Cucinelli, Ouellet et Arthur, 2016; Romero, Lepage et Lille, 2017). Pour Romero et ses collaboratrices (2016), la programmation créative va

[3] Une incrémentation est l'augmentation par 1 de la valeur d'une variable de type entier.

plus loin que l'écriture de code informatique. En ce sens, l'apprentissage de la programmation créative est à distinguer de l'apprentissage guidé de la programmation, par exemple via la répétition d'une séquence d'étapes pour reproduire un programme. À cet égard, elles évoquent l'existence d'outils permettant de commencer l'apprentissage de la programmation dans une approche créative dès le début, mais soulignent que beaucoup de ressources privilégient les séquences d'apprentissage décontextualisées de la programmation, notamment des sites Internet qui accompagnent l'apprenant ou l'apprenante pas-à-pas.

La conceptualisation de la créativité revêt une importance particulière dans le domaine de l'intelligence artificielle afin de qualifier le niveau de créativité de ce qu'un ordinateur produit. À cet effet, Wiggins (2006) soutient que la valeur créative d'un artéfact numérique créé par une intelligence artificielle s'évalue à partir d'une observation subjective d'un projet et pas nécessairement à partir de la complexité qu'il arbore. Wiggins s'intéresse à la créativité informatique, qu'il définit ainsi :

> L'étude et le support [par des personnes], à partir de moyens et de méthodes informatiques, de comportements adoptés par des systèmes naturels ou artificiels qui seraient considérés comme créatifs s'ils étaient adoptés par des humains. (p. 2, traduction libre)

Cette définition de la créativité s'apparente à ce que Romero et Lille (2017) disent lorsqu'ils affirment que « la créativité vise la création d'une solution nouvelle, innovante, pertinente, de valeur et qui fait preuve de parcimonie et d'élégance vis-à-vis d'une situation-problème initiale » (p. 30-31), notamment lorsqu'il est question de parcimonie ou de la distinction entre la complexité et la créativité.

Romero et al. (2017) considèrent que la programmation créative engage l'apprenant dans un processus de conception et de développement d'un travail original qui utilise le code informatique. Dans ce contexte, la programmation informatique peut être mobilisée comme un outil de co-construction de connaissances (Romero et al., 2017). En plus d'explorer des concepts de programmation comme les séquences et les boucles, la programmation créative implique de les appliquer à un projet signifiant pour l'apprenant ou l'apprenante (Strawhacker et Bers, 2015). C'est sur ce dernier point que nous établissons un lien entre la créativité en programmation informatique et l'engagement affectif : la démarche créative en programmation fait intervenir le rapport de l'individu à sa création (Weinberg, 1998), son rapport à l'erreur et sa propension à l'égocentrisme[4] (Pea, Soloway et Spohrer, 1987), ainsi qu'un aspect ludique (Resnick, 2007). Pour Resnick (2007), la création de programmes informatiques peut s'inscrire dans un cycle itératif de conception impliquant l'imagination, la création, le jeu, le partage et la réflexion. Resnick (2014) a par la suite

[4] Dans leur taxonomie des bogues informatiques, Pea et al. (1987) définissent l'égocentrisme ainsi : « L'égocentrisme consiste en une valeur exagérée donnée à la perspective personnelle d'une personne. C'est un trait de la pensée des enfants très répandu notamment dans la représentation de l'espace. » (p. 11)

développé cette idée et proposé l'approche des 4P pour décrire l'apprentissage créatif : « Projects, Peers, Passion, and Play » (para. 1).

Considérant le potentiel pédagogique accru de la programmation informatique si elle est pratiquée dans une approche créative, nous proposons que la marge créative dont disposent les programmeurs et programmeuses puisse être un aspect à explorer afin de décrire l'engagement affectif. Dans la présente étude, l'engagement affectif sera abordé en s'intéressant à la place de l'erreur chez une personne ainsi qu'à la marge créative dont elle dispose lorsqu'elle crée un programme informatique.

2.5 L'apprentissage autostructuré ou hétérostructuré de la programmation informatique

Comme ce projet de recherche trouve sa pertinence sociale dans le mouvement d'intégration de la programmation informatique aux activités scolaires, il apparaît utile d'établir la différence entre un apprentissage scolaire ou un apprentissage extrascolaire, réalisé à l'extérieur de la classe. Pour en discuter, nous mobilisons ici les distinctions entre l'apprentissage autostructuré et l'apprentissage hétérostructuré.

Ces deux formes d'apprentissage ont été abordées par Prévost (1994) qui distingue l'autoformation de l'hétéroformation. Il présente l'autoformation en réitérant à plusieurs reprises le caractère volontaire de l'individu : l'engagement dans une démarche d'apprentissage est à l'initiative de l'individu lui-même. Quant à l'hétéroformation, elle implique l'intervention d'autrui pour former (une institution par exemple). Prévost relate que ces visions de la formation traduisent différentes conceptions de l'apprentissage.

Ainsi, l'engagement d'un individu dans une démarche d'autoformation ou d'hétéroformation est un des paramètres qui permet de décrire son rapport au savoir. Par exemple, si l'individu considère la connaissance comme quelque chose de construit par chaque individu, alors l'autoformation est un moyen tout indiqué de réaliser des apprentissages. Si la connaissance est considérée comme transmissible, alors l'hétéroformation se présente comme un moyen de la transmettre. Cependant, si la connaissance est considérée comme construite par l'activité du sujet, comme c'est le cas dans la théorie piagétienne de l'intelligence, l'hétéroformation est limitée : la connaissance ne peut pas être transmise complètement intacte, sans appropriation par l'individu à l'aide du processus d'abstraction réfléchissante que nous avons décrit. L'autoformation s'apparente au courant constructiviste de l'apprentissage, voulant que l'apprenant soit engagé activement dans la structuration de sa formation. Not (1979) situe cette dernière conception de l'apprentissage, celle où le sujet est au centre de ses apprentissages, dans son contexte historique en évoquant la philosophie de l'éducation de Rousseau, les théories évolutionnistes du XIXe siècle et plusieurs pédagogues du XXe siècle dont Dewey, Freinet et Piaget.

En somme, Not associe l'hétérostructuration de la connaissance d'un individu à la pédagogie traditionnelle qu'il décrit en disant :

> La transmission des contenus est probablement la forme la plus archaïque et la plus simple qu'ait pu prendre la pédagogie de la connaissance : l'idée première est de faire passer le savoir de celui qui sait à celui qui ignore; ainsi s'organise un système de rapports entre le savoir et les personnes. (p. 23)

Il est à noter que cet extrait ne distingue pas connaissance et savoir, contrairement à la perspective socioconstructiviste selon laquelle le savoir est partagé par une communauté et la connaissance est propre à une personne. Par extension, cet extrait associe l'autostructuration de la connaissance aux pédagogies nouvelles disant que « [les méthodes qui] recommandent l'autostructuration [de la connaissance] sont nées d'une critique des méthodes d'hétérostructuration » (p. 22). Toutefois, l'hétéroformation et l'autoformation ne sont pas complètement détachées l'une de l'autre. Prévost (1994) propose plusieurs intermédiaires entre les deux : l'enseignement individualisé, l'autoformation assistée, l'autodidaxie et l'apprentissage expérientiel. Ainsi, Prévost situe la formation sur un « continuum qui évolue entre un pôle hétérostructuré et un pôle autostructuré » (p. 35) :

> D'un côté un système où l'individu n'aurait aucune action sur sa propre formation, une sorte de conditionnement, de dressage, que nous appellerons l'hétéroformation, un système rationnel où l'apprentissage a été pensé et apporté pour l'autre, par les autres. De l'autre côté, un système où l'individu créerait de toute pièce ses objectifs et ses moyens. Là, personne d'autre n'interviendrait, l'apprentissage serait une auto-construction de soi. (p. 35)

Dans une perspective constructiviste, les connaissances sont construites par l'activité intellectuelle du sujet. En ce sens, la possibilité qu'une connaissance soit structurée par autrui est un non-sens. Cependant, la structuration de la formation, et non de la connaissance, fait écho à l'implication de l'individu dans l'organisation ou l'institutionnalisation de ses apprentissages. Une formation exclusivement structurée par autrui, comme un programme de formation scolaire, laisse peu de place à cette implication. À l'inverse, une formation structurée par la personne elle-même demande une implication non seulement dans la construction des connaissances, mais aussi dans la recherche et la construction des situations d'apprentissage. Cette personne tente alors de se placer volontairement dans des situations qui lui permettront de réaliser un projet d'apprentissage. Ces deux concepts, ceux d'autoformation et d'hétéroformation, nous permettront d'explorer les efforts qui sont faits pour se placer volontairement dans des situations d'apprentissage par les programmeurs et programmeuses informatiques.

2.6 Objectif de recherche et question de recherche

Le cadre théorique nous a conduit à définir l'engagement comportemental, l'engagement cognitif et l'engagement affectif dans l'activité de programmation informatique. Notre recherche a pour objectif d'explorer les conditions qui peuvent conduire à ces trois formes d'engagement. Pour l'engagement comportemental, nous avons discuté de propositions qui affirment que les situations de programmation informatique doivent laisser une marge créative au programmeur ou à la programmeuse. Ainsi, nous viserons à caractériser la marge créative dans des situations de programmation informatique. Pour l'engagement cognitif, nous avons discuté du processus d'abstraction comme composante de la pensée informatique; ainsi viserons-nous à caractériser l'abstraction dans la programmation informatique. Finalement, pour l'engagement affectif, nous avons discuté du rapport au savoir des personnes; nous viserons à caractériser ce rapport au savoir chez des programmeurs et programmeuses. Nous formulerons la question de recherche ainsi : Quelles sont les conditions supportant l'engagement dans l'activité de programmation informatique ? Trois sous-questions conduiront à identifier ces conditions :

1) Quelles sont les manifestations d'un engagement comportemental dans l'activité de programmation informatique en lien avec le rapport au savoir?
2) Quelles sont les manifestations d'un engagement cognitif dans l'activité de programmation informatique en lien avec le processus d'abstraction ?
3) Quelles sont les manifestations d'un engagement affectif dans l'activité de programmation informatique en lien avec la marge créative ?

3 Méthode

3.1 Approche

La présente étude est une recherche interprétative visant à développer une compréhension riche de l'engagement dans l'activité de programmation informatique. La collecte de données consiste en des entretiens semi-dirigés individuels réalisés auprès d'un échantillon intentionnel. Avec ce type d'échantillon, « les éléments de la population sont choisis sur la base de critères précis, afin que les éléments soient représentatifs du phénomène à l'étude » (Fortin et Gagnon, 2016, p. 271). Fortin et Gagnon (2016) ajoutent qu'un tel échantillon « est souvent utilisé par les chercheurs en recherche qualitative pour sélectionner des sites particuliers, des personnes ou des activités dont ils espèrent obtenir des données informatives riches et significatives » (p. 271). Les entretiens que nous avons réalisés ont une portée ethnographique : ils visent à comprendre l'expérience commune, les pratiques et les croyances qui émergent d'une culture partagée (Brenner, 2006), dans ce cas-ci celle de la programmation informatique.

Nous avons établi que le concept d'engagement, tel qu'il est entendu en sciences de l'éducation, avec ses dimensions comportementales, affectives et cognitives, est peu abordé dans les recherches sur l'activité de programmation informatique. Dans ce contexte, la présente étude se veut exploratoire et vise à ouvrir des perspectives de recherche afin de mieux comprendre cet engagement. Notre recherche se veut inductive et émergente, c'est-à-dire que « le monde subjectif des participants est analysé pour produire une compréhension conceptuelle propre à la collecte de données » (Brenner, 2006, p. 360, traduction libre). En ce sens, nous adoptons une posture épistémologique socioconstructiviste de la connaissance.

3.2 Sujets et recrutement

Dans le contexte d'une recherche interprétative, Magnusson et Marecek (2015) proposent de constituer un échantillon à partir d'individus qui sont le plus susceptibles d'avoir vécu des expériences pertinentes par rapport à la question de recherche. C'est pourquoi les sujets ont été recrutés sur la base d'un critère : ils devaient avoir au moins cinq ans d'expérience cumulée en programmation informatique. Premièrement, ce critère nous permet de postuler que les sujets vivent ou ont déjà vécu l'expérience de l'engagement en programmation informatique étant donné qu'ils ont persisté dans leur pratique. Deuxièmement, de nombreuses études tendent à mettre en lumière des différences entre les personnes débutantes en programmation informatique et les personnes expertes. Par exemple, Kölling (1999) identifie des difficultés vécues par les novices en fonction des différents paradigmes de programmation. De façon similaire, Lister (2011) souligne que les novices n'appliquent pas les mêmes raisonnements et que cela les amène à vivre des difficultés supplémentaires. Dans cette optique, il nous

est apparu utile d'exclure les novices dans l'éventualité où ces difficultés pourraient avoir un impact sur leur engagement.

Les sujets ont été recrutés via un courriel envoyé sur deux listes de diffusion de l'Université Laval, une pour l'ensemble du personnel et l'autre pour l'ensemble des étudiants et étudiantes. Le courriel a été envoyé le 7 septembre 2017.

Magnusson et Marecek (2015) affirment qu'il n'existe pas de consensus quant au nombre idéal de sujets pour une recherche interprétative. Elles soulignent que les recommandations données dans la littérature pour le nombre minimal de sujets sont disparates, certaines plaçant le minimum à six (6) alors que d'autres le placent à 30. Elles se positionnent disant que ce nombre dépend de la question de recherche et de la profondeur des entretiens réalisés. Pour ces raisons et considérant la nature exploratoire de notre étude, nous avons opté pour un nombre élevé de sujets. L'objectif a ainsi été fixé à 20 sujets au total.

Vingt-cinq (25) personnes ont répondu au courriel de recrutement et ont manifesté de l'intérêt à participer. Toutes ces personnes se sont fait proposer un choix de trois plages horaires en vue d'un entretien, avec la possibilité de demander une plage horaire qui leur convenait davantage. Sept (7) personnes n'ont pas répondu à cette proposition et n'ont donc pas été rencontrées en entretien. En somme, ce sont dix-neuf (19) entretiens qui ont été réalisés.

L'entretien réalisé auprès du sujet 19 a été rejeté en raison d'un problème d'enregistrement audio : l'ordinateur qui enregistrait l'entretien était branché dans une barre multiprise, laquelle n'était toutefois pas branchée dans une prise de courant. L'ordinateur est entré en veille après les 15 premières minutes de l'entretien à notre insu, laissant environ 30 minutes non enregistrées. Les données ainsi recueillies étaient trop fragmentaires pour être utilisables dans le cadre de la recherche. Aucun problème technique n'est survenu avec les autres sujets.

Ainsi, le projet de recherche s'appuie sur dix-huit sujets (N=18), tous programmeurs ou programmeuses ayant cumulé au minimum cinq ans d'expérience. Parmi ces sujets, deux sont des femmes (n=2) et les autres sont des hommes (n=17). Nous tenons à préciser que le courriel de recrutement (annexe 2) avait été rédigé de façon épicène afin d'encourager femmes et hommes à participer sans discrimination.

3.3 Déroulement des entretiens

Les entretiens se sont déroulés entre le 28 septembre et le 26 octobre 2017 à l'Université Laval. À leur demande, deux (2) sujets ont été rencontrés à leur domicile afin de les accommoder en raison d'obligations familiales. Deux (2) sujets ont été rencontrés à leur lieu de travail afin de les accommoder. Les autres sujets au nombre

de quatorze (14) ont été rencontrés dans un bureau à l'Université Laval. Tous les entretiens se sont déroulés en privé.

En premier lieu, les sujets ont été accueillis par l'intervieweur puis invités à lire le formulaire de consentement et à poser leurs questions avant d'y apposer leur signature. En second lieu, l'intervieweur les informait du début de l'enregistrement audio puis l'entretien débutait. Les sujets ont été informés que l'entretien devait durer environ 45 minutes. L'entretien le plus court a duré 34 minutes, alors que l'entretien le plus long a duré 75 minutes. La durée moyenne est de 56 minutes et la durée médiane de 57 minutes.

Ils étaient également informés que cette durée pouvait varier en fonction des réponses qu'ils fournissaient, selon qu'ils avaient ou non le désir de s'exprimer davantage, mais qu'en aucun cas ils ne seraient interrompus par l'intervieweur. Les sujets étaient informés qu'ils pouvaient à tout moment revenir sur des propos qu'ils avaient tenus afin de les préciser ou d'établir des liens. L'intervieweur les informait aussi de son rôle qui consistait à les amener à étayer leur réflexion sans lui-même prendre position par rapport aux idées exprimées.

3.4 Le contenu des entretiens

L'annexe 1 présente le guide d'entretien qui a été élaboré à partir de la problématique et du cadre théorique. Il est divisé en trois thèmes. Le thème 1, *Récit de l'expérience en programmation informatique*, touche principalement à l'engagement comportemental. Il vise à donner l'occasion au sujet d'étayer son expérience en programmation informatique et la façon par laquelle il l'a apprise. Le thème 2, *Développement de compétences en programmation*, touche à l'engagement cognitif et à l'engagement affectif. Étant donné que les résultats ont fait l'objet d'une analyse émergente, le cadre théorique a été amené à se préciser à certains égards au fur et à mesure que la recherche progressait. Ainsi, le choix d'écarter le concept de *compétence* a été fait puisqu'il s'est avéré inopérant pour relier les propos de nos sujets. Toutefois, le contenu du thème 2 a pu être réinvesti aisément dans l'analyse. Le thème 3, *L'apprentissage pour tous de la programmation informatique*, visait à recueillir les idées et perceptions des sujets en lien avec l'apprentissage pour tous de la programmation dès l'école primaire. L'évolution de la question de recherche nous a conduit à ne pas aborder le contenu de ce thème dans les analyses étant donné que les propos des sujets nous amenaient sur des pistes trop nombreuses, ainsi a-t-il fallu faire des choix. Ces pistes concernaient, par exemple, la forme scolaire, l'histoire de l'éducation, les origines de l'informatique, le développement du marché de l'emploi, la croissance de l'intelligence artificielle, la structure des programmes de formation, la maîtrise de la langue, la scolarisation en bas âge, et beaucoup d'autres sujets. Ainsi, il nous est apparu prudent de concentrer nos efforts autour de la question de recherche, tout en n'excluant pas que ces résultats puissent alimenter de futures publications.

3.5 La posture épistémologique du chercheur

D'un point de vue personnel, l'auteur considère qu'il aborde la recherche avec une posture socioconstructiviste et se montre prudent, voire sceptique, face aux autres façons d'aborder la recherche lorsque celle-ci s'intéresse aux représentations des gens, à leur expérience, à leurs souvenirs et à leurs perceptions. Cette posture épistémologique amène à davantage d'ouverture quant aux liens à établir entre des idées similaires exprimées de façons différentes et par différents participants ou participantes. Elle peut ainsi amener à des idées en apparence généralisables, car interprétées par le chercheur lui-même, mais qui au fond demandent à être étudiées davantage. Nous réitérons donc que la présente étude n'a pas de visée universelle, mais que ses résultats pourront conduire à établir des devis de recherche visant la généralisation.

3.6 Méthode d'analyse

Chaque entretien a d'abord été retranscrit intégralement. La retranscription a été faite sans logiciel dédié à la tâche, de façon manuelle à l'aide de l'éditeur de texte Kate. Aucun logiciel de reconnaissance vocale n'a été utilisé. La retranscription a ensuite fait l'objet d'une relecture puis d'une codification en fonction des codes établis afin d'identifier et de classifier des passages d'intérêt en lien avec la question de recherche.

Les codes ont d'abord été établis à partir de l'analyse préliminaire de la retranscription du sujet 01 alors que les entretiens avec d'autres sujets n'avaient pas encore eu lieu. Ils ont été déterminés à partir de relectures successives et d'ajustements afin que leur nombre permette un niveau de précision qui correspond aux propos du sujet 01. Par exemple, le sujet 01 discutant séparément de mathématiques et de logique, deux catégories distinctes ont été établies plutôt qu'une seule les regroupant. Avec l'avancement des analyses, la pertinence des codes a pu être validée par l'équilibre qui s'est construit dans le nombre de propos retenus par les sujets pour chaque catégorie.

Les codes établis ne correspondent pas aux thèmes de l'entretien. Si, au départ, nous avions convenu de classifier les extraits des sujets en fonction des énoncés auxquels ils réagissaient ou des questions auxquelles ils répondaient, nous avons réalisé que cela ne permettait pas de capter avec précision les idées sous-jacentes. Par exemple, des sujets ont traité de leurs motivations à programmer en différents moments de l'entretien et pas seulement en réponse à la question explicite qui concernait leurs motivations. De plus, étant donné la nature exploratoire de l'étude, nous avons relancé les sujets à plusieurs reprises, notamment en pointant des contradictions afin de les amener à préciser leurs propos. Cela nous a conduit à un enrichissement des entretiens, mais aussi à un élargissement de la portée des questions prévues au guide d'entretien.

La codification des retranscriptions a été faite à l'aide d'un logiciel maison dont la fonction principale est de permettre l'annotation et la classification d'extraits de retranscriptions (Lepage, 2018). Une fois les extraits codifiés grâce à ce logiciel, ils ont été exportés en format CSV afin de les manipuler dans différents logiciels dont le tableur OpenOffice Calc. Comme le logiciel n'a servi qu'à faciliter la classification manuelle des extraits, il n'a pas été impliqué dans l'analyse sémantique des extraits, laquelle a été réalisée par nous, en aval de cette codification.

Les entretiens étaient semi-dirigés, ce qui signifie que les individus rencontrés ont été amenés à préciser leur pensée notamment par le biais de questions spontanées en lien avec leurs propos. Cela a donné lieu à des discours construits dans le vif de l'entretien truffés d'hésitations, de retours en arrière, de phrases inachevées, d'idées mises entre parenthèses. La relecture d'une telle retranscription nous est apparue par moment ardue en raison de la difficulté à transposer à l'écrit la ponctuation et les hésitations, par exemple. Ainsi, afin d'améliorer la lisibilité et de faciliter la compréhension, les citations qui figurent dans les analyses individuelles ont été formatées suivant ces règles :

- Nous avons ajouté, entre crochets, des mots permettant de compléter la citation. Ces mots ont été déterminés à partir du contexte dans lequel la citation a été extraite. Par exemple, lorsqu'une citation débute par un pronom référant à un autre élément qui ne fait pas partie de la citation, nous avons ajouté entre crochets le mot auquel réfère ce pronom. Si le pronom réfère à une idée plutôt qu'à un pronom, nous avons déterminé un mot que nous évaluions juste.

- Les crochets contenant trois points de suspension, […], peuvent signifier deux choses : un passage a été retiré sur la base qu'il discute d'un autre sujet que celui rapporté par la citation. Dans ce cas, nous avons été prudent afin de nous assurer que l'enchaînement des mots, par exemple entre sujets et verbes, soit fidèle à ce qui était exprimé dans la retranscription. Ils peuvent aussi signifier que nous avons retiré un mot que le sujet a repris tout de suite après. Par exemple, « le… la résolution de problème » (S03,586) aurait été remplacé par « […] la résolution de problème ». Dans les cas où le ton employé amenait à considérer cette reprise de mots comme une hésitation plutôt que comme une erreur sans conséquence, l'extrait est demeuré inchangé.

- Chaque citation est suivie d'une référence vers la retranscription intégrale. Cette référence est placée entre parenthèses et inclut le numéro du sujet suivi d'une virgule, elle-même suivie du numéro de ligne seul, ou des numéros de lignes séparées par un tiret.

- Les débuts et fins de phrases, ainsi que l'emploi des virgules, ont été ajustés de façon à se substituer aux intonations employées et permettre un enchaînement des idées fidèle à l'entretien. Cela a été fait

notamment sur la base des recommandations du *Chicago Manual of Style* (University of Chicago, 2017).

- Aussi sur la base du *Chicago Manual of Style*, les erreurs évidentes de grammaire ou d'orthographe ont été corrigées sans annotation supplémentaire afin de faciliter la compréhension. Par exemple, « des travail » (S03,515) a été remplacé par « des travaux » (S03,515).

- Les phrases négatives se sont vu ajouter le mot « ne » lorsqu'il avait été omis à l'oral afin de faciliter la lecture, et ce, sans annotation supplémentaire.

3.7 Description des codes établis pour l'analyse

Les codes établis ont été regroupés en fonction des trois types d'engagements de notre définition de l'engagement dans l'activité de programmation informatique : engagement comportemental, engagement cognitif et engagement affectif. Plusieurs codes ont été établis pour chacun de ces types d'engagement. Toutefois, étant donné la proximité entre ces codes et entre les propos qui y sont classifiés pour chaque sujet, les analyses individuelles de la section suivante seront découpées au niveau des types d'engagement et non au niveau des codes pour chaque type.

3.7.1 Engagement comportemental

Cette catégorie regroupe tous les extraits en lien avec le contexte dans lequel s'inscrivent les propos du sujet. Tel que précisé dans notre définition de l'engagement, l'engagement comportemental concerne ici les manifestations observables de l'engagement. Nous nous intéressons donc dans le discours des sujets aux réalisations qu'ils ont accomplies, au récit de leur expérience et aux anecdotes qu'ils racontent. Il peut s'agir d'un contexte personnel en lien, par exemple, avec la famille ou le milieu scolaire.

3.7.1.1 Débuts en programmation informatique

Ce code touche à la façon dont le sujet a appris la programmation informatique. Par exemple, c'est dans cette section que se trouvent des propos décrivant un apprentissage autodidacte, un apprentissage dans le cadre de programmes scolaires, ou un apprentissage fortuit.

3.7.1.2 Analogies établies

Plusieurs sujets ont établi des analogies entre la programmation informatique et d'autres activités humaines qu'ils ont pratiquées eux-mêmes ou qu'ils ont observé quelqu'un pratiquer. Ces analogies sont des comparaisons que les sujets établissent entre la programmation informatique et d'autres types d'activités

humaines. Ils les utilisent pour illustrer leur expérience de la programmation, et elles peuvent être pertinentes pour interpréter leurs propos.

3.7.1.3 Anecdotes racontées

Ce code vise à regrouper les anecdotes personnelles racontées par les sujets. Ces anecdotes n'ont pas toujours de lien entre elles mais peuvent être réinvesties pour illustrer certains extraits associés à d'autres codes.

3.7.2 Engagement cognitif

Cette catégorie regroupe les codes en lien avec l'activité cognitive impliquée dans la programmation informatique telle que perçue par les sujets.

3.7.2.1 Mathématiques

Ce code regroupe les extraits permettant d'inférer la vision qu'ont les sujets des mathématiques et de la place qu'ils y voient en programmation informatique.

3.7.2.2 Abstraction

Ce code regroupe les extraits permettant de comprendre la définition que donnent les sujets à l'abstraction. Ainsi, il ne regroupe pas que des propos en lien avec la conception de l'abstraction que nous avons étayée dans le cadre théorique, celle en lien avec l'abstraction réfléchissante. La discussion qui suivra l'analyse permettra de relier ces conceptions à notre cadre théorique ou de les en distinguer, s'il y a lieu.

3.7.2.3 Approximations successives

Ce code regroupe les propos suggérant le recours à la méthode des approximations successives, c'est-à-dire à des essais en continu ou itératifs visant à se rapprocher progressivement de la solution recherchée. Ces propos sont à distinguer de ceux traitant de l'essai-erreur, ce dernier faisant intervenir le concept d'aléation. Par exemple, les propos en lien avec l'idée de démarche itérative ont été classifiés dans cette catégorie.

3.7.2.4 Algorithmique

Ce code regroupe les propos permettant d'inférer la définition que les sujets donnent à l'algorithmique. Par exemple, les propos visant à distinguer l'algorithmique de l'écriture du code informatique ou à la situer dans l'activité de programmation seront discutés sous ce code.

3.7.2.5 Outils sémiotiques

Ce code regroupe les propos discutant des langages de programmation ou d'analyse en tant que moyen de communication entre humains ou entre l'humain et la machine. Aucun sujet n'a évoqué le concept d'outil sémiotique explicitement, mais le concept parvient à rassembler plusieurs idées évoquées.

3.7.2.6 Essai-erreur ou tâtonnement

Ce code, contrairement au code *approximations successives*, regroupe les propos faisant intervenir une démarche aléatoire en certains moments dans l'activité de programmation informatique. Par moment, les propos qui s'y trouvent touchent davantage aux méthodes concrètes en général qu'à l'essai-erreur spécifiquement. La discussion qui suivra l'analyse permettra d'expliquer davantage ce regroupement.

3.7.2.7 Logique

Suivant notre préoccupation d'explorer la distinction que font les sujets entre mathématiques et logique, s'il y a lieu, nous avons regroupé sous ce code les propos permettant d'inférer la place de la logique dans la programmation informatique.

3.7.3 Engagement affectif

Cette catégorie vise à regrouper les codes en lien avec les dimensions sociales et émotives impliquées dans la programmation informatique en tant qu'activité humaine. Suivant notre cadre théorique, un tel regroupement permet de baliser les attitudes mobilisées en programmation informatique avec les nuances que cela peut impliquer.

3.7.3.1 Relations de travail

Ce code regroupe les propos en lien avec la programmation en équipe, en contexte professionnel ou non. Nous y avons aussi regroupé les propos discutant des interactions entre corps de métier, dont certains apparentés à la programmation informatique.

3.7.3.2 Approche client

Ce code vise à regrouper tout ce qui concerne la programmation informatique réalisée pour autrui et non pour soi-même. En ce sens, bien que le terme client soit employé, ce tiers peut être le client dans une relation d'affaires, ou il peut être un ami pour qui le sujet a programmé.

3.7.3.3 Élégance

Le terme *élégance* pour ce code a été choisi à partir des entretiens avec plusieurs sujets. Le code regroupe des propos qui comparent ou associent la programmation à l'art, des propos qui discutent de ce que sont les bonnes pratiques, des propos qui discutent de la place de la créativité. L'élégance peut être considérée comme une caractéristique d'une solution créative simple, mais tout de même efficace (Madni, 2011).

3.7.3.4 Frustration

Ce code regroupe les propos des sujets en lien avec la réaction de frustration qui peut survenir en programmation. Il regroupe principalement les réactions à l'énoncé 3 du thème 2 où le sujet doit se prononcer en accord ou en désaccord avec l'idée que la programmation peut amener à vivre des frustrations. Le code a parfois été élargi pour intégrer ce qui touche aux attitudes personnelles face à la programmation informatique, étant donné que les sujets établissaient des liens entre l'idée de frustration et d'autres types de réactions.

3.7.3.5 Motivation

Ce code regroupe les propos des sujets où ils expliquent leurs motivations à programmer. Ces motivations ont principalement été discutées dans le premier thème de l'entretien. Nous avons aussi investigué les variations dans les motivations à programmer au fil du parcours de chaque sujet.

4 Résultats

La section suivante vise à présenter une analyse détaillée de chacun des entretiens réalisés ayant eu lieu auprès des 18 sujets. Pour chaque sujet, l'analyse est divisée en fonction des trois types d'engagement : comportemental, cognitif et affectif.

4.1 Analyse individuelle du sujet 01

4.1.1 Engagement comportemental

Le sujet 01 a appris à programmer dès la fin des années 1970 de façon autonome, alors qu'il était âgé d'environ 17 ans : « Fin des années 70 j'ai découvert la programmation, c'est vraiment ce qui m'a attiré vers la programmation, c'est le désir de programmer, de taper du code, puis c'est ce qui m'a attiré de faire un bac » (S01, 34-37). Il raconte ses premières expériences en programmation ainsi : « Je voyais à la télévision des jeux que je reproduisais sur mon TRS-80 pour mes chums, c'était plus pour mes amis que pour moi ! » (S01, 76-78). Il affirme que de voir les autres utiliser ses programmes était une source de valorisation personnelle : « Moi mon trip c'était vraiment de programmer puis j'aimais voir les autres utiliser mes [programmes], c'était […] auto-valorisant de voir les autres utiliser tes programmes » (S01, 78-81).

Le sujet 01 relate des anecdotes précises sur ses premiers contacts avec la programmation. Par exemple, il se rappelle son premier ordinateur et la mémoire vive dont il disposait : « C'était un TRS-80... couleur... 4K de mémoire... c'était en VC, langage de programmation, avec des petites cassettes, on sauvegardait là-dessus, c'est ça la technologie de l'époque » (S01, 51-54). Il ouvre parfois des parenthèses pour expliquer ses anecdotes à partir d'éléments de culture informatique. Par exemple, il explique les motifs qui ont mené à la création du langage C# par Microsoft avant d'en venir à l'utilisation qu'il en fait :

> On peut faire une petite histoire, c'est que C Sharp est né… par accident, c'est que Sun, la compagnie Sun avait fait le langage Java qui est un très très beau langage bien structuré, les chercheurs qui à force de voir les programmeurs ils ont dit « on va faire un langage qui est plaisant ». (S01, 122-127)

Pour lui, le critère de base de la programmation est le recours à des structures conditionnelles et à la dynamique de réaction à des événements. Le sujet évoque que la programmation informatique est toujours reliée à un autre domaine d'application : « L'informatique comme telle, ce n'est rien, quand tu vas faire des programmes tu vas l'appliquer à un autre domaine, aux finances, à un jeu, eee [mot inaudible] faire une liste d'épicerie » (S01, 445-448). Le sujet suggère que la tâche de programmation informatique peut varier selon les outils utilisés et donne en exemple un outil qu'il utilise :

> Microsoft, je dois leur donner ça. Visual Studio c'est vraiment un bel environnement pour déboguer des programmes, pour construire tes applications, les interfaces sont là pour aider le programmeur, c'est le fun quand t'as un... c'est comme en menuiserie, quand t'as les bons outils ça va être encore plus le fun de programmer. (S01, 110-115)

Selon lui, la programmation est omniprésente : « Il y a de la programmation partout, même […] le toaster est rendu avec plein de boutons » (S01, 910-912).

4.1.2 Engagement cognitif

Le sujet 01 raconte avoir été initié à l'analyse informatique lors de sa formation universitaire, et il la présente comme un moyen d'accroître la qualité et la vitesse du développement :

> Avant de programmer là, on va réfléchir puis on va designer nos affaires, puis ça a l'air sur le coup qu'on recule, qu'on prend plus de temps, mais on gagne tellement à faire le plan, on diminue le risque d'erreur, puis ça nous permet plein plein plein d'affaires qu'on avait tout bien designé, qu'on avait tout bien organisé nos affaires, après ça on va à vitesse grand V. (S01, 180-187)

En ce sens, afin d'illustrer l'équilibre entre l'action et l'analyse, le sujet 01 fait une comparaison entre un informaticien et un menuisier à plusieurs reprises :

> Moi souvent je compare justement un informaticien à un menuisier, il va bricoler quelque chose. Puis on s'entend qu'un bricoleur qui veut faire une table, il ne se fera pas un plan de sa table bien bien longtemps, il prend une planche de même, quatre pattes, je ne ferai pas une analyse très très sophistiquée ou je n'en ferai pas puis je vais tout de suite bricoler. (S01, 706-713)

Toujours en discutant de l'analyse en informatique, le sujet 01 fait un lien entre l'informatique et les mathématiques tout en ayant de la difficulté à y mettre des mots :

> Un algorithme, bien... je pense que c'est mathématique, à moins que je me trompe, tu sais les conditions, puis souvent quand tu veux faire une intégrale, souvent il va y avoir des conditions, si tu la fais de telle façon ou de telle façon. (S01, 356-360)

Il fait principalement ce lien à partir de ses intérêts personnels. Il affirme avoir un grand intérêt pour les mathématiques, ce qui l'a mené à choisir plusieurs cours avancés de mathématiques dans sa formation : « Quand j'étais plus jeune, vu que je tripais sur les maths, j'avais fait tous les cours de maths au cégep, 101, tous les cours, même ceux [qui ne sont] pas obligatoires » (S01, 354-336). Le lien entre mathématiques et informatique est selon lui au niveau de la logique et de la résolution de problèmes : « Si je fais juste résoudre une intégrale, il y a un problème... en informatique souvent il va y avoir un besoin ou un problème à résoudre, fait que faut que je pense à un algorithme » (S01, 352-356). Il établit aussi ce lien au niveau de l'algorithmique en donnant l'exemple de la résolution d'une intégrale : « Un algorithme, bien... je pense que c'est mathématique, à moins que je me trompe, tu sais les conditions, puis souvent quand tu veux faire une intégrale, souvent il va y avoir des conditions » (S01, 356-359). Le sujet 01 donne un exemple de cette proximité entre mathématiques

et informatique : « Quand […] tu fais des itérations, bien faut que tu saches compter comme il faut, des fois […] va falloir que tu fasses de la récursivité, faut que tu saches... comprendre les boucles » (S01, 1061-1065).

Le sujet 01 utilise le terme abstraction pour décrire ce qui nous apparaît relever de l'étape de modélisation d'un problème en informatique. Ainsi, lorsqu'il parle d'abstraction, il n'est pas question du processus d'abstraction réfléchissante que nous avons décrit. Il aborde cette étape de modélisation à partir d'exemples issus de programmation orientée objet : « Il y a un niveau d'abstraction qu'il faut que tu développes là pour... plus tu vas comprendre les relations, que ton objet dérive d'un autre objet, bien tu vas pouvoir mieux organiser ton univers » (S01, 496-500). Il évoque que cette modélisation trouve également une utilité dans la communication avec d'autres personnes qui ne partagent pas nécessairement son expertise en programmation : « Puis quand tu as défini ton univers d'objets, bien c'est... c'est plus clair après, bien je trouve, […] quand tu parles à un client » (S01, 512-514). Ses propos laissent suggérer que l'approche orientée objet n'est pas la seule façon de réaliser une modélisation en programmation. Il nomme par exemple la création de librairies pour encapsuler certaines fonctionnalités :

> Dans mes pages Web, je vais appeler une commande qui est centralisée dans ce qu'on appelle une librairie, puis toutes tes pages appellent ta librairie, si je te demande de juste inverser nom et prénom, bien tu vas l'inverser là et il va le changer aux 8 places. (S01, 291-295).

Quant à l'utilité de cette analyse et modélisation, le sujet 01 relate à plusieurs reprises qu'elles peuvent servir en phase préparatoire d'un projet afin de communiquer avec le client ou d'autres programmeurs dans le cas d'un travail en équipe :

> Il ne faut pas montrer les lignes de code à un client. Ce qui est important, quand on arrive en analyse informatique, c'est de trouver un langage de plus haut niveau pour parler à ton client. Puis avec le langage objet encore plus haut niveau, quand tu fais des petits dessins, puis tu n'as pas besoin de lui montrer toutes les propriétés d'un objet [comme] la date de création, tu lui présentes les objets qu'il y a dans son univers puis lui il se retrouve là-dedans. (S01, 522-530)

Le sujet 01 décrit la démarche de programmation informatique en évoquant un processus de validation de la solution. Il affirme qu'il est possible de réaliser après coup qu'un système répond mal au besoin du client :

> Si tout est parti dans une direction puis que tu te rends compte qu'il y a des activités du client qui ne peuvent pas rentrer dans ton modèle, bien ton modèle il n'atteint pas le but, donc c'est comme ça que je valide le modèle (S01, 313-317).

4.1.3 Engagement affectif

Le sujet 01 évoque le plaisir qu'il avait à programmer à ses débuts vers 17 ans. Il aimait entrer des lignes de code pour faire réagir un automate : « tu sais comme moi j'ai adoré programmer puis de dire "hein, je peux rentrer des lignes de code pour faire réagir un système, un automate, c'est trippant" » (S01, 1069-1072). Réaliser

des projets pour autrui a aussi été une source de motivation (l'exemple a déjà été donné des jeux qu'il programmait pour sa mère et ses amis).

Le sujet 01 décrit plusieurs contextes professionnels dans lesquels il a travaillé. Par exemple, il lui est arrivé de travailler à titre de seul programmeur sur un projet d'envergure et de devoir occuper plusieurs rôles simultanément : analyste, programmeur et gestionnaire de projet. Il affirme que dans des contextes où les ressources sont insuffisantes, il est courant et normal d'abandonner certaines pratiques d'analyse :

> Les programmeurs, souvent ils ont de la pression, puis faut qu'ils tournent les coins ronds, même si dans les règles de l'art, faut que tu fasses les choses de telle façon, mais ça va prendre plus de temps fait que faut que tu coupes telle affaire. (S01, 249-253)

Dans d'autres contextes, il a travaillé avec plusieurs programmeurs et analystes où chacun avait un rôle précis. Il évoque avoir travaillé en certaines occasions principalement comme analyste et dans ce cas, il n'était pas chargé du tout de l'écriture de code. Dans ce contexte, il soutient que la marge de manœuvre du programmeur est réduite afin de respecter les rôles de chacun.

Le sujet 01 discute à plusieurs reprises de la communication avec le client. Les propos en lien avec la communication avec le client sont à mi-chemin entre l'engagement affectif et l'engagement cognitif, étant donné qu'il affirme que le plan peut aussi servir à l'élaboration d'un contrat avec un client : « La finalité, ce qu'on appelle la portée là, quand tu rencontres le client, si tu ne définis pas une portée, c'est comme un contrat : on va faire ton système jusque-là, puis tes autres affaires, on ne les fera pas » (S01, 1013-1017). Cet exercice de création d'un plan dépasse donc la tâche purement informatique et se situe dans un contexte de relation avec d'autres personnes ou organisations. Pour élaborer ce plan, il évoque qu'il est question de comprendre l'univers du client : « C'est quand vraiment tu programmes pour quelqu'un d'autre, là faut que tu t'appropries tout […] l'univers du discours de la personne, tous les objets qu'elle utilise pour être sûr de... rien manquer » (S01, 690-694). Il souligne que cette étape de compréhension de l'univers du client ne se pose pas lorsqu'il programme pour lui-même en disant qu'il est important de « de faire un genre de contrat quand tu fais quelque chose pour un autre, quand tu le fais pour toi, t'as la vision que tu veux » (S01, 1019-2021).

Au niveau de la créativité en programmation informatique, le sujet 01 affirme se considérer comme un artiste :

> Souvent je dis à ma blonde "je suis un artiste", ma blonde dit "c'est vrai". Oui, je crée quelque chose, quand on programme c'est une création à la base. Je vais créer, ou nous allons créer parce que je le crée avec un client, mais on s'entend que le système d'information, il n'existait pas avant, fait qu'on a créé quelque chose. (S01, 1062-1067)

Il justifie l'idée sur la base de l'action de créer qui est au coeur de son travail. Il amène l'idée d'itération : « Tu peux avoir fait ton gabarit avant puis l'améliorer après, c'est une itération, mais oui je compare ça à l'art

effectivement » (S01, 1173-1175). Le sujet affirme avoir donné pour conseils à des programmeurs novices de ne pas être orientés exclusivement vers la finalité, mais aussi vers la démarche. Il donne de l'importance à la fierté dont doit témoigner un programmeur lorsqu'il montre son code à un autre programmeur.

Il résume cette démarche créative (et même co-créative) ainsi :

> Oui je crée quelque chose, quand on programme c'est une création à la base. Je vais créer, ou nous allons créer parce que je le crée avec un client, mais on s'entend que le système d'information, il n'existait pas avant, fait qu'on a créé quelque chose. (S01, 1163-1167)

Questionné sur les réactions émotives en programmation informatique, le sujet 01 trouve que le terme frustration est démesuré pour décrire ce qu'il peut vivre : « je n'irais pas jusqu'à un niveau de frustration, j'adore l'informatique fait que ça m'en prend beaucoup pour être frustré! » (S01, 616-618). Il affirme que le programmeur, en tant que personne, a un rôle à jouer dans la façon dont il vit les difficultés qu'il rencontre en programmant. La frustration lui apparaît comme une façon parmi d'autres de réagir face à un problème, et il évoque que la personne a le choix de la réaction qu'elle adopte : « Tu peux dire à la personne "c'est quelque chose que tu vas vivre, c'est à toi de décider comment tu le prends", tu sais si je t'insulte, c'est à toi de décider comment tu le prends » (S01, 574-577). Selon le sujet, quelqu'un qui vit souvent des frustrations en informatique n'est peut-être pas au bon endroit : « Je dirais si tu es en informatique puis tu es souvent frustré, peut-être que tu n'es pas à la bonne place » (S01, 581-583).

Finalement, au niveau de la place de l'erreur en programmation informatique, le sujet 01 reconnaît une place pour l'essai-erreur. Il nuance :

> Souvent ça demande toujours un peu de réflexion là, parce que c'est comme un problème mathématique que faut que tu réfléchisses à ce que tu fais, parce que si tu y vas par essai-erreur ça va te prendre 1000 ans. C'est comme un singe qui taperait du code, un moment donné faut que tu penses. (S01, 679-683)

Pour illustrer son usage de l'essai-erreur, il affirme : « Parfois, on se perd dans l'analyse alors que... tu fais juste deux trois petits tests puis tu as réglé ton problème » (S01, 684-686).

4.2 Analyse individuelle du sujet 02

4.2.1 Engagement comportemental

Le sujet 02 a été initié à la programmation informatique pour la première fois en secondaire 4 lors d'un cours d'informatique offert en option : « en secondaire 4 j'ai changé d'école, et puis j'ai pris un autre cours d'informatique, et puis là on enseignait vraiment la programmation » (S02, 186-187). La programmation leur était enseignée avec les langages Basic et Pascal. Il se souvient du matériel utilisé : « C'était sur des machines

Apple en noir et blanc mais avec interfaces graphiques, donc de quoi d'un peu... quand même un peu plus récent, un peu plus digne des années quatre-vingt-dix » (S02, 190-193). Il relate un cours d'informatique en secondaire 3 où la programmation n'était pas enseignée : « l'option en secondaire 3 c'était plus un cours d'initiation à l'utilisation de l'ordinateur […] c'était vraiment juste pour montrer c'était quoi un traitement de texte, un tableur et une base de données » (S02, 175-183). Il affirme qu'il avait des amis qui s'intéressaient à la programmation informatique, notamment un dont les parents travaillaient dans le domaine de l'informatique. Il dit s'être par la suite inscrit en technique de l'informatique au cégep sans trop se questionner et a réalisé un baccalauréat en informatique.

Le sujet 02 discute de plusieurs paradigmes de programmation :

> Toute formation moindrement sérieuse va enseigner la différence entre les principaux paradigmes de programmation […] qui sont utilisés dans l'industrie, la programmation […] qu'on appelait structurée ou impérative ou procédurale, c'est les bases rudimentaires de la programmation, d'écrire une séquence d'instructions à exécuter par ordinateur, mais c'est plus une façon qui est professionnelle de programmer de nos jours. (S02, 386-394)

Il affirme que ces paradigmes seront sans doute amenés à évoluer avec le développement de l'informatique (il donne l'exemple de l'intelligence artificielle) : « Quand on parle de paradigme de programmation, bien on ne sait pas ce que l'avenir nous réserve, on ne sait pas si c'est demain on va encore programmer […] le même genre de langages, avec les mêmes langages comme on fait actuellement » (S02, 433-438). Il présente l'informatique comme une branche appliquée des mathématiques, tout en soutenant qu'il y a une part de génie dans la science informatique : « à l'origine, l'informatique est issue, serait issue essentiellement des mathématiques mais... si on considère l'aspect électronique, il y a une part de l'informatique qui nous vient aussi probablement du secteur du génie » (S02, 657-661). Le sujet évoque le concept de mathématiques discrètes : « Du côté informatique […] on recourt aux mathématiques, mais on a recourt [aux] mathématiques qu'on appelle discrètes... qui sont... qui ne sont pas le même genre de mathématiques où dans le fond […] c'est des notions mathématiques qui sont... bien je ne suis pas mathématicien, je ne pourrais pas parfaitement le définir mais... c'est une branche des mathématiques où ce que c'est plusieurs notions disparates, plusieurs éléments d'informatique différents qui ne sont peut-être pas initialement tous reliés les unes aux autres » (S02, 613-623). Il évoque que la programmation n'est pas utilisée que pour développer des logiciels, mais qu'elle peut servir à développer des scripts pour gérer un parc informatique par exemple.

Le sujet établit un lien entre les langages de programmation et le champ de la linguistique. Il évoque la forme Backus-Naur pour « représenter des productions possibles des expressions dans un langage de programmation » (S02, 726-727). Selon ses propos, il s'agit d'un angle pour comprendre la relation des différents éléments entre eux dans un programme informatique. Il donne l'exemple d'imbrication des

instructions : « une procédure ou un appel de méthode va comprendre une ou plusieurs instructions » (S02, 730-731). Il établit un lien avec la grammaire hors contexte de Chomsky : « On parle de langages [informatiques], il y a par exemple le célèbre linguiste Noam Chomsky qui a élaboré les grammaires hors contexte, la forme Chomsky, en tout cas pour... des productions pour... des grammaires pour représenter des langages en informatique théorique » (S02, 713-717).

Le sujet 02 affirme que bien que tout le monde puisse programmer, ce n'est pas tout le monde qui aura le même intérêt. Il évoque le modèle RIASEC pour parler d'orientation professionnelle : « Si on s'inspire par exemple du RIASEC en matière d'orientation professionnelle, […] il y a des types de personnalités professionnelles qui sont pas mal plus appropriés pour... œuvrer... pour s'orienter vers l'informatique que d'autres » (S02, 767-771). Il donne des exemples : « Il y a des gens qui sont plus adeptes du plein air ou du travail manuel ou qui vont probablement éprouver plus de frustrations ou d'impatience à l'égard de l'aspect pointilleux de l'informatique » (S02, 772-775).

Le sujet raconte avoir rencontré un entrepreneur dans le domaine du jeu vidéo avec qui il a pu échanger. Il relate qu'il existe des programmes d'études qui visent explicitement ce domaine et s'adressent à des jeunes directement après le secondaire. Le sujet affirme qu'en ce moment il travaille avec des logiciels libres : « Je travaille essentiellement avec du logiciel libre, donc je fais le dépannage, l'administration de systèmes Linux en logiciel libre » (S02, 347-350).

4.2.2 Engagement cognitif

Le sujet 02 discute que la différence entre l'informatique et les mathématiques :

> C'est une façon de faire des mathématiques qui est appliquée à l'informatique, donc […] quand on compte en base deux dans un système binaire plutôt qu'en décimales, on applique la théorie des nombres, on applique le calcul au secteur de l'informatique, on parle de la notion d'algorithme, c'est une notion qui est plus exclusive à l'informatique. (S02, 624-630)

Il donne un exemple d'application : « cette implémentation là aussi a des abstractions, si on parle de circuits imprimés, en termes de génie informatique, en informatique on parle de porte logique pour représenter des transistors » (S02, 685-688).

Le sujet 02 évoque que l'apprentissage se fait entre autres avec du pseudo-code : « Quand […] il est question de niveaux d'abstraction, à l'école, ou au cégep et à... comme à l'université, on apprend à programmer aussi avec ce qu'on appelle du pseudo-code » (S02, 558-561). Il affirme qu'à l'intérieur du domaine des mathématiques, il y a plusieurs vocations possibles, certaines requérant davantage d'abstraction que d'autres. La place de l'abstraction peut varier selon les domaines :

> Selon la vocation aussi, il peut y avoir des vocations informatiques où est-ce qu'on a besoin de faire plus d'abstraction, particulièrement aussi il y a plusieurs questions mathématiques... comparativement à d'autres volets de l'industrie qui peuvent être plus techniques et que là, oui, les programmeurs vont être plus des bricoleurs. (S02, 882-888)

Le sujet 02 évoque l'existence de méthodes de développement servant à prototyper où les programmeurs développent rapidement sachant qu'ils auront à revenir sur le code par la suite : « Il y a de la programmation extrême qui sert essentiellement à prototyper, à développer rapidement, quitte à repasser sur... le travail accompli » (S02, 838-840). Lorsque le sujet discute de prototypage, nous comprenons qu'il envisage un prototype comme une solution volontairement imparfaite, développée rapidement afin de valider la faisabilité d'une solution ou les moyens pour la créer. Cela peut s'approcher de la preuve de concept discutée par d'autres sujets. Il nomme aussi des méthodes de développement en cascade ou agile. Il les décrit en nommant que l'on itère dans le développement en revoyant la planification à chaque itération.

Le sujet 02 affirme les programmeurs abordent le développement de différentes façons :

> Il y a des gens qui préfèrent se garrocher dans le code puis... développer au pif quitte à... repasser et réécrire, et il y a des gens qui, ne serait-ce qu'à l'école durant leurs études, qui préfèrent écrire sur papier un algorithme avant de se mettre […] à taper sur un clavier... ça dépend des goûts. (S02, 829-834)

Le sujet 02 évoque l'usage de la logique en informatique et nomme l'algèbre booléenne, mais ne développe pas outre mesure sur cet aspect.

4.2.3 Engagement affectif

Le sujet 02 affirme avoir toujours su qu'il ferait carrière en informatique. Il évoque brièvement aussi en parallèle des motivations personnelles reliées à la dimension artistique. Au cours de l'entretien, le sujet 02 relate plusieurs personnalités publiques qui font de la programmation informatique et qu'il a pu rencontrer à certaines occasions. Cela nous amène à considérer que la démarche créative en programmation informatique peut conduire à un certain prestige, qui peut agir comme une source de motivation.

Comme autre motivation, le sujet 02 évoque l'aspect artistique de la programmation informatique : « Il y a la dimension technologique, il y a la dimension industrielle, mais il y a aussi une dimension artistique, juste avec des applications qu'on... les jeux vidéos qu'on a eu l'habitude d'apprécier depuis son bas âge » (S02, 235-239). Il évoque que cela permet de créer quelque chose dont il a besoin ou qui lui plaît. Pour illustrer la dimension artistique, il donne l'exemple du jeu vidéo. Le sujet parle de « meilleures pratiques » (S02, 359) et donne l'exemple de la programmation MVC (modèle-vue-contrôleur) et la programmation orientée objet. Pour illustrer l'aspect artistique, il utilise aussi l'exemple de l'intégration Web évoquant que cela se rapproche de l'intégration multimédia :

> Du côté du navigateur Web, on a ce qu'on appelle des développeurs frontaux, front-end, qu'on appelle des intégrateurs Web qui sont formés en technique d'intégration multimédia qui est une formation qui a justement... un volet un peu plus artistique de design d'arts visuels graphiques […] pour lesquels il y a aussi un certain baccalauréat à l'université en arts visuels. (S02, 268-275)

Finalement, au sujet des réactions émotives, le sujet 02 ne développe pas sur l'idée de frustration, mais affirme que quelqu'un ayant un profil moins adapté à la programmation vivra davantage de frustrations.

4.3 Analyse individuelle du sujet 03

4.3.1 Engagement cognitif

Le sujet 03 a fait de la programmation informatique pour la première fois lors d'un cours optionnel alors qu'il était en secondaire 5 :

> C'était vraiment là les... les bases de la programmation. c'était en... VB... c'était en VB6, puis c'était vraiment les bases de la programmation, donc comment... comment programmer là vraiment tu sais des affaires de base, des petites interfaces, des petits jeux » (S03, 71-75).

Il avait une passion pour les jeux vidéos : « Je pense qu'au départ ça a été venu parce que tu sais je jouais beaucoup à des jeux vidéos » (S03, 57-59). Il affirme être revenu à l'université quelques années plus tard afin de compenser certaines lacunes en mathématiques : « Je travaillais en vision numérique, qui est un domaine assez haute technologie, qui demande un niveau mathématique assez élevé, c'est là je m'étais rendu compte aussi que j'avais une lacune à ce niveau-là » (S03, 143-146). Il affirme explicitement : « Je ne suis pas du genre à avoir appris la programmation par moi-même, ça a toujours été dans un cadre scolaire ou dans un but... dans un but de travail » (S03, 40-42).

Le sujet 03 affirme que la programmation est un processus de traitement des entrées pour produire une sortie : « Parce que programmer en tant que tel, c'est vraiment juste être capable de dire, selon des entrées, je fais un traitement dessus, puis j'ai une sortie » (S03,362-365). Il soutient que la programmation peut être faite individuellement ou en équipe, tout en affirmant que l'ampleur de certains logiciels est trop importante pour être gérée par une seule personne :

> Dès que t'arrives à un logiciel d'un certain... d'une certaine taille, la... le travail à faire pour... le travail nécessaire à le réaliser, c'est sûr et certain que ça va devenir trop gros pour une seule personne là. Mai, ultimement, ça peut être fait autant individuel qu'en équipe. (S03, 216-221)

Au sujet des bases de la programmation informatique, le sujet évoque l'identification des types de données :

> Si je veux manipuler des nombres, je vais utiliser soit des entiers ou des nombres flottants selon mon besoin, être capable d'identifier ça, ça ferait partie des bases. Si je veux être capable de manipuler un ensemble de données, bien là je vais utiliser un vecteur ou une liste. (S03, 100-105)

Au sujet des bases, il ajoute : « Être capable de diviser son code en fonctions... de manière à réduire un peu la redondance » (S03, 111-112). Le sujet évoque la présence de programmation dite procédurale à l'intérieur de la programmation orientée objet :

> Ça reste que quand tu fais de l'orienté objet, il y a une part de ce que tu fais qui est procédural donc à quelque part, le travail procédural que t'as... qu'on fait... n'est pas... complètement dissocié quand tu tombes en orienté objet. (S03, 460-464)

À quelques reprises, le sujet 03 illustre ses propos à l'aide de l'exemple d'un système de gestion des factures d'un commerce. Il évoque que pour créer un logiciel qui fait des factures, le programmeur doit se poser des questions afin de reproduire le modèle d'affaires :

> Si, par exemple, faut programmer un logiciel qui fait des factures, qui gère des factures, comment est-ce que je fais pour gérer mes factures, tout ça, en fonction des technologies qui me sont disponibles, comment est-ce que je fais pour... de ma... de structurer de manière logiquement mon code, de manière à être capable de reproduire le modèle d'affaires de ma gestion de facture. (S03, 257-264)

Il affirme également que ce questionnement constitue une motivation à programmer dans son cas.

Questionné à savoir si le fait de faire de la programmation modifiait sa façon de voir le monde, le sujet 03 répond par la négative. Il affirme toutefois que cela a modifié sa façon de voir le monde technologique. En utilisant un logiciel, il affirme être souvent capable de refaire les abstractions faites par le programmeur et de comprendre certaines limites ou certains choix. Il affirme également que cela l'aide à comprendre certains procédés du monde, surtout au niveau administratif : « Ça change ma vue de […] comment les procédés fonctionnent, les procédés du monde, surtout au niveau administratif, parce que tout est basé sur l'informatique » (S03, 946-950). Cette vision des choses a toutefois des limites : « On est encore toujours à réfléchir du point de vue du domaine et de la réalité et non pas au niveau des abstractions qu'on utilise » (S03, 954-956).

4.3.2 Engagement cognitif

Le sujet 03 évoque que les mathématiques ne sont pas un prérequis pour tous les domaines de l'informatique. Il affirme qu'à ses débuts dans les années 1960 à 1980, l'informatique était très près des mathématiques, mais que cela a changé :

> L'informatique, c'est assez jeune, puis quand ça l'a commencé dans les années soixante à quatre-vingts, c'était vraiment un travail de mathématicien. De mathématicien et de génie électrique qui faisait ça, mais maintenant aujourd'hui, selon le domaine dans lequel tu travailles, il y a possibilité que tu fasses de la programmation sans vraiment faire de mathématiques. Mais dans les deux cas c'est un travail vraiment qui est très logique. (S03, 503-510)

Le sujet 03 affirme que la démarche de programmation informatique implique de se questionner sur la réalité de certains domaines afin d'appliquer des méthodes informatiques. Ce questionnement engendre une prise de conscience de la complexité de choses d'apparence simples (le sujet reprend son exemple de facturation). Le sujet affirme que le programmeur doit faire preuve de créativité pour aller chercher des niveaux d'abstraction supplémentaires dans sa modélisation. La résolution de problèmes procède par des abstractions, ce que le sujet justifie en disant qu'il est très rare qu'une réalité puisse être transposée intégralement en informatique. Les abstractions peuvent être utiles pour répondre à des contraintes technologiques ou pour apporter des avantages au logiciel. Le sujet donne pour exemple la qualité d'un logiciel.

> Si tu vas un peu plus loin dans ta modélisation de ton modèle d'affaires, par exemple de manière à aller chercher des niveaux d'abstraction supplémentaires pour dissocier pour ton modèle d'affaires avec ta logique par ces abstractions-là, tu vas être capable de te retrouver avec un logiciel qui va être […] de meilleure qualité, il va être plus facile à maintenir, parce que ces niveaux d'abstraction-là vont permettre de dissocier la logique de programmation, qui elle est une chose, versus la logique d'affaires. (S03, 402-412)

Le sujet 03 soutient que l'usage de méthodes concrètes en programmation informatique sert à mieux cerner les abstractions nécessaires à la résolution de problèmes : « Souvent les méthodes plus bricoleur ou les méthodes concrètes qu'on utilise souvent ça va être des méthodes de manière à être capable de mieux cerner les abstractions qu'il faut réaliser » (S03, 612-615). Il affirme qu'il est possible de ne pas réaliser d'abstractions de niveau supérieur puis de reproduire un modèle d'affaires directement, mais remet en question la pérennité du logiciel. Toutefois, l'utilisation de ces méthodes concrètes ne signifie pas qu'il n'y a pas de processus d'abstraction à réaliser :

> En tant que tel tu vas toujours avoir une part d'analyse, ça c'est sûr. Mais par contre, il y a un certain niveau qu'il faut faire une différence entre deux niveaux d'abstraction, c'est-à-dire que tu peux régler un certain problème en y allant de manière extrêmement simple... avec un niveau d'abstraction minimum. (S02, 391-396)

En ce sens, bien que les méthodes concrètes soient possibles, l'abstraction est le cœur de la pensée informatique selon le sujet 03 :

> Je dirais […] qu'un certain niveau d'abstraction, c'est le coeur de la pensée informatique, quand tu fais de l'informatique, je pense que tu n'as pas le choix de penser avec un certain niveau d'abstraction, d'être capable de […] dissocier les technologies que tu utilises, être capable de dissocier le domaine d'affaires, et être capable de dissocier la solution au problème d'affaires puis justement être capable de relier les trois par ces abstractions-là. (S03, 595-604)

Dans cet extrait et à plusieurs reprises, le sujet 03 évoque à plusieurs reprises l'idée de problèmes d'affaires, de modèles d'affaires ou de domaines d'affaires. Il explique que le rôle du programmeur est d'abord de modéliser ce modèle et que plus le niveau d'abstraction est haut, plus la qualité du logiciel augmente. Le sujet dit que la

modélisation d'un domaine d'affaires demande de la réflexion de la part du programmeur : « Faut que tu réfléchisses justement à tes abstractions de mon domaine d'affaires qu'il faut que je modélise par exemple » (S03,371-373).

Le sujet 03 affirme que la base de la programmation est la capacité à ordonner des opérations de façon séquentielle : « Les bases de la programmation c'est vraiment être capable d'avoir cette base logique là, séquentielle, être capable de dire "cette opération-là va venir avant celle-là" » (S03, 93-96). Définir des algorithmes représente un travail de logique : « Quand tu définis des algorithmes, c'est des travaux rigoureux de logique de passer d'un point A à un point B, passer par certaines étapes logiques » (S03, 514-517). Il affirme qu'il est nécessaire de décortiquer les problèmes posés. Le sujet soutient que l'algorithmique implique plus que de communiquer des instructions à une machine et donne l'exemple d'algorithmes utilisés pour résoudre des problèmes mathématiques :

> L'algorithmique, ça rentre en jeu à toutes les fois que tu appliques une... une méthodologie de résolution de problème... c'est une certaine méthode algorithmique que tu utilises. Tu sais, quand […] il y a... ils expliquent des méthodes même en mathématiques, dans le cours de mathématiques, quand qu'ils t'apprennent des méthodes de résolution de problème, à quelque part ils t'apprennent un algorithme pour résoudre ton problème. (S03, 702-710)

Le sujet 03 nomme des méthodes de développement qu'il considère davantage comme concrètes et donne l'exemple du *test-driven development* (TDD). Il acquiesce à l'idée que l'essai-erreur est une méthode qui peut être utilisée en programmation. Enfin, le sujet 03 affirme : « "faut pas avoir peur des maths" [pour faire de la programmation], pas nécessairement, avoir peur de la logique, là tu n'as pas le choix [de ne pas avoir peur de la logique] » (S03, 510-512).

4.3.3 Engagement affectif

Le sujet 03 nomme que la programmation peut se faire en équipe ou individuellement. Il évoque des méthodes de développement comme la programmation en binôme (*pair programming*) :

> Le *pair programming* qu'on appelle là, c'est vraiment programmer à deux, sur la même machine, sur le même code, ça c'est des choses qui existent en fait... puis *pair programming*, tu sais pour en donner […] une explication, c'est il y a une personne qui elle est responsable vraiment d'écrire le code, donc […] c'est elle elle va utiliser ses efforts pour vraiment structurer le code, les lignes de code, tout ça, puis l'autre personne en arrière qui regarde son code elle va essayer... elle va focuser ses efforts à essayer de trouver les erreurs, trouver les erreurs que la personne peut faire puis elle va avoir une... elle va regarder un petit peu de plus haut niveau. (S03, 198-211)

Outre les motifs professionnels, le sujet 03 affirme aimer la rigueur exigée par la programmation informatique. Il affirme être toujours en questionnement sur la façon de résoudre un problème. Il affirme qu'il est nécessaire d'avoir une structure pour résoudre ces problèmes, et il s'agit là d'un défi qu'il aime. Il affirme aimer l'obligation

de résoudre un problème en tenant compte des contraintes appliquées par la technologie. Il a choisi son orientation professionnelle en fonction de son intérêt pour les jeux vidéos : « J'aimais ça jouer aux jeux, puis là je m'étais dit vu que j'étais au secondaire, bon bien comme choix de carrière, si j'aime jouer je vais peut-être aimer ça les développer » (S03, 274-277). Questionné sur un parallèle entre les jeux vidéos et la programmation quant aux motivations, il répond que ce n'est pas la même chose :

> Quand tu joues à un jeu vidéo, le but c'est vraiment de se rendre, tu sais, comme du point A au point B selon le jeu. Tandis qu'en programmation, c'est vraiment être capable de résoudre un problème. Tu sais, dans certains cas, les jeux ça va être de résoudre un problème, dans ce cas-là ça va être pareil, mais dans certains cas c'est juste de passer au travers […] des ennemis ou des... n'importe quoi. Puis il n'y a pas vraiment de résolution de problème. Donc ça dépendrait du jeu en fait. (S03, 278-287)

4.4 Analyse individuelle du sujet 04

4.4.1 Engagement comportemental

Le sujet 04 a été initié pour la première fois à la programmation à l'école primaire. Il a travaillé avec LOGO Writer à ce moment :

> À mon époque primaire, où ce qu'il y avait LOGO Writer, qui était une espèce […] de tortue qu'on pouvait programmer pour dessiner sur papier, donc on donnait des commandes de base sur cet appareil-là pour créer des schémas sur des feuilles. (S04, 46-51)

Il a appris à écrire du code HTML (pages Web) au milieu des années 1990 : « HTML en 1994, avec des... des Nescape Composer, ce genre d'outils là pour faire de la programmation, de l'intégration de pages Web » (S04,56-58). Il a, par la suite, appris la programmation dans sa formation au cégep :

> Par la suite au niveau de mes études en 2002, si je me rappelle bien, technique d'intégration multimédia où ce que l'a on a commencé à toucher plus la programmation en tant que tel au niveau eee conditionnel donc le javascript, on a touché au PHP, ASP aussi... donc ça c'est de 2002 à 2005. (S04, 59-64)

Il a par la suite travaillé dans une agence de publicité : « En 2005 j'ai rentré en agence de publicité où ce que là […] mon travail professionnel c'était de faire la programmation donc HTML, CSS, puis commencer à toucher au PHP, MySQL » (S04, 65-68).

Le sujet 04 raconte que ses premières tâches dans un de ses emplois étaient des tâches de programmation, par exemple développer une plateforme pour la gestion des études : « J'ai programmé cette plateforme-là et d'autres plateformes pour la gestion des études par exemple, qui étaient toutes développées en PHP MySQL » (S04, 81-83). Il affirme qu'il est passé par la suite davantage du côté du design d'interactions : « C'est comme

si je me dynamitais moi-même, je servais plus à rien. Puis c'est là que je me suis retrouvé plus en design d'interactions où est-ce que là j'ai pu amener le côté plus rationnel » (S04, 838-841).

Le sujet décrit la programmation comme une démarche de résolution de problèmes parmi d'autres : « Le code, pour moi, oui c'est un vecteur de créativité... mais ça reste juste un... une des méthodes, c'est une solution parmi tant d'autres » (S04, 565-569). Il évoque que la résolution de problème se passe toujours dans un écosystème et que, dans le cas de la programmation informatique, il y a une dimension informatique à cet écosystème :

> Comprendre l'ensemble d'un écosystème comme la programmation le fait souvent, c'est très important. C'est ce qui fait que... c'est dommage, souvent les gens vont catégoriser les informaticiens comme étant... un ordinateur, mais [cela] est beaucoup plus large au niveau de la compréhension du monde puis la représentation de... de comment que les choses interagissent aussi est beaucoup plus profonde que juste un ordinateur, mais comment comprendre une société, comment les humains se comportent. (S04, 279-288)

Questionné davantage sur cette idée d'écosystème, il établit un lien entre la programmation informatique et la psychologie humaine : « Tout ça est... en tant que tel... pourrait se traduire éventuellement en algorithmes comme on le voit présentement au niveau de modèles mental (sic) qui sont traduits... de psychologie qui sont amenés dans... dans des systèmes automatisés » (S04, 288-292). Il affirme qu'apprendre à coder n'est pas une fin en soi, l'important se situe davantage au niveau de la compréhension des interactions humaines :

> Apprendre un code, […] ça ne sert pas à grand-chose, je pense que l'importance est plus au niveau supérieur, de dire comprendre comment que les gens interagissent... puis à la limite, je pense c'est plus au niveau social et psychologique qu'on aurait quelque chose à gagner, plus que programmer. (S04, 561-566)

4.4.2 Engagement cognitif

Le sujet 04 évoque que ce qu'il aimait des mathématiques étant plus jeune, c'était la résolution de problèmes. Il affirme que cela explique pourquoi dans sa carrière il est davantage orienté vers la résolution de problèmes au niveau humain qu'au niveau de la programmation informatique. Il affirme avoir eu des cours de mathématiques au cégep qu'il n'aimait pas étant donné qu'il ne voyait pas l'utilité de ce qui était enseigné, mais qu'il a toujours aimé les mathématiques tout de même :

> J'ai eu des cours de maths au cégep que j'ai détesté parce que on ne voyait pas l'utilité de ce qui était montré... différentiel intégrales, j'ai toujours aimé les maths par contre... sans... sans vraiment comprendre pourquoi jusqu'à temps que je me rende compte que c'était une façon de résoudre des problématiques puis ce que j'aimais en tant que tel ce n'était pas tant les maths, mais plus la résolution de problèmes. (S04, 356-363)

Le sujet 04 affirme qu'un certain niveau de rationalité est nécessaire pour faire de la programmation informatique : « Je crois que la... la rationalité des gens qui font de la programmation est un... un trait de

personnalité je dirais... nécessaire pour que les gens analysent mieux les situations » (S04, 274-277). Il évoque que « les programmeurs ont souvent à prendre cette conscience-là de tout ce qui interagit à travers un système, que ce soit informatique ou de plus en plus organisationnel » (S04, 306-309).

Le sujet 04 évoque une démarche itérative dans la programmation informatique. Il évoque que les programmeurs se font des représentations mentales ou informatiques ainsi que des hypothèses qu'ils sont amenés à valider, ce qu'il qualifie d'essai-erreur. Il affirme que des frustrations peuvent venir avec cela, mais qu'une personne qui ne les accepte pas aurait de la difficulté à s'améliorer « dans la vie en général » (S04, 388). Il affirme que la vie est une « constante évolution d'essais-erreurs » (S04, 389-390) qui contribuent à nous rendre meilleurs. Le sujet affirme que l'essai-erreur survient aussi en phase d'identification du problème, pas seulement dans la résolution.

Le sujet 04 affirme que l'algorithmique consiste à « représenter une tâche ou quelque chose qu'on veut faire faire à une machine » (S04, 631-632). Selon lui, le sens d'un algorithme est en train de changer dans la mesure où la préoccupation de l'algorithmique est en train de devenir « imbriquer des comportements humains dans une machine » (S04, 634-635). Enfin, le sujet 04 évoque qu'il y a des façons plus adaptées à la communication que le code informatique, mais ne développe pas outre mesure cette idée.

4.4.3 Engagement affectif

Le sujet 04 affirme que la programmation n'est pas toujours la meilleure solution à un problème :

> Parce que une fois qu'on va avoir connu l'humain, on pourra après ça, au lieu d'enligner tout le temps vers une solution qui est... de code, on va l'enligner vers d'autres solutions où ce que là le tremplin est pris par quelqu'un d'autre qui va être soit un infographe, soit un programmeur qui va aller coder le truc qu'il faut, mais qui vont travailler en équipe pour comprendre comment que les choses doivent être appliquées. (S04, 580-588)

Le sujet 04 affirme que lorsqu'il était programmeur, il voyait le côté rationnel davantage à l'échelle de l'écosystème informatique : « Le côté rationnel que... qui était utile en programmation, je le ramène au niveau de comprendre les écosystèmes, et plus juste qu'au système numérique, mais plus humain » (S04, 494-497). Il donne l'exemple d'un projet qu'il propose à ses étudiants :

> Le projet qu'on fait, c'est l'épicerie en ligne pour personnes âgées. Bien les gens vont comprendre comment que les clients fonctionnent, avec un entrepôt, comment le service à la clientèle fonctionne, puis toutes ces données-là qu'on représente de façon schématique par des modèles de données... vont interagir ensemble, puis comment que l'information se promène dans des systèmes. (S04, 846-853)

Le sujet 04 affirme :

> Les gens souvent vont dire soit au niveau de la danse, ou l'art, je m'exprime par l'art, ou je m'exprime par le code, je me suis rarement exprimé par un code fait que... ça c'est une autre affaire que je trouve un peu bizarre. (S04, 600-604)

Il considère toutefois un aspect artistique au code au niveau de l'exploration des solutions. Il donne pour exemple des projets personnels sur lesquels il a travaillé où le produit fini n'avait pas d'utilité réelle : « L'exploration artistique je dirais... plus que l'utilisé de la chose, donc le projet concret n'était pas […] de réaliser une utilité par rapport à ce qu'on faisait, mais plus d'essayer des choses, donc plus le côté exploratoire des arts » (S04, 177-181).

Le sujet 04 décrit sa motivation principale au niveau de la résolution de problèmes. La programmation est une façon différente d'autres pour résoudre des problèmes :

> C'était de la découverte, je pense que c'est beaucoup la relation problématique solution qui m'a toujours animé, puis je trouve que ça peut résoudre beaucoup des problématiques, des fois, cette solution-là, la programmation étant une des technologies qu'on peut utiliser pour résoudre des problématiques. (S04, 192-197)

Dans cet extrait, le participant discute de résolution de problèmes. Nous avons cherché à comprendre la place qu'il accorde à la programmation informatique dans la résolution de problèmes. Il affirme qu'il a réalisé au cours des dernières années que la résolution d'un symptôme du problème ne signifie pas nécessairement la résolution du problème :

> Je n'ai plus d'intérêt à faire de la programmation, puis c'est ce qui fait que je ne fais plus de contrats de pige... avec la programmation. Ici, je ne fais plus de développement non plus, bien pour des outils internes, bien je trouve ça redondant, souvent la résolution […] du symptôme n'est pas la résolution du problème. Donc, à ce moment-là, ça ne sert à rien d'essayer de développer un système si les gens, [par] exemple, ne l'adoptent pas. (S04, 370-378)

Quant à ses motivations à faire de la programmation informatique, elles ont évolué dans le temps. Lorsqu'il a commencé la programmation, sa motivation à faire des sites Web était de communiquer en publiant des choses en ligne : « HTML c'était pour faire un site Web, pour communiquer avec les autres, donc c'était vraiment mon... mon but avec ça, de se faire une page personnelle, une présence en ligne pour pouvoir communiquer puis publier des choses en ligne » (S04, 123-127). Par la suite, à cette dimension communicationnelle s'est ajoutée une dimension professionnelle étant donné qu'il a travaillé comme programmeur.

Finalement, au sujet des réactions émotives et des frustrations pouvant être ressenties en faisant de la programmation, le sujet 04 affirme : « C'est juste une question de patience, de volonté, de détermination... donc c'est souvent des traits qu'on va retrouver chez les programmeurs, des gens qui... qui vont continuer à chercher la solution, à essayer » (S04, 403-407).

4.5 Analyse individuelle de la sujet 05

4.5.1 Engagement comportemental

La[5] sujet a été initiée à la programmation informatique en secondaire 1. Elle évoque avoir programmé dans le logiciel Microsoft Excel d'abord : « À l'école on a appris Tap'Touche en secondaire 1, puis... on a appris à programmer dans Excel tout ça » (S05, 36-38). Elle a poursuivi son apprentissage en contexte scolaire au cours de son secondaire : « Jusqu'en secondaire 5 où est-ce qu'on a appris le VB, le Visual Basic, donc le VB 5... puis c'est ça j'ai développé... j'ai commencé à développer mes compétences en programmation dès le secondaire » (S05, 38-42). Elle a programmé avec MatLab pour du travail mathématique au cégep, puis avec d'autres langages à l'université : « [au] cégep bien là on a fait du MatLab, puis du Maple, donc programmation plus niveau mathématiques, rendue à l'université je suis entrée en maths pures, donc j'ai fait des cours introduction à la programmation, puis de algorithmes programmation » (S05, 42-46). À l'université, dans le cadre de sa maîtrise et de son doctorat, elle travaille avec de la « programmation plus graphique au niveau des... Mindstorms et des... LEGO WeDo et beaucoup beaucoup Scratch » (S05, 54-56).

La sujet distingue deux types de programmation et affirme avoir travaillé avec les deux : » Je fais les deux types de programmation, programmation textuelle ou graphique » (S05, 273-274). Elle donne pour exemple Scratch pour Arduino, Scratch sur Raspberry Pi, l'interface de LEGO Mindstorms. Sur l'accessibilité de la technologie en général, elle se prononce en affirmant que « la technologie c'est facile parce qu'on l'a rendue facile » et que « la technologie qu'on veut accessible à tous, on l'a rendue facile » (S05, 784-785). Selon elle, la créativité fait partie intégrante de la programmation informatique. Sa définition de la programmation informatique implique au minimum des structures conditionnelles (elle donne l'exemple de cellules conditionnelles dans Excel). Elle parle de créer une solution à un problème ouvert : « En quelque sorte tu vas créer... une solution à ton problème. De la même façon que quand tu as un problème ouvert, tu vas créer une solution à ton problème » (S05, 673-675). C'est ce qui explique la place de la créativité selon elle. Finalement elle donne pour exemple la programmation d'un appareil d'enregistrement pour enregistrer une émission, affirmant qu'il s'agit d'une programmation très simple et très dirigée : « Programmer le vidéo pour qu'il enregistre, ma grand-mère elle ne sait pas programmer, mais elle sait enregistrer son émission préférée le dimanche matin, […] ça revient à quand même programmer » (S05, 350-353).

[5] Tel que discuté en introduction, il a été choisi d'employer le terme épicène « sujet » plutôt que « participant ». Lorsque cette personne est une femme, nous avons considéré le terme comme féminin, voilà ce qui justifie l'emploi de l'article « la ».

La sujet évoque l'essai *Program or Be Programmed* (Rushkoff, 2010) affirmant qu'il relate que pour « modeler le futur, faut que tu sois capable de le comprendre actuellement, et actuellement le futur est en développement technologique » (S05, 127-129).

La sujet donne des exemples issus de son travail d'enseignante. Par exemple, en demandant un travail à ses étudiants et étudiantes, elle leur donne carte blanche sur la façon de représenter la chose : « Bien si pour toi la meilleure façon de présenter ton intention c'est avec un PowerPoint, vas-y » (S05, 900-901). Elle évoque le développement de compétences. Elle donne l'exemple de Microsoft Word pour affirmer qu'un logiciel peut être très complexe au point où il est difficile pour une seule personne de le maîtriser entièrement : « Tu peux pousser Word très très loin, qui t'amène peut-être à éventuellement devoir programmer pour vraiment l'utiliser à son maximum, encore là même moi je ne l'utilise pas à son maximum » (S05, 195-198). Elle discute de l'évolution des jeux vidéos pour illustrer que la technologie est parfois rendue trop facile :

> Quand tu jouais au NES, à Mario, puis que tu commençais, tu appuyais sur Start tu arrivais au premier niveau puis c'est "débrouille-toi". Maintenant, tu mets Assassin's Creed, puis ça te prend une heure et demie de tutoriel. On a rendu ça trop facile, pour justement effectivement éviter qu'on ait à réfléchir, puis c'est ça le problème, parce qu'on ne va pas plus loin, on veut pas aller plus loin, parce qu'on ne ressent pas le besoin d'aller plus loin, on n'a pas besoin de réfléchir, parce que justement, ils ont déjà réfléchi pour nous autres. (S05, 805-814)

4.5.2 Engagement cognitif

La sujet évoque qu'il ne faut pas avoir peur des mathématiques, bien qu'elles ne soient pas impliquées dans toutes les formes de programmation. Elle affirme :

> [Il ne] faut pas avoir peur [des mathématiques]. Les mathématiques ne sont pas impliquées dans toutes formes de programmation, c'est les méthodes de résolution de problèmes qui sont impliquées dans la programmation. Tu peux avoir peur des maths mais être capable de programmer en Scratch. (S05, 483-487)

La sujet affirme que la résolution de problèmes est un processus abstrait menant à une solution concrète : « C'est le lien abstrait concret qui est toujours là entre ce que tu fais et le résultat que tu obtiens. Donc ton résultat est souvent concret par rapport à une résolution de problème abstraite » (S05,628-631). Les niveaux d'abstraction en informatique varient selon le domaine de travail : « Si t'es en intelligence artificielle puis que tu conçois un logiciel d'apprentissage automatisé, bien ton niveau d'abstraction est pas mal plus élevé que quelqu'un qui veut faire bouger son chat du point A au point B avec Scratch » (S05, 425-429).

La sujet 05 décrit l'algorithmique en disant : « Tu dis à tes chiffres en quelque sorte quoi faire pour arriver à la solution » (S05, 715-716). Elle donne un exemple : « Par exemple je donne un projet de yogourt où est-ce que

là... avec le Arduino, ils branchent une sonde de température, bien là faut qu'ils calculent... faut que la température soit maintenue entre 42 et 44 degrés » (S05, 491-495).

La sujet questionne la place de la programmation dans la communication entre humains : « On a les bases... la littérature, la littératie, la mathématique puis la littératie numérique pour communiquer, donc je ne dis pas nécessairement que […] c'est une base, ça fait partie des bases qui permettent de communiquer à l'ère... aujourd'hui » (S05,684-688). Elle distingue la communication entre individus de la communication entre un individu et une machine : » Tu veux parler de communiquer avec une machine? Avec un autre être humain? » (S05, 689-690). La littératie numérique n'est selon elle pas la plus indiquée pour la communication entre humains : « Parce que avec un autre être humain, [je ne suis] pas sûre que tu vas considérer la littératie numérique en premier, donc ce n'est peut-être pas ça la base » (S05, 690-692).

La sujet donne l'exemple d'un projet de construction d'une grue avec LEGO Mindstorms où elle a observé des gens travailler par essai-erreur afin d'observer le comportement de blocs de programmation : « J'ai fonctionné par essai-erreur, des fois je… j'essaie… bon, si je fais telle chose, qu'est-ce que ça fait? Si je fais telle chose, qu'est-ce que ça fait? » (S05, 616-618).

4.5.3 Engagement affectif

Tout d'abord, à plusieurs reprises au cours de l'entretien, la sujet 05 décrit une forme de collaboration avec d'autres personnes qui s'apparente à celle qui a lieu dans une communauté de pratique. Par exemple, elle évoque aller chercher des informations sur Internet lorsqu'elle fait de la programmation informatique :

> Oui je vais aller chercher des... des trucs sur Internet, par exemple je vais aller beaucoup sur des forums par exemple pour le Arduino, je n'ai pas le choix parce que je ne le connais pas du tout, puis je l'apprends au fur et à mesure. (S05, 96-99)

Nous considérons que son propos reprend la distinction entre la création et la consommation. À ce sujet, elle évoque l'existence d'interfaces informatiques de plus en plus faciles à utiliser. Elle évoque que l'utilisateur est parfois dirigé par le logiciel qu'il utilise et donne un exemple :

> Est-ce que tu veux être dirigé par le logiciel ou diriger le logiciel? C'est ça l'esprit de Program of Be Programmed. […] [L'auteur, Douglas Rushkoff] expliquait : […] Facebook, c'est quoi? […] tu demandes à un enfant, l'enfant il va dire bien 'ça permet de faire des amis'. Non, Facebook ça te permet de monitorer tes relations avec d'autres personnes, à la base Facebook est ça. (S05, 141-148)

Dans une optique de création plutôt que de consommation, la sujet évoque que la programmation informatique mène à des réalisations qui sont des sources de gratification pour elle : « Devant un problème puis que ça ne marche pas, puis que tu finis par trouver la solution, ça c'est le fun, c'est gratifiant » (S05, 467-469). Ses

motivations ont varié avec le temps. Au secondaire, lorsqu'elle a débuté la programmation informatique, la note obtenue était une motivation, alors qu'aujourd'hui elle le fait par intérêt personnel : « c'est sûr au départ la motivation c'est surtout la note, tu sais quand c'est dans... tu es dans un contexte scolaire, l'objectif c'est de réussir l'examen, réussir le travail qui est demandé » (S05, 114-117). Elle est motivée par l'enseignement de la programmation informatique : « J'aime ça enseigner la programmation à des futurs enseignants ou à des élèves, mais plus une programmation basic du style Scratch » (S05, 89-91).

4.6 Analyse individuelle du sujet 06

4.6.1 Engagement comportemental

Le sujet 06 a commencé la programmation informatique alors qu'il était au secondaire, mais pas dans un contexte scolaire :

> Un ami avait une espèce de petit ordinateur tout petit avec juste une ligne, un écran d'une ligne, […] sur lequel il y avait des jeux... des jeux de bonhomme pendu, des jeux de mots, […] des jeux comme ça, puis il y avait aussi, pour une raison qui m'échappe encore pourquoi c'était sur un petit ordinateur comme ça pour enfants, mais il y avait de la programmation qui était disponible. On pouvait programmer en Basic. (S06, 47-55)

Ses premières expériences en informatique étaient motivées par le désir de comprendre comment les choses fonctionnent; il nomme en ce sens un IBM 286 : « J'ai commencé à entrer ces lignes de codes là sans comprendre ce que... ce que j'écrivais, mais en suivant... la... en essayant de comprendre la logique, éventuellement je m'amusais […] à modifier un peu ces programmes-là » (S06, 58-62). Au début de son secondaire, il savait vouloir s'orienter en informatique :

> Déjà début secondaire, je voulais aller en informatique, j'avais déjà regardé les programmes de cégep, puis je voulais déjà aller en informatique, je voulais m'en venir à Québec, parce qu'il y avait un programme DEC-Bac, donc déjà là... à ce moment-là, j'avais déjà fait mon choix. (S06, 105-110)

Il affirme qu'au début il ne comprenait pas le code qu'il entrait. Ses premiers programmes étaient des jeux :

> Au courant de mon secondaire, j'ai commencé à programmer dans des langages un […] un peu plus modernes puis j'ai commencé à faire des jeux, donc c'est surtout à programmer des jeux, la plupart des gens commencent comme ça à essayer de faire des petits programmes qui vont faire quelque chose d'amusant ou quelque chose d'intéressant. (S06, 97-104)

Le sujet parle de la programmation informatique comme d'un outil de résolution de problèmes :

> Les gens qui travaillent dans des compagnies où ils programment, bien ils résolvent des problèmes, ils utilisent la programmation comme un outil pour résoudre leurs problèmes, puis ils font ça de manière professionnelle. Donc c'est comme des gens qui passent leur journée à réfléchir à […] comment je vais résoudre mon problème, puis qui mettent sous forme de code […]. La capacité à mettre en ordre leurs idées et à déterminer ça va être quoi la […] séquence d'instructions qui va résoudre le problème puis de le mettre en lignes de code. (S06,484-494)

Il affirme que certains concepts informatiques peuvent aider à mieux résoudre certains problèmes : « [les cours théoriques d'informatique m'ont] permis vraiment de comprendre plus en profondeur la théorie des langages, ou même la théorie de la complexité, qu'est-ce qui fait qu'un programme est plus efficace qu'un autre? » (S06, 270-273). Il affirme que lorsqu'il a commencé à travailler dans le domaine de l'intelligence artificielle :

> Là, la programmation est devenue un peu plus un outil, bien en tout cas le côté plus professionnel de la chose, c'est devenu plus un outil pour faire des expérimentations, implémenter des algorithmes d'intelligence artificielle, regarder comment ils performent, les exécuter sur diverses tâches. (S06, 278-283)

La programmation peut être utilisée de façons différentes selon que l'on souhaite réaliser un prototype ou une solution optimale : « À partir de la maîtrise, doctorat, c'est le python qui a pris toute la place, donc qui est de plus haut niveau que les langages comme le C, le C++, mais qui permet de faire du prototypage extrêmement rapidement » (S06, 294-298).

Réfléchir comme un informaticien n'implique pas nécessairement d'être capable de faire de la programmation informatique selon le sujet 06. Il donne l'exemple d'un cuisinier qui conçoit une recette :

> Un programme informatique, tout ce que ça fait, c'est suivre la recette que le programmeur a écrite. Mais le programmeur, il a inventé la recette, il y avait un problème, puis il a été cherché les éléments, les morceaux dont il avait besoin pour résoudre son problème puis il a écrit la recette. Donc on pourrait même dire qu'un chef cuisinier réfléchit comme un informaticien, son but c'est de faire un repas... qui aille certaines propriétés, puis là il va penser à ses affaires, il va penser à tel ingrédient, il va penser aux quantités, il va donc réfléchir d'avance puis, avec des expérimentations, finir par faire le produit qu'il veut faire […] donc il a réfléchi comme un informaticien. (S06, 498-510)

Le sujet dresse une analogie entre la culture numérique d'aujourd'hui et la culture littéraire médiévale en rapportant un exemple issu de l'essai Program or Be Programmed (Rushkoff, 2010) : alors qu'une population analphabète médiévale était dépendante d'un rapporteur de nouvelles pour s'informer, la population analphabète numérique d'aujourd'hui est dépendante des programmeurs et programmeuses qui créent les outils que nous utilisons :

> Il y avait un crieur public qui lisait les nouvelles ou qui donnait les nouvelles aux gens dans la... dans la rue puis c'est cette personne... cette personne-là était capable de lire ce qui était écrit sur le papier puis elle devait le communiquer aux autres, puis les autres étaient dépendants de cette personne-là pour avoir l'information. (S06, 899-905)

Pour justifier son idée que tout le monde peut apprendre à programmer, le sujet dresse une comparaison avec l'apprentissage d'un instrument de musique :

> Est-ce que tout le monde peut jouer de la musique? La réponse serait peut-être probablement un peu la même, il y en a qui vont prendre un instrument, […] ils vont l'apprendre en deux jours puis ils vont être capables de jouer, puis il y en a qui vont travailler extrêmement fort puis ils vont finir par réussir mais ça va être plus long. (S06, 464-470)

Le sujet 06 illustre ses propos par des exemples de son parcours. Par exemple, il a réalisé sa maîtrise et son doctorat sur le thème de l'intelligence artificielle. Il affiche des préoccupations de citoyenneté numérique en donnant l'exemple de Facebook et de la collecte de données personnelles :

> Si, il y a 15 ans, on avait dit aux gens "dans 15 ans vous allez vous-mêmes donner toutes vos photos personnelles de votre famille à une grosse compagnie, puis vous allez gratuitement aller annoter qui est sur la photo, c'est les visages de qui, ça c'est mon enfant, ça c'est ma mère, mon père [...]". (S06, 1102-1107)

4.6.2 Engagement cognitif

Le sujet 06 a complété une maîtrise et un doctorat en informatique. Il affirme que la recherche en informatique se rapproche davantage des mathématiques et qu'à un certain niveau il s'agit peut-être « plus des probabilités statistiques que de l'informatique » (S06, 383-384). Cela doit être compris à partir de son sujet de recherche qui concerne l'apprentissage machine en intelligence artificielle. Il affirme le caractère interdisciplinaire des mathématiques, donnant l'exemple de cours d'histoire, de géographie ou de sciences qui font intervenir des chiffres dans l'étude d'objets non mathématiques :

> Plus tard au primaire, quand on commence à avoir des cours d'histoire, des cours de géographie, des cours de plein de sujets en sciences, on lit des textes, donc on développe la langue, on a des chiffres qu'on doit mettre en relation, donc on développe les mathématiques, tout en apprenant des choses qui sont intéressantes puis qui sont dans un autre sujet. (S06, 992-999)

Le sujet 06 affirme aussi qu'il n'est pas essentiel d'utiliser ou de connaître les mathématiques qui supportent l'informatique. Plus précisément, le sujet 06 affirme que le niveau d'abstraction varie d'un langage à l'autre. Il nomme des langages de bas niveau (C, C++) expliquant que « eux généralement demandent de réinventer la roue à chaque fois qu'on veut faire quelque chose » (S06, 606-607). Pour les langages de haut niveau (e.g. Java, Python), il affirme : « On ne réinvente pas la roue, on réutilise ce qui est disponible à chaque fois » (S06, 617-618). En début de projet, le sujet 06 affirme qu'il y a un niveau d'abstraction qui consiste à réfléchir à ce qu'on veut que l'application fasse :

> Il y a un certain niveau d'abstraction plus élevé où est-ce que je vais tout mettre de côté les détails techniques, je vais penser au design de mon application, design graphique, puis je ne réfléchis même pas à comment je vais le faire à... comment je vais le programmer, je réfléchis juste à ce que je veux que ça aille l'air puis qu'est-ce que je veux que ça fasse. (S06, 524-531)

Questionné sur la place de l'essai-erreur dans la programmation informatique, le sujet affirme que « généralement l'essai-erreur n'est pas aléatoire, on apprend de l'erreur puis on construit là-dessus » (S06, 832-833), ce qui s'apparente davantage au concept d'approximation successive. Il ajoute :

> On essaie une solution, ça ne fonctionne pas, on essaie de comprendre pourquoi ça ne fonctionne pas puis là ça nous donne une information de plus pour résoudre notre problème, puis on avance comme ça pour résoudre notre problème tranquillement pas vite vers quelque chose qui marche. (S06, 840-845)

Le sujet évoque l'existence de différents types de programmeurs ou de personnes, certains étant davantage bricoleurs alors que d'autres préférant débuter par une analyse de haut niveau. Le sujet 06 pense qu'il est impossible de résoudre un problème seulement par essais-erreurs : « Je ne pense pas qu'on puisse résoudre un problème global que par de l'essai-erreur » (S06, 810-812).

Le sujet 06 donne l'exemple d'un étudiant universitaire à qui il a enseigné qui n'a jamais réussi à réaliser un des travaux pratiques qu'il qualifie de simple :

> [Un étudiant] qui a passé vingt heures, dix ou vingt heures je m'en rappelle plus, à essayer de résoudre le problème de trouver un chiffre puis dire à l'utilisateur 'est-ce qu'il est plus petit ou il est plus grand?', mettre tout ça dans une boucle, ce qui est un programme assez simple, mais lui... cet étudiant-là je pense qu'il a fait de l'essai-erreur pendant vingt heures. (S06, 813-819)

Il s'agit d'un programme informatique qui sélectionne un nombre aléatoire, et l'utilisateur doit le deviner à l'aide des indices *plus haut* ou *plus bas* fourni par le programme. Au sujet de cet étudiant, il ajoute :

> Il n'était pas en mesure de le faire à ce moment-là, donc par essai-erreur, il a essayé des choses qui étaient très très près de la solution, mais ça ne marchait pas... puis il ne savait pas comment corriger son erreur. (S06, 825-829)

Le sujet 06 affirme : « La logique, il ne faut certainement pas en avoir peur parce que c'est ce qu'on fait 100% du temps » (S06, 655-657).

4.6.3 Engagement affectif

Le sujet 06 affirme que souvent, un programmeur peut être amené à adopter plusieurs rôles :

> Ça va arriver souvent qu'un informaticien […] va avoir plusieurs chapeaux, un chapeau dans lequel il va plus réfléchir globalement à son application. Puis, après ça, il va changer de chapeau puis il va dire "ok là il faut que je l'optimise, il faut que je regarde comment je vais résoudre les problèmes à plus bas niveau". (S06, 548-554)

Dans ce cas, il est autant responsable de la vision globale du projet que de l'implémentation des détails techniques. Cela peut être un avantage étant donné que, selon lui, certains détails techniques peuvent avoir une incidence sur la vision globale du projet : « C'est vraiment [une] plus-value de... d'être capable de... de connaître ou de savoir comment les choses vont être faites à l'intérieur parce que parfois, ces détails-là sont importants pour la vue globale » (S06, 568-562). Selon le sujet 06 affirme que ce découpage des rôles peut s'observer dans plusieurs domaines : « Ça c'est probablement dans tous les domaines, il y a des gens qui sont juste intéressés à faire ce qu'on leur demande puis ils ne veulent pas nécessairement y réfléchir au plus haut niveau » (S06, 579-583).

Le sujet 06 affirme que les utilisateurs d'applications comme Facebook devraient être conscients qu'il y a une compagnie derrière avec des objectifs. Cette conscience amènerait peut-être un changement dans la façon d'utiliser ces applications :

> Si on utilise ces applications-là sans comprendre que il y a une compagnie derrière, puisque cette compagnie-là rend ce service-là... avec un but derrière, qui est de faire de l'argent, toutes les compagnies existent pour faire de l'argent sauf les organismes de bienfaisance... ils réfléchiraient peut-être autrement puis ils feraient peut-être attention. (S06, 1126-1132)

Le sujet 06 évoque que certains problèmes en programmation informatique peuvent être réglés d'une façon optimale et standardisée par la communauté, il donne l'exemple de la manipulation d'une liste avec Python : « Les gens ils ont posé exactement la même question qu'on s'est posée parce que chaque jour on se pose des millions de questions que des millions de personnes se sont posées » (S06, 760-763). La recherche sur Internet permet de trouver des façons optimales de résoudre un problème et s'avère plus efficace et moins coûteuse que de réfléchir à résoudre ce problème soi-même.

Dans une estimation spontanée, le sujet affirme que « 90% de la vie d'un logiciel et du temps qu'on met dedans, c'est le débogage puis de tout résoudre les problèmes qu'on n'avait pas pensé » (S06,679-682). Selon lui, cela peut représenter des sources de frustrations et il faut être prêt à y faire face.

Le sujet 06 affirme qu'à ses débuts, ses motivations à faire de la programmation informatique étaient de s'amuser et de comprendre comment les choses fonctionnaient. Il note un changement dans l'évolution technologique depuis son enfance affirmant qu'il est de moins en moins utile de comprendre afin d'utiliser :

> Si je regarde les générations un peu plus jeunes que moi [pour] qui tout était plus facile, donc les gens qui aimaient ça, ils essayaient de comprendre comment ça marchait […]. Il y a des gens qui étaient curieux de nature, qui essayaient de comprendre comment ça marchait aussi, mais on n'a pas besoin de comprendre pour utiliser. Puis plus ça va, moins on a besoin de comprendre pour utiliser. Mais je pense que c'est important d'éduquer les jeunes pour qu'ils apprennent ou qu'ils aient le goût de comprendre comment ça fonctionne. (S06, 1176-1186)

4.7 Analyse individuelle du sujet 07

4.7.1 Engagement comportemental

Le sujet 07 a commencé à programmer en 1985 alors qu'il était âgé de 10 ans. Il affirme, en riant, n'avoir compris ce qu'il faisait que quelques années plus tard :

> J'ai commencé à apprendre à programmer en 1985. J'avais dix ans, premier ordinateur. À l'époque, les ordinateurs familiaux, c'était le tout début. Puis j'ai commencé à comprendre ce que je faisais vers 1991-1992 […] où, là, changement d'ordinateur, cette fois-ci avec un disque dur, un vrai écran et tout ça. (S07, 34-40)

À cette époque, il programmait en Basic, puis en assembleur, et ensuite en Pascal. Pendant ses études, il a appris le langage C de façon autonome et il a par la suite choisi de s'orienter en informatique. Il a fait un doctorat en cryptographie et travaille encore dans ce domaine aujourd'hui. Il affirme que la cryptographie se trouve à la frontière entre les mathématiques et l'informatique.

Le sujet 07 énonce que la constitution des langages de programmation est compréhensible dans la mesure où l'on comprend ce qui les sous-tend :

> Si on connaît l'assembleur qui est traduit en dessous, tout d'un coup le C devient logique, et l'assembleur devient logique si on connaît le langage machine qui est en dessous, qui lui est logique si on sait comment fonctionne un circuit avec des portes logiques. (S07, 530-535).

Pour le sujet 07, cette connaissance est essentielle : « [sinon] ça devient simplement […] un aspect magique disons, là le processeur fait sa magie, bien le processeur étant magique, bien il n'est plus prédictible » (S07, 524-526). Pour le sujet 07, la tâche de programmation implique essentiellement de taper du code, et il se positionne contre la segmentation des rôles voulant que l'analyste n'écrive pas de code :

> Il y a des architectes, des fonctionnels, des organiques et tout ça puis ça tombe sous l'analyste, c'est qu'à la fin ça tombe sur les développeurs qui ont juste besoin de suivre les instructions comme des singes en quelque sorte. Et qui sont souvent, socialement parlant, au bas de l'échelle. Alors, bien ça, ça permet d'avoir effectivement des énormes structures où on paye plein de gens, on obtient un code qui est en général de qualité assez douteuse. (S07, 566-574)

Selon lui, un bon programmeur ne laisse rien en plan, ne laisse pas de « bouts incomplets » (S07, 642). Il parle d'un « mélange de patience et d'obstination forcenée » (S07, 643-644). Le sujet évoque une préoccupation de pérennité dans la construction de logiciels.

Le sujet 07 décrit des communautés Open Source (logiciel libre) auxquelles il contribue. Le succès de ces communautés tient notamment au fait, selon lui, qu'un ordinateur est accessible et que tout le monde peut en avoir un. Dans ces communautés, il évoque qu'il n'y a pas de « spécialiste de la politique au sens platonicien du terme » (S07, 241-242), c'est-à-dire que tous les membres de ces communautés font de la programmation et aucun ne se consacre exclusivement à l'administration de la communauté. Cela donne, selon lui, que « en développement informatique, en fait, chaque développeur, dans son carré de sable en quelque sorte, a assez de compétences pour avoir un avis intéressant » (S07, 247-250).

Selon le sujet 07, la tâche de programmation implique de « penser dans les deux sens et à tous les niveaux à chaque fois » (S07, 560). Il exprime ainsi que le programmeur doit avoir une idée du « but ultime » (S07, 553-554), de la vision globale, alors que le travail d'écriture de code se fait pourtant dans le sens inverse. Il parle de processus « top-down » (S07, 551) et « bottom-up » (S07, 552) qui sont réalisés simultanément :

> On a aussi […] une autre échelle comme ça, […] tout ce qu'ils appellent le développement bottom-up et top-down où top-down ça veut dire on réfléchit dans les grandes lignes [à] la tâche qu'on veut accomplir puis on la découpe en sous-tâches, en morceaux, jusqu'à arriver à des tâches élémentaires qui sont ce que le programmeur va taper. En bottom-up, ça veut dire on commence par les tâches élémentaires et quand on en a suffisamment, on les assemble pour fabriquer jusqu'à arriver […] au but. Et pour bien programmer, il faut penser top-down, et programmer bottom-up en quelque sorte. (S07, 541-552)

Le sujet 07 donne en exemple le domaine de la sécurité informatique dans lequel il travaille pour expliquer qu'il est insuffisant qu'un programme informatique fonctionne dans des conditions normales; il doit aussi fonctionner dans des conditions anormales et « éventuellement que quelqu'un a forcées pour être anormales » (S07, 283-284).

Le sujet 07 décrit le profil type d'un programmeur à partir d'une métaphore, celle d'Achille. Achille est un héros, un meurtrier qui « fait un massacre effroyable et c'est tout ce qu'il sait faire, fondamentalement » (S07, 642-643). Mais « dans tout ce qui est raconté dans l'Iliade, on voit d'abord qu'on ne peut pas le manager, en gros c'est un très bon guerrier, mais on ne peut pas le commander » (S07, 644-647). Le sujet soutient que, un peu à la manière d'Achille, il faut amadouer le programmeur et « quand il s'y met, il y va vraiment à fond donc il est extrêmement persistant dans tout ce qu'il veut » (S07, 648-650).

Le sujet 07 a raconté plusieurs anecdotes. Il évoque avoir fait l'école normale supérieure à Paris, là où les concepteurs du langage Caml ont étudié :

Quand j'ai fait mes études, en fait, c'était à l'école normale supérieure à Paris et les concepteurs de Caml sont passés par là aussi donc... Et les gens qui enseignaient la programmation le faisaient en Caml, c'était le milieu des camélistes, tous les douze d'entre eux étaient là. (S07, 379-384)

Le sujet affirme que dans le cadre de son travail, il lit beaucoup de code. Il raconte qu'au cours de la fin de semaine précédant notre entretien, il travaillait sur un algorithme de cryptographie dans le cadre d'une collaboration avec des chercheurs.

Le sujet raconte son passage dans un organisme public alors qu'il ne portait pas le titre de développeur bien que son travail consistait en du développement informatique : « ils m'ont envoyé à [organisme public] un peu comme homme à tout faire, […] je n'étais pas officiellement un développeur, c'est un grade particulier, ils sont très jaloux de leurs choses, donc je faisais officiellement des scripts » (S07, 75-79).

Le sujet 07 propose une réflexion au sujet de l'utilisation de Google :

> Google, je constate […] qu'il y a eu un changement dans l'utilisation de l'ordinateur […]. En particulier de tout ce qui est relié à Internet dans notamment des buts d'apprentissage […]. Fin années quatre-vingt-dix, quand on faisait des recherches, quand on manipulait l'ordinateur, on allait voir sur Google, on lançait des recherches, des mots-clés pour aller chercher de l'information. Ça correspondait à ce qu'on faisait avant, avant quand on voulait savoir quelque chose, on allait à la bibliothèque, avec ses petits, je l'ai fait. Maintenant, la plupart des gens que j'observe, surtout ceux qui ont moins de 25 ans, ne font pas de recherche, ils ne savent pas très bien faire des recherches. Ils reçoivent beaucoup d'information, ils sont connectés à des réseaux, genre Twitter, Facebook ou d'autres réseaux sociaux, où ils ont beaucoup de connaissances qui les abreuvent d'énormément d'information et ce qu'ils en font essentiellement, en fait l'essentiel de leur temps ne consiste pas à aller chercher de l'information, mais à trier ce qu'ils reçoivent, à couper des morceaux. […] Du coup, quand on leur demande d'aller chercher des choses en disant "bon, sur tel sujet, hop, va me chercher ça sur Google", ils sont en général assez perdus, ils ne sont pas efficaces. (S07, 1009-1033)

Le sujet 07 évoque qu'à ses débuts en informatique, enfant, il ne comprenait pas tout à fait ce qu'il faisait et certaines choses lui semblaient obscures. C'est le cas du comptage en base 2, par exemple.

4.7.2 Engagement cognitif

Le sujet 07 affirme que certaines notions mathématiques sont incontournables, comme par exemple la manipulation des nombres entiers. Il affirme qu'il s'agit de l'algèbre, mais pas d'analyse. Il affirme que l'ordinateur est une machine à calculer et que les mathématiques permettent d'analyser l'ordinateur. Il évoque que « les équations dérivées partielles, ce genre de choses, ce sont des mathématiques qui servent assez peu en informatique » (S07, 684-686). L'algèbre est selon lui incontournable. Il évoque le cas de « millions de sites Web qui tournent par des gens qui ne sauraient pas ce qu'est un corps fini » (S07, 47) pour illustrer la place des mathématiques dans le domaine du Web. Il affirme que son domaine, la cryptographie, est « à la frontière entre informatique et mathématiques » (S07, 690-692).

Le sujet 07 affirme qu'il est difficile de résoudre un problème informatique exclusivement par la pensée « parce qu'il faut plus ou moins se faire tourner son propre ordinateur dans son cerveau, et le cerveau humain il n'est pas très doué pour ça, c'est pour ça qu'on a les ordinateurs » (S07, 834-837). Il évoque qu'en réfléchissant à un problème, il y a un risque d'oublier un aspect du problème qui « au moment de le traduire en quelque chose de tangible va se révéler à ce moment-là » (S07, 840-841). Il expose ensuite son souhait que tout architecte logiciel essaie lui-même de programmer des choses afin d'avoir confiance « qu'il a bien vu toutes les facettes du problème » (S07, 843-844).

Le sujet 07 expose son idéal de la programmation informatique voulant que pour réaliser un bon programme il faille reprendre le travail par trois fois :

> Pour faire bien un programme, il faut le faire trois fois. La première fois, c'est pour comprendre le problème, puis après on l'efface. Après, on le fait une deuxième fois pour comprendre la solution. Et une fois qu'on arrive au bout, là on comprend la solution, donc on sait comment on aurait dû l'écrire. Donc on efface, et là on l'écrit correctement. (S07, 737-743)

Le sujet 07 énonce que l'algorithmique est le travail du cerveau humain et non de l'ordinateur. En outre, il évoque l'existence de la méthode essai-erreur en programmation informatique, mais affirme que « ce n'est pas professionnel, on ne devrait pas travailler comme ça » (S07, 731-732). Au sujet de son recours à l'essai-erreur, il affirme : « C'est effectivement pour affiner ma compréhension de ce qui se passe » (S07, 733-734).

Comme d'autres sujets, le sujet 07 discute du lien entre les mathématiques, la logique et la programmation informatique. Selon lui, la logique et les mathématiques sont aidantes pour faire de la programmation informatique :

> L'informatique, c'est quelque chose qui marche mieux quand on a un bagage de logique et de mathématiques en dessous. Et logique, ça vient notamment par la structuration... structure des phrases, donc la grammaire. C'est par là qu'on aborde la logique […] quand on est petit. Sujet, action, complément. Et l'algorithmique va se construire par-dessus. (S07, 945-951)

4.7.3 Engagement affectif

Le sujet 07 affirme que pour avoir une équipe cohésive qui travaille sur un même projet, il faut que les membres de l'équipe soient tous motivés par le projet et que chacun respecte les compétences des autres. Il affirme, en riant, que pour les programmeurs, « ce n'est pas gagné » (S07, 327). Il affirme que les équipes où le travail est divisé entre analyste, architectes et développeurs donnent du code de mauvaise qualité. Il raconte avoir déjà travaillé comme architecte et considère que « c'est essentiellement du vent » (S07, 592-593). Le sujet fait état d'une observation voulant que plus un individu a de l'expérience, plus il s'éloigne de l'écriture du code, alors que ce devrait être le contraire selon lui. L'idéal, selon lui, est de garder un projet sous le contrôle d'une seule personne. Il reconnaît que ce n'est pas possible pour répondre à certains besoins.

Le sujet 07 aborde la question des communautés de programmation en affirmant que les membres ont tendance à pointer le ridicule des autres. Il compare cela à la « fabrication d'une meute qui simplement ne se tape pas dessus entre eux, mais seulement sur les autres » (S07, 313-315). Il donne en exemple des communautés Open Source dans lesquels il y a « une sorte de chef spirituel que personne ne conteste et qui a le dernier mot quoi qu'il advienne » (S07, 218-220). Il affirme qu'il existe aussi quelques projets dirigés de façon communautaire et donne l'exemple du logiciel VLC. Selon lui, il est plus difficile et plus rare d'avoir des projets dirigés par la communauté.

Le sujet 07 utilise le concept de dette technique pour expliquer que plus un code est gros et travaillé par plusieurs personnes, plus il est difficile à maintenir. Le sujet 07 relate une expérience de travail où il a dû travailler avec des contraintes extérieures, par exemple produire un système embarqué[6] de terminaux de paiement.

Le sujet 07 propose que la programmation informatique en soi est une activité créatrice et qu'elle peut aussi être le support à une autre activité créatrice (par exemple, les films d'animation). Il affirme :

> Quand je fais du développement à titre personnel, c'est tout […] en code open source […], je serai jamais payé là-dessus, mais il y a une part de démonstration, pour montrer aux autres comment c'est fait. Donc en quelque sorte, il s'agit pour moi de démontrer ma virtuosité, donc c'est un peu comme faire une peinture. Il faut que ce soit beau. Il faut que ça en mette plein les yeux. (S07, 189-196)

Il utilise le concept d'élégance pour expliquer que certaines solutions sont « plaisantes à l'oeil » (S07, 187-188), affirmant que cela a un côté artistique. Cela peut être associé à ses motivations lorsqu'il a débuté à programmer. Il raconte que, plus jeune, il était fasciné de faire bouger des choses. Il affirme qu'aujourd'hui il considère « le développement [informatique] comme une sorte d'expression artistique » (S07, 179-180). Il affirme que peu de gens autour de lui s'intéressaient à l'informatique à l'âge où il a débuté (à partir de 10 ans).

4.8 Analyse individuelle de la sujet 08

4.8.1 Engagement comportemental

La sujet 08 a été initiée à la programmation vers l'âge de 12 ans. Sa mère l'avait inscrite à un cours d'informatique le samedi matin où elle a appris LOGO. Par la suite, elle a poursuivi l'apprentissage de façon autonome et se rappelle avoir appris à programmer en Basic avec un VIC-20. Elle a ensuite réalisé une technique en

[6] Un système embarqué est un système informatique indépendant et autonome installé dans un appareil ou un objet qui a une vocation qui n'est pas seulement informatique, par exemple : un avion, un terminal de paiement, une station météo. Ces systèmes viennent avec des contraintes propres aux milieux dans lesquels ils sont déployés.

informatique puis un baccalauréat en informatique. Elle affirme préférer l'analyse à la programmation informatique : « […] j'ai programmé de l'orienté objet, mais après ça j'ai tombé dans l'analyse, puis je préférais plus l'analyse » (S08, 120-122). Elle raconte avoir depuis cessé de programmer pour se consacrer professionnellement à l'analyse, exclusivement.

La sujet 08 compare le passage de la programmation fonctionnelle vers la programmation orientée objet au passage entre une image à deux dimensions vers une image à trois dimensions :

> [La programmation orientée objet] c'est une autre façon de programmer, c'est vraiment une autre façon d'architecturer tous les services. À l'époque, ce n'était pas des services, c'était des fonctions puis des méthodes, mais c'était des objets au lieu […] que ce soit des tables et des fonctions. Tu sais, c'était vraiment plus... quand que tu tombes en 2D à 3D, c'était plus comme ça que moi je le voyais. (S08, 168-174)

Elle évoque que la programmation informatique implique plusieurs rôles en plus de l'écriture du code et affirme faire de la gestion de projets. La programmation informatique lui a permis de développer davantage de repères dans l'utilisation de logiciels. Elle donne l'exemple de la recherche sur Google. Elle connaît certaines fonctionnalités de plus que sa mère :

> Si j'étais une personne comme ma mère, je recherche [sur] Google d'une façon, et si je suis comme moi, je recherche d'une autre façon. Il y a des paramètres dans la barre de recherche qu'on a appris à découvrir. (S08, 618-622)

La sujet 08 compare sa compétence en informatique à celle d'un mécanicien en affirmant que dans les deux cas, cela influence leur point de vue par rapport à l'objet de travail. Alors que le mécanicien n'aura pas les mêmes critères pour choisir une voiture que quelqu'un qui n'a pas d'expertise en mécanique automobile, elle n'a pas les mêmes critères pour choisir un logiciel que quelqu'un qui n'a pas d'expertise en programmation informatique :

> Ce n'est pas les mêmes critères quand tu es mécanicien, même d'une personne à l'autre tu n'as pas les mêmes critères, moi mon critère c'est je suis capable de remorquer une roulotte, parce que moi je veux remorquer une roulotte, mais le mécanicien lui c'est je ne veux pas mettre trop de temps dessus, faut qu'il soit fiable. (S08, 681-686)

La sujet 08 a travaillé dans différents contextes, ce qui l'a amenée à travailler avec plusieurs langages. Elle relate avoir travaillé avec Rex et Cobol : « Quand j'ai commencé chez [compagnie] je faisais du Rex, je programmais en Rex, puis Rex c'est comme l'ancêtre du Cobol » (S08, 154-156). Elle raconte avoir travaillé à l'élaboration d'un système d'inscription à des cours pour une université. Elle a ensuite interagi avec ce système en tant qu'utilisatrice devant s'inscrire à des cours.

La sujet raconte une anecdote où elle a dû travailler avec Cobol pour faire communiquer entre elles deux plateformes, alors qu'elle était employée comme développeuse Web. Elle raconte cela comme une expérience professionnelle angoissante :

> J'ai une expérience vraiment concrète, j'étais consultante pour [compagnie] puis on m'avait mis à [organisme public], puis je programmais une passerelle entre la plateforme intermédiaire puis centrale. Donc [il] fallait que je fasse parler le COBOL avec du Web. Puis moi j'étais programmeuse Web seulement et il y avait quelqu'un qui faisait le COBOL, mais on avait de la misère à se rejoindre entre les deux puis c'était ça qu'il fallait qu'on fasse, ça j'ai vraiment travaillé fort pour être capable de passer des données images du COBOL à l'autre puis de Web à... c'était épouvantable, j'ai « rushé ». (S08, 351-362)

Elle raconte avoir travaillé le soir alors qu'elle n'était pas payée afin de réussir cette tâche.

4.8.2 Engagement cognitif

La sujet 08 affirme que la place des mathématiques dépend de ce qui est à programmer. Elle donne l'exemple d'un travail scolaire qu'elle a dû réaliser lors de sa technique où elle devait simuler le fonctionnement d'une pompe à essence :

> On simulait la pompe, la pompe quand qu'on va mettre du gaz, tu sais les chiffres puis tout, les taxes, puis tu sais... pour programmer ça je n'avais pas besoin vraiment de mathématiques, à part être capable de multiplier les litres [et le prix par litre], mais tu sais ce n'était pas très poussé en mathématiques. (S08, 520-525)

La sujet évoque que des notions de grammaire peuvent être logiques, mais soutient que la logique est davantage associée aux mathématiques : « Lire, écrire, ce n'est pas très... il y a des règles de grammaire qui peuvent être logiques, mais je trouve que la notion logique, mathématiques, vient avec programmation, mathématiques » (S08, 501-505). Selon elle, l'aspect logique est plus présent que l'aspect mathématique (notons qu'elle ne les oppose pas). Elle affirme que certains programmes peuvent nécessiter davantage de mathématiques : « Par contre, si j'avais à faire un écart-type puis des statistiques sur je ne sais pas moi, le nombre de personnes qui échouent en français, là à ce moment-là ça irait me chercher plus de la mathématique » (S08, 525-528).

En relatant avoir participé à la création d'un système d'inscription à des cours pour une université, la sujet affirme : « Au début tu avances, c'est gris, puis après ça tu poses tes blocs, puis là tes blocs sont bien en place puis là tu es capable de voir au travers tes blocs » (S08, 297-300). En outre, elle acquiesce à l'énoncé voulant que l'algorithmique consiste à communiquer des instructions à une machine. Elle ajoute : « C'est communiquer des instructions ou se communiquer des instructions pour se le décrire » (S08, 469-470). Enfin, elle reconnaît que l'essai-erreur peut se situer au niveau de la preuve de concept. Elle affirme que l'essai-erreur peut se situer à petite échelle ou à plus grande échelle :

> L'idée des essais-erreurs ça peut aussi se traduire en preuve de concept. Est-ce que ce […] ce que j'ai pensé est réaliste? Est-ce que ça se fait? Ou, finalement, "ah, ça ne se fait pas". Des fois, essai-erreur ça peut être un tout petit peu, à toute petite échelle, comme un bogue j'essaie des affaires puis je corrige. Ou bien c'est "j'ai une pensée, est-ce que c'est faisable, est-ce que on est dans le champ avant de continuer", je le réalise plus en preuve de concept. (S08, 428-436)

La sujet 08 accorde de l'importance à l'apprentissage de la logique : « Il faut que les jeunes aussi soient logiques » (S08, 472-473). Elle affirme que la logique est davantage associée aux mathématiques, bien que des règles de grammaire puissent être logiques.

4.8.3 Engagement affectif

La sujet 08 affirme avoir la conviction qu'un programmeur peut faire de l'analyse et vice-versa. Elle évoque son milieu de travail où ces deux appellations désignent des fonctions différentes, mais elle maintient que les deux, programmeur comme analyste, pourraient occuper la fonction de l'autre.

La sujet 08 affirme que les programmeurs ont parfois tendance à ne pas penser à l'utilisateur et à concevoir ce qui relève de la « bébelle d'informatique » (S08, 592). Maintenant qu'elle travaille comme analyste plutôt que programmeur, elle explique que son rôle consiste à « se mettre dans la peau de l'utilisateur » (S08, 589). Elle raconte qu'à ses débuts en programmation informatique, elle était davantage axée sur le résultat et n'était pas bonne pour l'analyse.

La sujet 08 affirme que lorsqu'elle faisait de la programmation informatique [maintenant elle fait de l'analyse], elle vivait beaucoup de frustrations en raison de problèmes techniques qu'elle ne parvenait pas à résoudre. Elle évoque le manque de support et la charge de travail que cela représentait. Au sujet du passage vers l'analyse plutôt que la programmation, elle affirme : « Enfin! Je ne fais plus de cauchemars sur mon code » (S08, 346-347).

La sujet 08 raconte qu'elle a commencé à s'intéresser à l'informatique alors qu'elle ne pratiquait pas de sport, l'informatique étant apparue comme une activité intéressante pour « travailler ses méninges » (S08, 99). À ses débuts, sa motivation était de voir le résultat concret : « Bien, c'est voir le résultat concret que je faisais, des carrés, des ronds, "ah!", j'avais un pop-up, j'avais un message, c'était juste quelque chose que je tapais puis j'étais capable de voir le résultat » (S08, 82-85). Elle évoque l'association entre ce qu'elle tapait et le résultat qu'elle obtenait. Il s'agit ici d'une forme de rétroaction immédiate par rapport à une action réalisée par la personne. Dans ce cas-ci, la sujet a tiré du plaisir de cette rétroaction.

4.9 Analyse individuelle du sujet 09

4.9.1 Engagement comportemental

Le sujet 09 a débuté la programmation informatique alors qu'il était âgé de 11 ans. Jusqu'à son entrée à l'université, il a appris la programmation à l'extérieur de l'école, par lui-même :

> [le site sur lequel j'ai appris à programmer] c'est le site du zéro là, […] c'est un site de tutoriels faciles d'approche qui existait dans le temps, puis comme les gens de mon âge le connaissent souvent, souvent ils ont commencé par les sites Web sur ce tutoriel-là, ou la programmation en C. Fait que j'ai commencé en faisant des sites Web en PHP puis la programmation en C avec... des petits tutoriels sur le même site est venue par la suite. (S09, 51-58)

Il affirme que c'était un passe-temps et qu'il n'en faisait parfois pas pendant plusieurs mois pour y revenir par la suite. À l'université, il a débuté un baccalauréat intégré en mathématiques et informatique puis a par la suite changé pour un baccalauréat en informatique. Il réalise en ce moment une maîtrise en informatique.

Il affirme qu'avec l'expérience, un programmeur apprend à mieux interagir avec les langages de programmation, ce qui donne lieu à une meilleure anticipation des erreurs pouvant survenir à la compilation. Il ajoute que la programmation informatique n'implique pas seulement que l'écriture du code, elle implique aussi de « être capable de sous-diviser ce programme-là parce que sinon il devient pas intelligible puis ça devient impossible de faire des projets de plus grande envergure » (S09, 270-272).

Nous avons questionné le sujet sur la transversalité des apprentissages en programmation informatique. Selon lui, sa connaissance de la programmation induit des changements dans la façon dont il utilise des applications. Pour illustrer cela, il donne l'exemple de l'application Snapchat :

> Snapchat, j'ai jamais vu ça, mais tu sais je n'aurais pas besoin de demander à mon ami comment mettre [un] filtre, je connais […] les entrées utilisateurs qui sont disponibles à un programmeur Android, parce que j'ai déjà "gossé" avec des applications Android par le passé. Fait que là, je suis capable de "ok, bien là peut-être que"… tu sais, dans l'espace des touches possibles, j'ai le *swipe*, j'ai le *double tap*, je me doute que dans leur API, c'est ces options-là qui sont offertes aux programmeurs, fait que tu sais je vais être capable d'apprendre à utiliser Snapchat rapidement même si je l'ai jamais vue avant. (S09, 594-605)

Il donne un exemple similaire pour le logiciel Microsoft Word disant que lorsqu'il cherche quelque chose, il a de « bonnes heuristiques pour trouver le bon menu rapidement » (S09, 609-610). Il ajoute que lorsqu'il répare le téléphone de sa mère, cette dernière a l'impression qu'il fait de la magie. Il parle de « l'aspect mystique » (S09, 226) de la programmation dans la population générale. Il termine en racontant utiliser la programmation de deux façons dans le cadre de sa maîtrise : comme moyen pour réaliser des analyses comme dans toute science, mais aussi comme moyen pour construire son objet de recherche.

4.9.2 Engagement cognitif

Le sujet 09 affirme que les cours de mathématiques qu'il a suivis n'étaient pas nécessaires pour faire de la programmation Web :

> Est-ce que le cours des mathématiques était préalable à la programmation Web? Pas du tout. Il y a moyen d'arriver aux mêmes conclusions sans passer par la logique abstraite, tu sais. [On] pourrait te l'enseigner [la logique] comme une technique de programmation. (S09, 408-412)

Il raconte que parmi les étudiants au baccalauréat en informatique, certains ont de la difficulté à faire les liens entre les cours de mathématiques pour informaticien et l'informatique :

> Par exemple au sein du bac, il y a deux écoles de pensée, il y a beaucoup de gens qui n'aiment pas leurs cours de mathématiques, un cours qui s'appelle mathématiques pour l'informaticien finalement, puis là ils arrivent là, ce cours-là ne parle d'aucune technologie, il n'y a pas d'interaction nécessaire avec l'ordinateur, tu sais tu pourrais passer ce cours-là en écrivant tes preuves sur des feuilles, c'est un cours de maths finalement. Puis il y a des gens qui ont de la difficulté à faire le lien entre ça puis... le programme qu'ils font en programmant. (S09, 364-374)

Le sujet 09 affirme que l'abstraction est présente dans toutes les formes de programmation et qu'il est inopportun de la considérer davantage dans la programmation orientée objet. Il ajoute que « ce n'est pas parce que tu as l'abstraction que tu as un objet » (S09, 296-297). Il affirme que l'abstraction est présente en programmation informatique peu importe le paradigme et croit qu'il ne faut pas associer l'abstraction au paradigme objet plus qu'à un autre :

> Il y a la programmation fonctionnelle qui est un autre paradigme qui n'implique pas une programmation orientée objet puis qui te permet de faire cette abstraction-là, parce que là bien au lieu d'avoir un objet avec des méthodes, ton abstraction se fait en termes de fonctions. Tu as une fonction qui s'appelle, je ne sais pas, "télécharger image", qui abstrait le transfert des bits d'un serveur à un autre. Puis ça n'a pas besoin d'être intégré à un objet pour être abstrait. Je ne pense pas qu'il faut connecter ces deux notions-là. (S09, 311-320)

Nous considérons que cette façon d'envisager l'abstraction correspond davantage à la modélisation d'un problème qu'au processus d'abstraction réfléchissante. Toutefois, la façon par laquelle il décrit cette modélisation nous amène à considérer que l'abstraction réfléchissante est la dimension cognitive de cette modélisation. Par exemple, il affirme que la programmation implique d'être « capable de voir qu'une sous-tâche particulière peut être abstraite finalement » (S09, 279-280). Il parle ensuite de diviser une tâche « dans ta tête » (S09, 283-284) puis d'être capable d'en faire l'implémentation.

Le sujet 09 affirme : « Le compilateur te donne le output en fait c'est le meilleur outil pour apprendre à programmer » (S09, 818-820). Il compare le compilateur à un enseignant qui donne une rétroaction pour t'indiquer que quelque chose n'est pas correct. Il parle d'un essai-erreur possible « d'un point de vue large »

(S09, 482) en donnant l'exemple du *test-driven development*. Il comprend l'essai-erreur comme une démarche aléatoire où il s'agit de « copier-coller des bouts de code que tu as pris un peu partout sur internet […] jusqu'à ce que la sortie te semble correcte » (S09, 496-498). Il poursuit sa réflexion en affirmant que si un programmeur se considère exclusivement comme un bricoleur et juge la pensée abstraite inutile, il aura de la difficulté à programmer en équipe « parce que le code que tu vas produire ne sera pas de bonne qualité, il ne sera pas lisible » (S09, 435-436). Il qualifie le bricolage en programmation informatique d'artisanat, affirmant qu'il s'agit effectivement d'un aspect de la programmation, mais que la programmation ne doit pas se réduire à cela. Il donne un exemple de condition logique utile à la programmation informatique, l'inversion d'une condition :

> L'inversion d'une condition. Admettons que tu avais une branche dans ton programme qui disait "si tel cas fait ça", puis ensuite tu as un "et". Fait que si tu as ça, "et ça", exécute telle opération. Maintenant, pour inverser ça, admettons tu décides que finalement tu veux faire le cas inverse, tu as besoin de notion de logique pour savoir que pour inverser ça, tu as besoin de remplacer le "et" par un "ou". Ça c'est un exemple de logique typique qu'on t'enseigne juste à l'université. (S09, 393-402)

4.9.3 Engagement affectif

Le sujet 09 affirme qu'en travaillant en équipe, il faut expliquer ce qu'on fait aux autres :

> Bien quand tu travailles en équipe ou en groupe, il y a toute la partie d'expliquer ce que tu as fait, qui n'est peut-être pas une tâche mais une compétence qui est super connexe. Quand t'écris pour toi-même, ce n'est pas trop grave, mais quand tu travailles avec d'autres, c'est important que les autres puissent comprendre ce que tu as fait. (S09, 188-193)

Il évoque l'importance de la documentation puis affirme une tendance dans la communauté de programmation pour « écrire du code qui est assez clair pour que quelqu'un d'autre qui a un bagage similaire puisse le lire puis le comprendre puis l'expliquer aussi » (S09, 195-197). Il ajoute qu'à ses débuts en programmation informatique, un aspect « vraiment cool » (S09, 161) était de publier son site Web sur Internet et d'inviter quelqu'un d'autre à taper l'adresse pour aller le visiter. Il affirme :

> Jusqu'à un certain point, c'est vrai que toutes les applications ont été « designées ». Puis ça c'est quelque chose aussi qui est important de garder en tête […]. Puis là c'est plus quand j'apprends des choses à des personnes plus vieilles ou qui sont moins habituées […], elles vont avoir peur de briser l'ordi. C'est souvent la première chose que tu entends, les personnes n'osent pas aller fouiller dans les menus parce qu'elles ont peur de faire une opération qui va genre causer un crash ou... mais là quand tu te souviens que ces applications-là ou les programmes ont été créés par des gens qui avaient des êtres humains en tête, bien là tu te dis ça ne devrait pas être possible pour moi de briser mon téléphone en utilisant Snapchat. (S09, 714-727)

Le sujet 09 utilise le concept d'espace de solution pour décrire la marge de manœuvre en programmation informatique. Ces propos peuvent être associés au concept de marge créative défini dans le cadre théorique. Il

affirme qu'il y a plusieurs approches pour résoudre un problème et que certaines approches sont meilleures que d'autres :

> Souvent l'espace de solutions est assez grand, il y a plusieurs approches. Puis il y a une espèce de forme d'artisanat dans le sens que... toutes les approches ne sont pas aussi bonnes, puis il y en a qui sont plus lisibles par exemple ou plus compréhensibles que d'autres, fait que là ce qui est plaisant c'est d'explorer toutes ces solutions-là puis de choisir celle que tu juges qui est la meilleure. (S09, 137-143)

Le sujet 09 est d'accord pour dire que « le code est un vecteur de créativité » (S09, 519) justement en raison de l'exploration possible des solutions. Il affirme que parfois, une bonne solution est originale. Le sujet parle de la « communauté de programmation » (S09, 304) pour valoriser certaines pratiques plutôt que d'autres.

Au moment de discuter des retombées de l'erreur en programmation informatique, le sujet 09 explique:

> Au début c'est super frustrant, puis ce l'est de moins en moins. J'ai parlé du compilateur, maintenant quand le compilateur trouve quelque chose, c'est une bonne nouvelle en fait, tu finis par le réaliser, parce que tu préfères... bien c'est moins risqué que ce soit le compilateur qui dise que tu as fait une erreur que l'erreur se glisse dans ton code puis que tu la retrouves plus tard. (S09, 811-817)

Il affirme que l'activité de programmation peut donner lieu à des frustrations, mais se questionner à savoir si ces frustrations sont particulières à la programmation informatique. Il affirme : « Faut que tu apprennes à le [le compilateur] considérer comme un outil, faut vraiment apprendre à se détacher des erreurs de compilation, tu sais tout le monde en a, tout le monde en fait » (S09, 824-827). Il questionne la frustration en programmation informatique en se demandant si elle est particulière à cette activité. Il dresse une comparaison en racontant avoir déjà construit une boîte de bois et avoir été frustré d'utiliser un clou trop long. Il affirme que « c'est une frustration aussi, comme dans n'importe quel artisanat » (S09,806, 806-807).

Enfin, le sujet 09 évoque plusieurs motivations dont certaines se sont développées avec l'expérience. Au départ, la programmation était un hobby personnel. Il évoque la satisfaction de transformer un problème en apparence insolvable en une solution qui s'exécute automatiquement par un ordinateur. Il parle de la programmation de sites Web disant que c'est « une porte d'accès vers d'autres types de programmation » (S09, 170). Aujourd'hui, il programme de façon professionnelle dans le cadre de sa recherche à la maîtrise. Sur ses débuts en programmation, il affirme qu'il avait du plaisir à reproduire les sites Web qu'il visitait. Il parle de « faire passer le site internet de quelque chose de statique que tu consommais à quelque chose que tu pouvais créer aussi » (S09, pp. 158-160).

4.10 Analyse individuelle du sujet 10

4.10.1 Engagement comportemental

Le sujet 10 a appris la programmation de façon autodidacte et situe cela vers le début du secondaire. Il l'a apprise à l'extérieur de l'école. Il a réalisé principalement de la programmation de sites Web, mais pas exclusivement. Certains de ces sites Web ont été utilisés en contexte professionnel. Il raconte ses débuts en programmation informatique ainsi :

> J'ai commencé à faire de la programmation de façon autodidacte à partir du début de mon secondaire. Je travaillais sur des sites Web au départ, avec l'ancien logiciel qui s'appelait Front Page puis je me suis mis tranquillement à aller modifier le code du site. (S10, 36-40)

Il a ensuite appris le langage PHP qu'il utilisait via un ordinateur Linux et son propre serveur. Il mentionne avoir exploré les langages Perl et C++ :

> À ce moment-là je faisais beaucoup de programmation pour la découvrir, j'ai fait un peu de Perl, j'ai jamais vraiment retouché à ça. C++ j'ai essayé de l'apprendre, avant le cégep où j'ai eu mon cours, mais j'arrêtais parce que ça ne me servait à rien. (S10, 164-168)

Nous avons questionné le sujet 10 sur ce qui unit ces différentes formes ou langages de programmation afin de mieux saisir son engagement comportemental. Il soutient que la programmation implique au minimum des structures conditionnelles. En ce sens, il ne considère pas le HTML comme de la programmation informatique étant donné que « il n'y a pas de logique dans le HTML » (S10, 508). Il parle du langage HTML comme d'une « construction sans plan » (S10, 499). Dans le cas de la programmation Web, cela peut se traduire par au minimum une section dynamique (changeante) sur une page. Il donne l'exemple de l'intégration automatique d'un en-tête sur une page. Il affirme que le langage PHP sert à générer du code HTML :

> Souvent le PHP c'est des conditions qui vont donner du HTML différent, selon […] si la condition est remplie ou pas. Mais c'est certain que moi faut que je détermine... faut que je l'écrive le HTML de chacune de ces conditions-là. Mais c'est sûr que c'est PHP qui va décider lequel va être affiché […] selon […] ma programmation. PHP n'a pas de volonté propre, c'est le programmeur qui décide. S'il est bon, il devrait savoir ce qui va s'afficher au bout du compte! (S10, 542-550)

Le sujet 10 évoque des différences de complexité entre les langages de programmation et donne l'exemple de la déclaration de variables en C++ qui requière de la rigueur quant au type et à la taille des données :

> On est averti lorsqu'on commence à l'apprendre, il faut allouer la mémoire à chaque variable. On ne peut pas faire ce qu'on veut après avec ça. C'est sûr que les langages avec lesquels je travaillais, surtout PHP, étaient très très flexibles par rapport à ça. Puis C++ sert à faire des logiciels, il faut interagir avec l'ordinateur, avec le système d'exploitation, tandis que PHP si tu as le même serveur, peu importe le client tu vas toujours avoir le même résultat, ce qui apporte une certaine sécurité, quand on est programmeur on veut que le résultat soit toujours le même. (S10, 190-200)

Questionné sur la place de la créativité dans la programmation informatique, le sujet 10 affirme que plusieurs solutions peuvent mener au même résultat : « Il y a plusieurs solutions aussi pour arriver au même résultat, bien c'est le cas lorsqu'on résout une équation différentielle ou intégrale aussi, est-ce que les intégrales c'est de la créativité? C'est un peu la même situation » (S10, 736-740). Enfin, il raconte avoir créé un site Web pour sa famille lorsqu'il était plus jeune. La programmation était alors « Un outil pour procéder de façon efficace » (S10, 717-718). Le site Web servait par exemple à diffuser des photos.

4.10.2 Engagement cognitif

Le sujet 10 identifie l'algèbre comme un préalable à la programmation informatique. Le concept de variable est incontournable selon lui : « Le concept de variable en fait, avant d'arriver en mathématiques d'algèbre, on ne sait pas ce que c'est une lettre pour définir... pour correspondre à une valeur, c'est là qu'on l'apprend finalement. [Cela] fait que avant ça, c'est difficile » (S10, 446-450). Il identifie d'autres éléments essentiels à la programmation informatique : « La priorité des opérations c'est essentiel, les boucles c'est quand même quelque chose qu'il faut, l'incrémentation de variables, tout ça, […] avant d'avoir fait de l'algèbre à l'école, je ne vois pas très bien comment on peut faire de la programmation » (S10, 385-389).

Le sujet 10 affirme que l'abstraction ne peut être que le cœur de la pensée informatique. Il s'appuie sur l'idée selon laquelle la programmation informatique consiste en la construction d'objets intangibles et que, dès lors, tout ce qu'elle construit est abstrait : « Bricoleurs sur quoi? Sur quelque chose qui n'existe... qui n'est pas matériel, donc ça c'est de l'abstraction » (S10, 582-584). Il affirme que « il n'y a aucune action humaine qui ne nécessite pas de réflexion » (S10, 616-617). Il affirme :

> Lorsqu'on programme, forcément on veut traiter de l'information ou rendre de l'information telle qu'elle n'était pas auparavant. [Il] faut avoir une pensée transformatrice, […] travailler sur un objet qui n'existe pas, c'est de l'abstraction il n'y a pas d'autre chose à dire que ça! (S10, 576-581)

Le sujet 10 affirme qu'il lui arrive de faire de l'essai-erreur et que cela l'amène à constater ce qui ne fonctionne pas puis à améliorer le code. C'est ainsi qu'il distingue la programmation comme compétence professionnelle de l'algorithmique comme compétence « de réflexion, de traitement » (S10, 275-276). Il parle d'un minimum de deux niveaux d'abstraction, celui des « intrants » (S10, 291) à savoir de l'information disponible, et celui de la résolution du problème. Il termine en affirmant que l'essai-erreur est « dangereux parce qu'on ne saura pas

comment on est arrivé au résultat » (S10, 627-628). Il affirme que dans ce cas-là, il est difficile de « garantir l'intégrité du résultat » (S10, 630).

4.10.3 Engagement affectif

Le sujet 10 affirme ne jamais avoir fait de la programmation de façon professionnelle. Il dit être au fait que cela implique des contraintes comme la gestion et le travail d'équipe. Il affirme que « il faut que le programmeur soit capable de rendre un résultat sans savoir comment il va être exploité après » (S10, 342-344). Il évoque certaines contraintes extérieures qui ne s'appliquent pas dans le cas de programmation amateur, pour soi-même et sans but professionnel. Il donne l'exemple de la sécurité informatique.

Le sujet 10 s'en remet à une définition juridique anglo-saxonne de la créativité et affirme : « Au sens anglo-saxon du droit d'auteur, la créativité c'est le travail, un travail original » (S10, 704-705). Il affirme qu'en ce sens la programmation informatique est une création. Il doute toutefois que la programmation puisse être envisagée comme une expression « artistique ou abstraite » (S10, 708). Il doute que quelqu'un puisse s'exprimer par la programmation informatique : « Est-ce qu'on peut s'exprimer par la programmation? J'en doute » (S10, 709-710). Il ajoute que « on peut trouver un code beau par contre, on peut trouver un code élégant » (S10, 729-730).

Enfin, le sujet 10 affirme que la patience est propre à chaque individu et donne un exemple personnel :

> Ça, c'est éminemment personnel la patience. Moi j'ai eu beaucoup de frustrations parce que je ne réussissais pas à trouver un problème que j'avais dans mon code où j'avais des niveaux de difficulté, d'abstraction qui étaient trop élevés, donc je me couchais à 4h du matin pendant plusieurs semaines. (S10, 395-400)

C'est ainsi qu'à ses débuts, il a travaillé avec un logiciel de conception de pages Web, Front Page. Il a ensuite commencé à rédiger son propre code lorsqu'il s'est senti contraint par le logiciel. Il crée encore des sites Web pour des organismes qu'il supporte. Il ajoute qu'à ses débuts, son frère s'était mis à créer lui aussi des sites Web. Sur ce propos, il ajoute : « Je dois dire qu'à cette époque-là, ça m'agaçait un peu » (S10, 113-114). Il affirme qu'il considérait cette activité comme sa « chasse gardée » (S10, 123). À ce sujet, il se rappelle qu'un oncle lui avait dit « [la connaissance] on peut la cumuler sans perte pour autrui » (S10, 127-128).

4.11 Analyse individuelle du sujet 11

4.11.1 Engagement comportemental

Le sujet 11 a commencé à faire de la programmation informatique alors qu'il était en secondaire 3. Il a d'abord appris de façon autonome grâce à des tutoriels en ligne :

> Il y avait un site Web ça s'appelait le site du zéro, « asteur » ça a un autre nom, mais dans le fond ce site Web là il y avait plein de tutoriels sur le développement, bien la programmation, puis dans le fond j'avais commencé un tutoriel sur le PHP. Puis […] je n'étais pas capable de faire un programme correctement en PHP au complet, mais je comprenais une certaine base de logique donc quand je suis arrivé au cégep […], c'est là vraiment j'ai compris comment programmer comme il faut. (S11, 62-72)

Nous avons demandé au sujet 11 d'élaborer sur l'apprentissage via des tutoriels en ligne, et il considère que cette méthode a des limites : « Avec les tutoriels c'était tout le temps un peu flou, puis peu de pratique en tant que tel » (S11, 72-73). Il a ensuite fait un baccalauréat en informatique puis une maîtrise, et réalise en ce moment un doctorat. Il affirme avoir eu au secondaire des cours d'informatique dont un cours optionnel où il réalisait un peu de programmation (Visual Basic).

Nous avons tenté de mettre en lumière la définition de la programmation informatique du sujet 11. Selon lui, la programmation informatique n'implique pas seulement l'écriture de code :

> Premièrement faut que tu saches comment spécifier ton programme, qui est plus soit... on appelle ça de l'analyse fonctionnelle, dans le fond tu vas dire "mon programme je veux qu'il fasse telle telle affaire". Puis, tu sais, faut que ce soit vraiment assez spécifié en général. (S11, 353-357)

Par la suite, différents outils peuvent servir à développer cette analyse sous forme de programme. En plus des langages de programmation qu'il a utilisés pour faire de la programmation Web et dans le cadre de ses recherches, il considère que le logiciel Microsoft Excel permet de faire de la programmation, par exemple via l'utilisation des formules. Il utilise cet exemple pour illustrer son propos que l'apprentissage pour tous de la programmation permettrait à tous d'automatiser davantage de choses dans leur travail. De façon générale, pour l'utilisation ou la création d'applications, il affirme qu'il aime les logiciels libres, car « un logiciel open source ne pas t'enfermer dans un écosystème parce que tu peux toujours prendre ce logiciel-là puis l'adapter à ce que tu veux faire » (S11, 995-997).

Dans tous les cas, la programmation débute par la décision de ce que le programme doit faire. Il évoque que cette spécification doit donner lieu à un découpage du programme. Il affirme que chaque paradigme implique le découpage d'un programme de différentes façons :

> Est-ce que genre au lieu de découper en classes, bien tu [découpes] en modules puis en fonctions? Puis ça c'est un peu semblable, c'est ton module en tant que tel [qui] est semblable à ton orienté objet. Puis sinon tu as d'autres paradigmes de programmation comme la programmation fonctionnelle, où est-ce que tu vas découper... où est-ce que en tant que tel ton programme... bien ça ressemble un peu au paradigme […] procédural dans le sens où […] tu découpes ton programme en fonctions, puis souvent en modules. (S11,467-477)

Au sujet de ce découpage, il affirme :

> Souvent, tu vas découper ton programme en classes, […] chaque classe va avoir sa responsabilité en tant que telle. Dans le fond, tu ne veux pas que la responsabilité de deux classes s'entrecoupe ou... Ouin, tu ne veux pas que la responsabilité de deux classes s'entrecoupe mais elles doivent quand même communiquer entre elles pour faire ce que tu veux que ton programme fasse. (S11, 375-382)

Le sujet 11 raconte avoir fait un stage dans un organisme public, affirmant avoir travaillé principalement avec Java. Dans ce contexte, il ne réalisait pas de programmation bas niveau et affirme que « Java souvent est une abstraction au bas niveau » (S11, 415-416). Selon lui, la programmation de bas niveau devient nécessaire lorsque le langage utilisé ne permet pas de réaliser les optimisations nécessaires. En ce sens, il donne pour exemple des algorithmes d'apprentissage machine :

> Admettons si je voulais programmer ma propre couche, là j'aurais probablement à programmer, à faire du code pour le processeur graphique en tant que tel. Fait que ce serait une partie de code que j'aurais à faire qui serait plus bas niveau en tant que tel. (S11, 432-437)

Enfin, il donne l'exemple d'un de ses amis qui a programmé un compteur de personnes à l'aide d'un dispositif Raspberry Pi pour le gymnase qu'il fréquente. Avec l'accord de la direction, il l'a installé à l'entrée du gymnase : « puis dans le fond ça permettait de compter le nombre de personnes en ce moment dans le gym, puis il avait fait un site Web où est-ce qu'il affichait le compte du nombre de personnes dans le gym » (S11, 649-652).

4.11.2 Engagement cognitif

Le sujet 11 discute de la place des mathématiques dans la programmation informatique. Selon lui, les mathématiques ne sont pas toujours utiles en programmation informatique. Il raconte que dans le baccalauréat en informatique, il y a beaucoup de cours de mathématiques en début de parcours et que certains étudiants n'aiment pas ces cours. Il donne l'exemple d'une de ses amies qui a choisi de changer de parcours pour aller étudier en systèmes d'information organisationnels, et elle travaille aujourd'hui comme programmeuse dans un organisme public. Il affirme la logique est nécessaire en informatique, peu importe le type de programmation :

> La plupart des gens qui travaillent, que je connais qui travaillent en informatique, ils ne font pas vraiment de mathématiques dans leur... dans leur quotidien admettons. Fait que c'est pour ça que mathématiques est plus ou moins bon [dans l'énoncé], puis... mais par contre la logique c'est vrai. (S11, 524-529)

Cette implication de la logique dans la programmation informatique se manifeste entre autres au niveau de l'analyse fonctionnelle. À cet égard, le sujet 11 affirme que prendre une décision en programmation informatique implique d'évaluer « chacun des aspects un après l'autre » (S11, 338-339). Il décrit le lien entre l'analyse fonctionnelle et la programmation informatique en évoquant que « si l'analyse fonctionnelle n'était pas assez claire, bien tu vas avoir de la misère à programmer ton programme parce que tu ne sais pas ce que tu veux qu'il fasse » (S11, 395-397).

Le sujet 11 discute de l'implication de l'algorithmique dans la programmation informatique. En outre, il considère que cet aspect peut prendre davantage d'importance dans des situations où l'optimisation est importante :

> Le côté algorithmique, c'est vraiment quand tu as des tâches plus complexes à faire faut vraiment que tu fasses un algorithme pour effectuer ta tâche puis que... faut que tu analyses comment ton algorithme […] va performer en fonction de la taille des données qu'il va prendre en entrée. Puis, finalement, tu vas avoir le côté plus bas niveau dans le fond qui est finalement les instructions que tu envoies au processeur qui sont exécutées. (S11, 398-406)

Il soulève que l'optimisation est importante, et que dans certains cas l'algorithme en soi peut être optimisé mais reposer sur l'utilisation d'instructions non performantes. Il ajoute que « l'optimisation de bas niveau, ça dépend tout le temps de la tâche à faire » (S11, 412-413).

Le sujet 11 considère que la programmation de sites Web par laquelle il a commencé peut être considérée comme du bricolage dans le sens où elle implique un collage de propriétés HTML. Il affirme notamment que « tu arranges un peu les affaires puis là quand tu dis "ah, bien ça ne marche pas, je devrais changer telle affaire pour telle autre affaire" » (S11, 634-636).

Il affirme que l'essai-erreur ne peut pas être fait au moment de l'analyse fonctionnelle, car des erreurs faites à ce moment ne peuvent pas être corrigées plus tard :

> Quand tu fais de l'analyse, plus fonctionnelle, ou admettons quand […] tu sépares ton truc en classes ou en modules, […] tu ne peux pas vraiment faire d'essai-erreur à cette étape-là, parce que sinon après ça il est trop tard pour le corriger. Fait que si tu tombes à cette place-là, ça impliquerait de refaire l'abstraction, puis ça implique de recoder beaucoup de trucs. C'est vraiment plus quand tu es vraiment... tu as une manière de vérifier est-ce que ton truc il fonctionne ou pas. (S11, 724-732)

En ce qui concerne sa pratique personnelle, il affirme utiliser l'essai-erreur lorsqu'il considère que cette méthode est plus rapide :

> Tu es en train de coder de quoi puis ça ne marche pas, tu es comme "je pourrais réfléchir à ce que mon code fait, ou je pourrais juste essayer les trois ou quatre affaires, les trois quatre alternatives que j'ai, puis un moment donné ça va marcher". (S11, 616-621)

Pour illustrer cet usage de l'essai-erreur, il donne l'exemple de l'utilisation d'un nombre négatif ou d'un nombre positif dans une équation, affirmant qu'il serait possible de le déterminer en réfléchissant, mais qu'il est plus rapide de tester les deux cas possibles et de constater lequel fonctionne. Il envisage aussi l'essai-erreur à l'échelle de la démarche de recherche dans le cadre de son projet de maîtrise. Il posait des hypothèses, testait des algorithmes et constatait le résultat. Il donne le prototypage en exemple pour l'utilisation de méthodes concrètes : « Méthodes concrètes ça peut être aussi du prototypage » (S11, 670-671).

Le sujet 11 explique que la logique est nécessaire afin de parvenir à « découper ta tâche en morceaux » (S11, 534-535). Il affirme que contrairement aux mathématiques, la logique est nécessaire pour la programmation informatique.

4.11.3 Engagement affectif

Le sujet 11 affirme qu'une de ses amies qui a choisi de s'orienter vers une formation en systèmes d'information organisationnels doit évaluer les besoins des clients dans le cadre de son travail. Il affirme que « la programmation s'entrecoupe avec le côté social » (S11, 599-600).

Le sujet 11 nomme comme motivation le désir de comprendre comment les choses fonctionnent. Cette motivation s'applique autant dans le cas de son initiation à la programmation de sites Web que dans le cas de son doctorat qui concerne l'apprentissage machine. Avant de commencer la programmation Web, alors qu'il était au primaire, le sujet 11 affirme qu'il avait du plaisir à apprendre comment utiliser Microsoft Word :

> J'aimais ça parce que genre tu pouvais vraiment faire des beaux documents, c'est un peu comme les sites Web, un peu comme par rapport à ce que j'aimais aux sites Web, c'est que en faisant ça... en faisant ça dans un document Word, je pouvais refaire par exemple n'importe quel beau document que je voyais, tu sais des belles affiches, je pouvais faire des belles affiches avec... en faisant du Word. (S11, 864-871)

4.12 Analyse individuelle du sujet 12

4.12.1 Engagement comportemental

La première initiation du sujet 12 à la programmation informatique s'est déroulée au début de son baccalauréat qu'il a par la suite abandonné. Il affirme qu'il n'était pas prêt et que le contexte rendait difficile la poursuite des études : « Je n'étais pas outillé pour que ça fonctionne, je n'avais pas le bon environnement, on était 300 dans un cours, le prof il donne son cours, il s'en va » (S12,107-109). Lorsqu'il a choisi l'informatique, il n'imaginait pas qu'il aurait à faire de la programmation informatique. Le sujet a par la suite fait une technique accélérée en informatique, une formule qui lui convenait mieux :

> Au cégep, ça a été totalement l'inverse en étant dans une technique accélérée. Tu sais, au lieu d'être quatre mois, c'est deux mois par session, tu ne peux pas échouer un cours, si tu échoues tu es out, puis les profs ils sont attitrés, ils ne peuvent pas donner cinq cours, puis pendant deux mois ils sont intenses sur un seul cours, donc ils sont très près de toi fait que là c'était le bon environnement m'aider à réussir. (S12,110-116)

Depuis, il a travaillé comme programmeur dans différents contextes, notamment en Europe où il affirme que « ils forment beaucoup d'ingénieurs mais pas de programmeurs » (S12,49-50).

Le sujet 12 affirme que pour intervenir sur un code, le programmeur doit se construire une représentation mentale du programme. Il donne l'exemple de la programmation orientée objet :

> C'est de garder, surtout en ouvrant comme ça un nouveau projet, d'essayer de construire une image de toutes les classes, toutes les possibilités et par conséquent ouvrir le bon fichier, on se ramasse avec des centaines de répertoires puis de fichiers. (S12,502-506)

Cela peut amener des difficultés :

> À chaque fois, t'es obligé de dire "ah ok, là c'était là que je voulais faire un module, mais ce n'est pas là c'est dans la classe parent, ah bien non c'est dans la vue qui est liée au contrôleur, ah bien non c'est dans le modèle...", ainsi de suite. Donc il y a beaucoup, beaucoup, de petits fichiers à ouvrir, de petits codes, de petits bouts de code. (S12,508-513)

Le sujet 12 décrit sa tâche de programmation en évoquant l'analyse, la schématisation des données. Il travaille principalement en programmation Web et reprend des exemples de ce domaine, par exemple lorsqu'il explique comment se déroule la construction des maquettes : « Nous on prend les maquettes on rentre le contenu dedans, on s'arrange pour qu'une seule page soit utilisée pour toutes les pages d'un site, d'un seul gabarit qui reçoit tout le contenu » (S12,188-192). Il distingue l'analyse et la programmation : « C'est plus de l'analyse que de la programmation, c'est d'arriver à schématiser la réalité » (S12,406-407). Le sujet 12 propose de restreindre la définition d'un bogue en informatique :

> Les bogues je vois plus ça comme quelque chose... quand le projet est lancé depuis longtemps... Pour moi, un bogue ce n'est pas quand tu viens d'écrire une ligne puis qu'elle ne fonctionne pas. C'est un peu une distinction inutile, mais moi déboguer, c'est vraiment ça fait 6 mois que ça roule, je pense que tout va bien, mais là il y a un utilisateur qui vient me dire […] "eille aujourd'hui ça a fait ça puis ça n'a jamais fait ça, j'ai rien fait de différent". (S12,658-667)

Au moment de l'entretien, le sujet 12 raconte qu'il venait de passer 24 heures sur la résolution d'un problème. Il affirme qu'il recherchait obstinément la solution et qu'il devait attaquer le problème « toujours avec un autre angle, parce qu'un moment donné ça ne sert à rien de répéter » (S12,651-652).

Le sujet 12 raconte faire du bénévolat auprès d'un site Web dont l'objectif est d'informer des élèves du secondaire sur des carrières potentielles. Lorsqu'il parle de son métier, il met de l'avant la logique, les mathématiques, la résolution de problèmes et les frustrations.

Le sujet 12 donne l'exemple de sa mère qui ne comprend pas la télécommande de la télé « parce qu'elle essaie de retenir par cœur la suite de boutons » (S12,939-940). À l'inverse, il raconte que son enfant de 6 ans « est capable de changer la source de la télé sans voir le bouton, parce qu'il s'étire le bras puis il sait où il est le bouton » (S12,888-890). Il utilise cet exemple pour justifier que l'apprentissage de la programmation est possible à partir du milieu du primaire.

4.12.2 Engagement cognitif

Le sujet 12 affirme : « Pour moi les mathématiques ça désigne surtout l'arithmétique, mais je comprends que c'est bien bien d'autre chose, donc dans ce sens-là, savoir... c'est sûr qu'on utilise des fois un peu addition ou division ou tout ça » (S12,575-578). Il utilise les mathématiques surtout au niveau arithmétique dans son travail et donne l'exemple de l'affichage d'un pourcentage. Il a déjà eu à faire des usages plus complexes des mathématiques et donne l'exemple du travail avec des nombres en binaires et de l'utilisation de masques.

Le sujet 12 affirme que lorsqu'il entre pour la première fois dans un projet, il est long de construire une représentation mentale permettant de savoir où aller pour intervenir sur le projet. Il affirme que lorsque l'essai-erreur est utilisé, par exemple pour modifier un code sans vérifier si cela a des répercussions ailleurs, cela engendre un risque : « Soit qu'on connaît très bien le projet puis qu'on dit "ah, j'ai aucune crainte, je sais que je peux modifier ça, personne ne l'utilise", soit on est un peu insouciant » (S12,730-732).

Le sujet 12 affirme recourir à l'essai-erreur en phase d'apprentissage et donne l'exemple de l'utilisation du point-virgule :

> C'est sûr que quand on apprend, bien on met une ligne, oh, elle plante, "ah oui pas de point-virgule", faut que je me souvienne que ça prend un point-virgule partout. C'est ce genre d'essais là, tu mets une fonction, tu penses que ça va trier ton tableau, il dit "tu n'as pas mis les paramètres dans le bon sens", ok, j'apprends ça. (S12,744-750)

Le sujet 12 affirme recourir à l'essai-erreur parfois par lâcheté et donne l'exemple de l'ordre des arguments d'une fonction. Il affirme que l'essai-erreur ne devrait pas être utilisé en contexte professionnel, mais ajoute : « Sauf que tout le monde le fait [utiliser l'essai-erreur], donc dans un sens tout le monde devrait [immobiliser sa voiture] trois secondes à un [arrêt obligatoire], mais personne ne le fait » (S12,768-770).

Le sujet 12 affirme que la logique booléenne est « quasiment une des bases principales de la programmation » (S12,303-304). Il affirme y avoir été initié en secondaire 4, mais c'est au cégep qu'il en a compris l'importance. Il évoque que la manière de structurer le code implique aussi une part de logique et donne l'exemple de l'évitement des répétitions :

> Pour moi, la logique c'est vraiment... bien il y a le fait de dire qu'est-ce que ça veut dire une instruction qui est vraie puis une autre qui est fausse, mais il y a aussi dans comment on structure notre code, ça aussi au cégep ils nous le rentrent dans la tête : "évitez de dédoubler". (S12,585-590)

4.12.3 Engagement affectif

Le sujet 12 décrit la division des rôles dans son équipe de travail de développement Web. Il affirme que les rôles sont très bien définis. Chacun a une connaissance du rôle de l'autre, par exemple il est capable de « parler avec

la graphiste puis d'échanger sur ses décisions » (S12,203-204). Il est le seul programmeur de l'équipe dans laquelle il travaille et affirme que dans des équipes « plus rodées » (S12,678-679), il y a des méthodes de type agile où il y a un contrôle qualité et que plusieurs individus interviennent sur un code. Il affirme qu'ouvrir le code d'un collègue lui demande beaucoup de temps pour se situer.

Le sujet 12 affirme que « [la programmation] ce n'est pas juste de programmer l'ordinateur, mais c'est de représenter la réalité qui nous est présentée » (S12,425-427). Il est donc nécessaire d'interagir « avec la personne qui t'a donné une commande » (S12,453). Dans son domaine, celui du développement Web, le sujet 12 affirme que les problèmes finissent par se ressembler et qu'il est rare qu'un client demande quelque chose de complètement nouveau :

> Bon, sur le Web on est relativement... on ne parle pas de 20 millions d'environnements, on parle pas mal toujours d'une page Web donc on n'est pas obligés de dire "attends, attends, je n'ai aucune idée de quoi tu me parles", la plupart du temps le client il va te parler d'un site Web que tu connais ou d'en faire un nouveau qui est pareil à l'autre puis donc... on n'est pas si loin que ça. (S12,428-435)

Mais dans le cas où cela se produit, le sujet 12 affirme qu'il est nécessaire d'adopter une attitude d'ouverture :

> Je ne sais pas si ça peut rentrer dans les niveaux d'abstraction, mais pour moi c'est super important d'être à l'écoute de notre client. Quelque chose que j'ai souvent vu autour de moi, ou en tout cas vu à l'occasion autour de moi, d'autres programmeurs qui se braquent parce qu'ils disent "ouin mais ça serait bien plus simple s'il acceptait que...". On a tendance à recevoir une commande puis vouloir la modifier comment nous on la ferait, mais ce n'est pas ça notre travail […]. C'est important de se mettre dans les souliers du client puis de dire "ah ok, je comprends pourquoi il m'a demandé ça, pourquoi il veut de telle telle façon". (S12,373-386)

Le sujet 12 évoque l'aspect visuel du code qui se manifeste dans la façon d'indenter, de l'organiser, de commenter :

> Il y a un espèce d'aspect visuel au code, tu peux décider de mettre quatre retours vides ou le bracket au bout de la ligne ou en dessous de l'instruction, tu sais il y a une forme de liberté quand on écrit notre code que personne va jamais voir à moins d'ouvrir la page, le fichier de code. Donc, dans ce sens-là, c'est mon code, c'est ma façon de l'indenter, de l'organiser, de mettre des commentaires tout ça. (S12,844-851)

Le sujet 12 affirme : « J'ai toujours dit que c'était mon art à moi, c'était ma manière de créer, de programmer... pour moi c'est un art » (S12,314-316). Il affirme avoir parfois écrit du code dont il était fier en raison de la clarté à laquelle il était parvenu. Il évoque qu'il est difficile d'ouvrir le code des autres, mais que parfois certaines solutions optimales éprouvées peuvent l'amener à apprécier le code fait par d'autres.

Le sujet 12 affirme qu'il est patient, mais que certaines personnes peuvent être de bons programmeurs même s'ils sont impatients.

Le sujet 12 affirme avoir toujours été motivé par la résolution de problèmes, autant dans ses cours de mathématiques que dans sa formation en informatique.

4.13 Analyse individuelle du sujet 13

4.13.1 Engagement comportemental

Le sujet 13 a été initié à la programmation informatique après le Lycée alors qu'il était en attente pour aller poursuivre ses études supérieures en France. Il affirme qu'il avait pour « projets de vacances » (S13,71) d'apprendre à utiliser l'ordinateur et d'apprendre à conduire des automobiles. Il s'est donc inscrit dans une école d'informatique, mais son objectif n'était pas d'apprendre la programmation, mais plutôt l'usage de l'ordinateur. Étant donné son profil, il a été dirigé vers l'apprentissage de la programmation informatique. Le langage avec lequel il a commencé était GW-BASIC.

Le sujet 13 affirme :

> Quand j'entends programmation, je n'entends pas que l'ordinateur, j'entends prévoir une tâche de choses à une étape 0 par exemple et à une étape 1 cette liste d'étapes, de choses, on le confie soit à soi-même, soit à quelqu'un d'autre qui l'exécute. (S13,487-491)

La programmation informatique implique selon lui que l'exécutant est l'informatique, mais le terme programmation employé seul peut désigner un exécutant variable :

> Je programmais les humains. Mon script leur disait "prends telle chose, mets-le dans tel sens, coupe-le à telle longueur", ainsi de suite, et l'ouvrier exécute tout ça. Et à la fin, un produit final sort. Donc je considérais déjà que c'était de la programmation. L'exécutant, là-bas, c'est l'ouvrier, en accord ou en équipe avec les machines. Et cette même chose aujourd'hui, on tente de le transférer à un ordinateur, une machine qui est spécifiquement faite pour exécuter des tâches de façon automatique. Donc pour moi ça c'est de la programmation. Appliqué à l'ordinateur, c'est de la programmation informatique. (S13,503-513)

Le sujet 13 considère que la programmation informatique est une passion pour les uns et une activité lucrative pour d'autres. Il fait état de clivages entre ce que les programmeurs aiment créer et l'utilité que les clients y trouvent, soulignant que des méthodes comme Agile ont pour fonction de « cadrer le développement pour pouvoir mettre du temps, pouvoir mettre un coût sur le travail qu'on est en train de faire » (S13,404-405).

Le sujet 13 affirme que « dès lors que l'on projette déjà ce qu'on va faire demain, on y tient, on le cadre un peu, c'est une façon de programmer » (S13,476-479).

Le sujet 13 dresse une analogie entre l'usage de la voiture et l'usage de l'ordinateur. Il affirme que plus tôt au XXe siècle, la voiture était « une invention technologique qui nous a libérés où on peut voyager, on peut aller

loin, la notion de distance et de temps a changé » (S13,1062-1065). Il croit que cela se produira de la même façon avec l'informatique qui éventuellement deviendra un outil au même titre que d'autres. Il dresse le même parallèle avec l'apparition de la radio. Cela l'amène à affirmer que « si on regarde [l'informatique] comme un outil, l'informaticien magicien ça va baisser un peu et utilisateur besoin ça va monter » (S13,881-883).

Le sujet 13 relate plusieurs anecdotes en lien avec l'histoire des technologies qu'il a manipulées. Par exemple, il évoque l'apparition des fichiers de type CSV puis de XML. Il raconte l'usage de PowerBuilder qu'il a fait :

> [PowerBuilder] peut avoir n'importe quelle base de données comme Microsoft SQL Server ou Oracle ou n'importe quelle, mais il est plus compatible avec une base de données de SAP que j'oublie, donc les tables, quand vous les sélectionnez tout de suite, vous pouvez positionner les colonnes de n'importe quelle façon et ça fait comme un écran comme si on programmait un écran visuel. (S13,278-284)

Le sujet 13 raconte un usage des mathématiques au quotidien qu'il avait à faire dans son enfance : dans son pays d'origine, il existait une coupure de 100 francs imprimée sur un billet rouge. Les adultes non scolarisés l'appelaient « billet rouge » (S13,703), ce qui le forçait à faire des conversions spontanées. Il considère que cette utilisation spontanée quotidienne des mathématiques prédispose aux mathématiques : « du coup, tous les enfants qui font ça dans leur vie, ils ont des prédispositions mathématiques » (S03,710-712).

Le sujet 13 relate que pour programmer, il faut « descendre au niveau du raisonnement de l'ordinateur » (S13,719-720). Il considère qu'en faisant cela, tout le monde est capable d'y arriver. Le sujet 13 se prononce contre l'établissement de préalables mathématiques pour faire de la programmation informatique à l'école en affirmant que ces préalables empêchent des gens pourtant capables et intéressés d'accéder à la programmation informatique.

Le sujet 13 a milité pour les logiciels libres et est un des fondateurs du groupe Linux d'un pays d'Afrique.

4.13.2 Engagement cognitif

Le sujet 13 affirme que les mathématiques ne sont pas utilisées également dans toutes les formes de programmation informatique. Il donne un exemple : « Le programme qui a produit, qui a enregistré la liste des utilisateurs, qui a noté les heures de travail, qui a calculé leur salaire et tout, il y a très peu de mathématiques » (S13,681-684). Il considère que poser les mathématiques comme préalables à la programmation « est une erreur parce que les mathématiques sont juste une façon d'exprimer quelque chose que j'ai du mal à nommer » (S13,633-635). Il considère que cela a pour effet de mettre des gens de côté. Il donne ensuite l'exemple de l'arithmétique « parce que l'humain de façon naturelle il fait l'arithmétique » (S13,689-690).

Le sujet 13 s'en remet à la définition qu'il a proposé de la programmation selon laquelle l'exécutant peut varier. Ainsi, il considère que « plus le niveau d'abstraction est élevé, plus l'exécutant est générique ou vaste » (S13,569-571) et que « plus le niveau d'abstraction diminue, plus l'exécutant est plus précis » (S13,571-572). Il affirme que la programmation orientée objet permet de résoudre certains problèmes où il est important d'avoir un niveau d'abstraction permettant l'adaptabilité à différents contextes. Il donne l'exemple du langage UML qui sert à exprimer des représentations et de « rester aussi dans l'abstrait » (S13,600). Le sujet 13 met aussi l'abstraction en lien avec le principe de bricolage et affirme qu'à « un moment donné, il faut du concret, et ce concret n'a pu s'adapter au changement » (S13,825-826). Il ajoute que « avec le temps, on a compris que il vaut mieux adopter la notion de bricoler, voir le résultat, bricoler, voir le résultat parce que tout penser d'un seul coup on échappe certaines choses » (S13,826-829).

Le sujet 13 affirme que la programmation orientée objet « est une façon de... pas de modéliser mais de nommer l'abstraction, c'est une façon d'exprimer l'abstraction » (S13,591-592). Il soutient : « Entre les humains il faut qu'on ait un langage, il faut qu'on ait une façon de se transmettre nos idées, donc l'orienté objet en est une... un moyen » (S13,593-595). Il développe l'idée et soutient que « le langage est un intermédiaire... c'est comme moi je vous parle, et au lieu que j'utilise des verbes j'utilise le langage des signes, c'est un intermédiaire » (S13,550-553). Le sujet 13 ajoute : « Il reste à ce que cet intermédiaire soit assez riche pour faire passer toute mon idée ou toutes mes instructions » (S13,553-555). Le sujet 13 est d'accord avec l'idée que le bricolage fait partie de la tâche de programmation si le terme n'est pas employé de façon péjorative.

4.13.3 Engagement affectif

Le sujet 13 évoque l'apparition des fenêtres Windows 3.1, il présente la fenêtre comme quelque chose ayant du sens pour l'humain : « Ça a du sens pour l'humain, les gens ont commencé par apprécier, c'est une façon de rendre la chose accessible » (S13,1033-1035). Il compare avec Linux qui à cette époque s'utilisait en ligne de commandes. Il affirme : « Celui qui tape au clavier il a un niveau... je n'ai pas catégorisé, mais il fait plus d'efforts que quelqu'un qui avec un doigt va choisir un... avec une interface tactile » (S13,1039-1042). Il poursuit en comparant Linux et Apple, le premier étant orienté développeurs et le second orienté consommateurs.

Le sujet 13 évoque une différence entre programmer pour soi et programmer pour autrui, affirmant que ce tiers n'accorde peut-être pas de valeur à la démarche alors que le programmeur oui : « Pour lui c'est le résultat qui l'intéresse » (S13,195). Il en vient à affirmer : « Pour mes projets personnels, il m'arrive d'être plus appliqué que pour l'employeur » (S13,205-206).

Le sujet 13 évoque qu'il est difficile de gérer une équipe de programmeurs :

> On a du mal à les gérer, à la fin ils sortent un produit, ils sont contents de leur produit, mais le client ou l'industrie n'est pas content parce que le produit est beau mais il ne sert pas le client. (S13,400-403)

Le sujet 13 a appris à programmer car il anticipait la croissance de l'informatique : « Ce que j'anticipais c'est la forte présence d'ordinateurs dans notre vie proche et future et puisque les ordinateurs seront présents, moi je voulais savoir les manipuler » (S13,106-109). Nous rappelons que le sujet 13 a appris à programmer de façon fortuite alors que son intention première était d'apprendre à manipuler l'ordinateur. Il affirme :

> Le côté automatique de la chose me passionne, le côté vitesse, parce que d'où moi je suis parti, je programmais les humains. les humains, ils n'ont pas la vitesse de l'ordinateur, parce qu'il faut reconnaître que l'ordinateur n'a aucune intelligence si ce n'est que sa vitesse. (S13,215-220)

Finalement, le sujet 13 relate deux raisons à sa pratique de la programmation informatique : la passion et le métier. Pour expliquer la passion, il donne l'exemple de logiciels Open Source développés par des passionnés sans financement.

4.14 Analyse individuelle du sujet 14

4.14.1 Engagement comportemental

Le sujet 14 a fait de la programmation informatique pour la première fois dans le cadre d'un travail scolaire alors qu'il étudiait au baccalauréat en physique. Par la suite, il a réalisé une maîtrise dans laquelle il devait réaliser du calcul numérique à l'aide de la programmation informatique : « Ma maîtrise je l'ai faite à [université canadienne], c'était beaucoup de calcul numérique, donc là ça a été beaucoup plus de programmation pour le calcul numérique des simulations, dynamique moléculaire » (S14,34-38). Par la suite il s'est inscrit à une autre maîtrise, cette fois en informatique, et il affirme que « ça a vraiment été plus la formalisation des connaissances en informatique » (S14,50-51). Il affirme que cela lui a fait « laisser un peu le calcul scientifique pour faire plus de programmation plus informatique d'affaires » (S14,52-53).

Le sujet 14 réaffirme une distinction entre la programmation pour le « calcul numérique » (S14,99) et la programmation « informatique d'affaires » (S14,102,103). Il évoque que dans les deux cas, qu'il programme pour lui ou pour quelqu'un d'autre, il y a des objectifs à poursuivre. Ces objectifs varient dans les deux cas : « Si c'est un code pour faire du calcul numérique, bien il faut que ce soit optimisé, pour moi ou pour quelqu'un d'autre je vise la même chose » (S14,481-483) alors que « pour un besoin d'affaires, bien l'idée de la maintenabilité, le code propre, l'architecture logicielle, bien autant pour moi que pour quelqu'un d'autre c'est le même objectif » (S14,483-486). Il affirme :

> Si on prend le code d'une application Web, ce ne sera pas optimal, ce ne sera pas optimisé sur la performance tandis que du code pour le calcul numérique, bien au contraire le but c'est d'être le plus rapide possible parce que sinon les simulations sont très longues. (S14,368-372)

Le sujet 14 affirme que « dans la façon de programmer, il y a à peu près toujours une façon d'écrire le code que ça ressemble à des phrases ou des choses comme ça » (S14,734-737). Il établit ainsi un parallèle avec l'écriture de la langue. La programmation permet selon lui de relier la langue et les mathématiques : « C'est sûr qu'il y a un lien entre l'informatique puis les mathématiques fait que ça permet, ça relie un peu les deux » (S14,739-741). Le sujet 14 affirme que la réflexion abstraite est nécessaire étant donné que la résolution d'un problème ou le développement d'une fonctionnalité ne passe pas souvent par des chemins directs : « Il faut quand même séparer la fonctionnalité en sous-problèmes, en sous-fonctionnalités, selon ce qui a déjà d'existant pour ne rajouter des problèmes ou de la complexité dans le code » (S14,562-565).

Le sujet 14 raconte que sa pratique de la programmation informatique lui a permis de mieux comprendre certaines choses en lien avec l'informatique ou l'électronique en général :

> Tu regardes ici dans la pièce tout ce qu'il y a qui a potentiellement de l'informatique qui roule dessus, du code, je vois l'imprimante, le téléphone, je sais que c'est un téléphone IP, donc je sais que c'est différent, il y a le téléphone cellulaire, il y a l'écran. (S14,709-713)

4.14.2 Engagement cognitif

Le sujet 14 affirme que « il y a moyen de programmer que ce soit autre chose que des mathématiques et de la logique » (S14,399-400). Il redonne l'exemple de la programmation Web où « on ne traite pas des chiffres directement ou des 0 et des 1 directement » (S14,402-404). Il se sert de cet exemple pour affirmer que « une personne qui est un peu moins à l'aise avec les mathématiques ou la logique pourrait quand même être un bon programmeur » (S14,404-406).

Le sujet 14 affirme que la programmation informatique implique de réfléchir à plusieurs niveaux d'abstraction :

> On parle de plusieurs niveaux d'abstraction, on en a parlé tantôt un peu avec les cours à la maîtrise, de parler d'architecture logicielle, l'assurance qualité, ce qui s'appelle le code propre, la décomposition en différents modules, en différentes classes quand c'est de l'orienté objet, c'est quand même un autre niveau. (S14,335-341)

Le sujet 14 apporte une nuance sur l'usage du terme « simplement » (S14,639) dans l'énoncé 3.1 affirmant que l'algorithmique, même si cela implique de communiquer des instructions à un ordinateur, n'est pas toujours simple.

Le sujet 14 évoque que l'aspect bricolage peut se constater aussi dans la méthode du test-driven development. Cette méthode enlève la portion aléatoire du bricolage étant donné que les tests sont réfléchis d'avance par le programmeur. Le sujet distingue le bricolage de l'essai-erreur :

> […] essai-erreur, c'est plus essayer des choses un peu aléatoirement jusqu'à temps qu'un moment donné ça va fonctionner, il y a moins de planification qui est faite. Dans le fond le bricolage c'est plus essayer de brancher des morceaux jusqu'à temps que ça fonctionne au lieu d'essayer de prévoir un chemin, un plan précis. (S14,531-537)

Le sujet 14 affirme : « mathématiques, logique, à mon avis ce n'est pas primordial pour être programmeur » (S14,432-433). Il est d'accord avec l'énoncé affirmant que ça prend certaines capacités intellectuelles d'analyse.

4.14.3 Engagement affectif

Le sujet 14 observe une différence dans le travail d'équipe qu'il a réalisé à sa maîtrise et plus tard en informatique d'affaires. Lors de sa maîtrise, chacun avait son projet et la collaboration se faisait via l'entraide mutuelle : « Des fois il y a des questions que quelqu'un bloquait, bien c'était facile d'aller parler aux autres, poser des questions aux autres, échanger avec les autres » (S14,126-128). Au sujet de son expérience en milieu de travail, il affirme : « On était plus une équipe de développement, une vingtaine de personnes sur le même projet sur le même code, fait que là c'est sûr qu'on collabore plus étroitement au même projet » (S14,132-135).

Le sujet 14 affirme que la transformation du besoin d'un client en morceaux de code, « ce n'est pas une tâche simple » (S14,463). Il établit un lien avec le concept d'expérience utilisateur pour décrire la tâche d'analyse et donne l'exemple d'une demande de fonctionnalité d'un client où l'emplacement des boutons a une importance.

Le sujet 14 affirme que la patience varie d'une personne à l'autre. Il affirme : « Il y en a qui, avec le temps ou pour différentes raisons, deviennent moins intéressés à apprendre des nouvelles choses ou à changer des façons de travailler des fois » (S14,440-443). Il identifie comme source de frustrations la lenteur de la compilation : « Quand c'est vraiment trop long ou que ça ne fonctionne pas, ça va être une autre source de frustration » (S14,601-603). Il décrit les frustrations comme inhérentes au travail d'équipe :

> Dès qu'on travaille en équipe, tout ça, nécessairement on va avoir des différences entre les personnes donc la façon de voir un problème ou la façon de résoudre un problème, il va y avoir des différents qui vont se créer, donc oui des frustrations sûrement. (S14,408-413)

Le sujet 14 affirme que ses motivations ont changé, au début il faisait de la programmation par intérêt personnel pour le calcul numérique alors que dans son emploi actuel il est davantage intéressé par l'automatisation. Il continue à faire du calcul numérique par intérêt personnel. Il explore aussi le domaine de l'intelligence artificielle par intérêt personnel affirmant que cela rejoint le calcul numérique et les mathématiques. Questionné sur la différence entre programmer pour soi ou pour quelqu'un d'autre, il affirme : « [la transposition d'un besoin en

programmation] est différente un peu parce que l'idée vient de moi » (S14,493-494). Il ajoute : « La compréhension est plus facile à faire, parce que j'analyse une idée que j'ai eue, c'est sûr qu'il y a une étape de moins » (S14,495-497). Il souligne qu'au-delà de cette étape, la programmation pour lui ou pour autrui, « il n'y a plus de différence » (S14,499).

4.15 Analyse individuelle du sujet 15

4.15.1 Engagement comportemental

Le sujet 15 affirme avoir commencé la programmation informatique lorsqu'il était en secondaire 3 avec le logiciel mIRC : « mIRC il avait son propre langage de programmation, c'était un programme vanille où les gens parlaient ensemble, puis si tu voulais tu pouvais commencer à développer puis là tu pouvais faire des logiciels, des petits jeux avec mIRC » (S15,81-85). Par la suite, il a étudié en technique en informatique au cégep puis a réalisé un premier stage professionnel à l'agence canadienne de développement international. Il a ensuite fait un baccalauréat en informatique puis des stages dans une compagnie d'assurance où il travaille aujourd'hui, depuis 2007. Il a aussi travaillé en Europe en développement informatique. Nous n'avons pas été capable d'établir à quoi le sujet référait lorsqu'il a utilisé l'expression « programme vanille ».

Le sujet 15 affirme que la programmation orientée objet « est une meilleure représentation vraiment de ce qui se passe à l'entour de nous » (S15,250-251). Questionné sur les composantes au cœur de la programmation informatique, le sujet affirme que la manipulation d'un interrupteur pour allumer une lumière n'est pas de la programmation informatique étant donné que « il n'y a pas de réflexion » (S15,602-603). Il affirme : « L'interrupteur, c'est peut-être un petit peu trop simpliste, mais pour moi de programmer son thermostat, c'est de la programmation » (S15,587-589). Il affirme que tout le monde fait de la programmation et donne plusieurs exemples outre que celui du thermostat : « Quand tu programmes ta manette de télévision, tu programmes, quand tu set l'heure sur ton micro-onde c'est de la programmation, dès que tu donnes une instruction à une machine, tu programmes » (S15,546-550).

Le sujet 15 affirme que certaines personnes, en informatique, ont une réflexion plus proche de la machine que d'autres. Il ne se considère pas de ceux-là : « Tu vas avoir des patterns qui ont une équivalence qui vient du vrai monde » (S15,213-214). Il donne des exemples issus de son travail où il doit modéliser des REER, des produits d'épargne. Il affirme : « Dans le fond on développe ce qui se passe vraiment en réalité » (S15,257-258).

Lorsqu'il est questionné sur la similarité entre un emploi de programmeur en assembleur et un emploi de programmeur orienté objet, il compare cela à la différence entre « quelqu'un qui enseigne à la garderie » (S15,199-200) et un « enseignant d'université » (S15,201).

Le sujet 15 raconte que sa fille sait utiliser son téléphone afin d'illustrer que les produits de Google et Apple sont faciles à utiliser. Il évoque : « Il y a des gens que c'est leur métier de s'assurer que les produits de Google, de Apple, que n'importe qui puisse les utiliser » (S15,1154-1156).

Le sujet 15 raconte que dans sa formation, ses professeurs amenaient les étudiants à explorer plusieurs méthodes de résolution de problèmes :

> Quand tu es à l'université ou au cégep, puis souvent les profs vont te poser une question genre "c'est quoi la meilleure méthode pour rechercher dans un tableau de strings?". Puis là tout le monde va faire la même méthode, une boucle puis qui va chercher, puis là après il va te dire "oui, mais là, il y a des méthodes beaucoup plus performantes". (S15,916-922)

Le sujet 15 affirme que la programmation informatique, « c'est toujours d'être créatif dans la solution à un besoin » (S15,1028-1029). Il construit un exemple sur mesure en lien avec son domaine d'emploi (la finance) pour illustrer comment cette créativité peut se manifester : « Les gens ils ont besoin d'imprimer les relevés de tous nos clients à chaque année, à chaque année, il faut imprimer l'ensemble des relevés de tous les sujets » (S15,991-993). Puis il explique qu'il est impossible de lancer l'impression de centaines de milliers de relevés simultanément en une nuit étant donné le nombre d'imprimantes disponibles, par exemple. Il affirme qu'il faut trouver « le meilleur angle » (S15,994). Son équipe se questionne alors sur une façon de diviser l'impression : « On peut tu réfléchir à comment on va atteindre le but d'imprimer des relevés de tout le monde, mais intelligemment puis pas juste lancer une impression qui ne finira jamais » (S15,1006,1009).

4.15.2 Engagement cognitif

Le sujet 15 affirme que « les maths ça dépend vraiment ce que tu fais comme type de programmation je pense » (S15,787-789). Il parle ensuite du métier d'expert d'expérience utilisateur en disant :

> Lui [le designer], s'il n'a aucune connaissance mathématiques, ce n'est pas grave du tout, il fait des maquettes, il s'assure que les écrans ils ont du sens, que les interactions entre chaque étape du processus ont du sens, puis que d'un écran à l'autre tu n'es pas perdu. (S15,807-811)

Il ajoute que « par contre, faut que son [le designer] esprit de logique soit très développé » (S15,812-813).

Le sujet 15 affirme que l'abstraction ne s'entend pas de la même façon en programmation orientée objet ou en assembleur :

> En assembleur, c'est très complexe, parce qu'il faut justement que tu connaisses au niveau hardware, vraiment comment les choses fonctionnent, fait que faut que tu sois capable de comprendre au niveau machine, tandis qu'en langage orienté objet, bien là faut que tu conceptualises des objets, fait que c'est très différent. (S15,703-709).

Selon le sujet 15, la programmation en assembleur implique un niveau d'abstraction pour se mettre au niveau de la machine et la programmation orientée objet pour se mettre au niveau « objet relationnel conceptuel » (S15,712-713).

Le sujet 15 affirme que l'algorithmique « c'est plus que de donner des instructions, c'est plus justement de conceptualiser des principes » (S15,910-912).

Le sujet 15 considère que la place de l'essai-erreur dans la programmation informatique dépend du type de tâches effectuées ou du stade de développement d'un projet : « Si tu fais du nouveau développement, bien là tu vas faire des essais, c'est des nouveaux essais sur du code qui n'a jamais été testé, donc là ce n'est pas du essai-erreur, c'est plus une réflexion de comment ça devrait fonctionner » (S15,858-862). Il affirme que « si c'est vraiment un programme existant que là il marche plus, il y a un problème, bien là tu es plus dans l'essai-erreur d'essayer de comprendre qu'est-ce qui se passe » (S15,866-869). Il donne aussi pour exemple d'utilisation de l'essai-erreur le support ou la maintenance.

Le sujet 15 est d'accord avec l'énoncé qu'il ne faut pas avoir peur des mathématiques et de la logique mais souligne qu'il s'agit d'une utopie, car « sur le terrain, il y a des gens qui sont très mauvais en mathématiques qui sont programmeurs » (S15,796-798).

4.15.3 Engagement affectif

Au moment de la rencontre, le sujet 15 occupe un poste de gestionnaire de projet depuis quelques mois; auparavant il était programmeur. Il affirme avoir fait ce changement car « [dans une équipe de développement] on est souvent les derniers dans le processus décisionnel d'une entreprise » (S15,429-430). Il souligne toutefois qu'il appréciait l'ambiance de travail avec ses collègues. Il affirme que « souvent ils [les patrons] ne comprennent pas l'importance » (S15,454-455) de la maintenabilité d'un programme informatique. Il compare cela à la mécanique automobile : « Tu veux le meilleur rapport qualité-prix » (S15,463-464).

Le sujet 15 évoque le cas du *pair programming* et affirme que cette technique n'est pas toujours appliquée de la même façon : parfois les deux ont le même rôle, et parfois l'un des deux agit davantage comme analyste. Il affirme avoir travaillé de cette façon avec un pair davantage spécialisé dans le contrôle qualité.

Le sujet 15 raconte avoir été confronté à des clients qui ne comprenaient pas la tâche de programmation informatique. En parlant du client, il affirme : « Lui, tout ce qu'il veut, c'est payer moins cher » (S15,453-454).

Le sujet 15 affirme : « Tu sais maintenant c'est quasiment de l'art là » (S15,953) et que la programmation informatique « dans le fond c'est un peu de créativité » (S15,986).

Le sujet 15 s'est intéressé à la programmation informatique pour la première fois avec mIRC qui permettait de développer « des petits jeux » (S15,85). Il raconte que « je m'étais fait des amis sur mIRC puis on développait un peu des logiciels, quand on parlait, ça encodait les messages » (S15,86-88). Il raconte avoir participé à « développer des protocoles de transfert pour partager des fichiers en ligne » (S15,94-95). Il affirme avoir choisi la technique en informatique sans se questionner plus outre : « Quand j'avais fait un stage, j'avais aimé, fait que j'ai continué avec le bac » (S15,105-106). Il nomme aussi la rémunération comme source de motivation.

4.16 Analyse individuelle du sujet 16

4.16.1 Engagement comportemental

Le sujet 16 a appris la programmation lors de son baccalauréat en sciences géomatiques. Il parle alors « d'enseignement formel » (S16,34) de la programmation informatique. Il a par la suite programmé et appris la programmation informatique « par des projets de recherche, par des stages, puis en emploi, là j'ai eu à travailler surtout sur des bases de données que j'ai développées » (S16,35-38).

Le sujet 16 dit de la programmation informatique que « c'est de la construction, on met les blocs un à la suite de l'autre pour arriver à quelque chose » (S16,883-885). Cette construction fait suite à un travail préalable d'algorithmique et d'analyse : « On peut dessiner, on peut concevoir un algorithme, concevoir un programme, il y a cette partie-là d'analyse, mais la programmation strictement, c'est de la construction » (S16,881-884). Il parle aussi de la programmation informatique comme « une façon d'aborder un problème, d'aborder la communication d'une solution qui est complètement différente de ce qu'on fait intuitivement ou naturellement » (S16,152-155). Selon lui, la différence d'un langage à un autre, « c'est plus que juste la syntaxe […] il y a la syntaxe, c'est sûr, mais en arrière, c'est grosso modo les mêmes fonctions, les mêmes comportements que tu veux faire faire à l'ordinateur » (S16,165-168). Il donne un autre exemple :

> Dans l'ordre, j'ai appris le langage C, et ensuite j'ai appris du VB, et tout d'un coup, en apprenant Visual Basic, tu regardes le problème à l'envers : tu regardes le problème plus en termes de 'je veux avoir telle interface'. (S16,168-172)

Le sujet 16 énonce que la programmation informatique varie d'un langage à l'autre :

> D'un langage à l'autre, tu n'implanteras pas ta solution de la même façon, tu n'as pas les mêmes considérations, tu ne cherches pas à optimiser ou à être efficace sur les mêmes aspects en fonction du langage ou en fonction de l'environnement. (S16,198-202)

Il donne des exemples en ce sens : « Un code de qualité ou un code optimisé en C, c'est important, quand tu arrives en VB, ça n'a plus aucune incidence » (S16,179-181). Il donne aussi comme exemple la programmation orientée objet en C++ : « Ensuite j'ai fait du C++ dans une approche purement orientée objet […] puis là ça

devenait complètement différent comme façon de voir comment l'information vivait à l'intérieur du programme, comment l'ordinateur accédait à l'information » (S16,184-194).

Le sujet 16 décrit une part du travail cognitif impliqué dans la programmation informatique disant que « il faut toujours garder en tête qu'est-ce que mon programme a mémorisé de ce que j'ai fait, mémorisé étant évidemment les variables, comment est-ce que je peux le rappeler, où est-ce que c'est » (S16,552-556).

Le sujet 16 établit une analogie entre le programmeur et un chef d'orchestre, disant que « il y a plusieurs choses qui se font en même temps, plus ou moins, mais je veux dire faut les coordonner puis faut que les choses arrivent en même temps ou existent plutôt avant que j'en aie besoin dans le programme ».

Le sujet 16 raconte qu'il saisit les occasions de programmer pour des projets personnels à différents moments dans l'année, alors que son métier consiste désormais à enseigner plutôt qu'à programmer :

> Je n'ai plus d'occasions de programmer beaucoup, mais il y a des moments dans l'année, surtout la session d'été où est-ce que là on a plus de temps pour se consacrer sur des projets, j'essaie d'avoir un projet de programmation. (S16,562-566).

Le sujet 16 raconte avoir souvenir que son père lui avait dit « si tu penses que l'ordinateur devrait faire quelque chose, dis-toi qu'il y a quelqu'un qui y a déjà pensé, puis qu'il le fait déjà » (S16,88-90). Il raconte ensuite avoir remis en doute cette affirmation, notamment en disant : « J'ai rencontré des frustrations, des situations où est-ce que je me disais "l'ordinateur ne fait pas les choses comme je voudrais qu'il fasse" » (S16,93-95).

Le sujet 16 raconte avoir appris un langage de programmation par loisir : « L'année passée j'ai pris un cours à distance sur le python juste parce que ça me tentait de réapprendre un nouveau langage » (S16,259-261).

4.16.2 Engagement cognitif

Le sujet 16 affirme : « Mathématiques, informatique, logique, je les vois comme trois stratégies pour développer une certaine compétence en algorithmique, j'aime prendre le chapeau plus d'algorithmique et de résolution de problèmes » (S16,762-767). Il considère que les mathématiques font partie de la programmation informatique, même si cela n'est pas toujours apparent : « Il ne s'en rendra pas compte, il va peut-être y [les mathématiques] faire appel quand même, mais il pourrait quand même avoir l'impression de ne rien savoir en mathématiques puis de quand même pouvoir faire un minimum de programmation » (S16,783-787). Le sujet 16 évoque toutefois que « on peut probablement se passer de la pensée mathématique, puis quand même être capable de faire des choses, les gens vont être capables d'être logique sur des formes géométriques ou sur des lettres » (S16,769-772). Finalement, le sujet 16 identifie un blocage que certaines personnes peuvent avoir avec les mathématiques : « Il se passe quelque chose avec les mathématiques que des gens vont vraiment bloquer [...]

on a quasiment l'impression des fois que c'est un mécanisme de défense pour certaines personnes de ne pas vouloir faire de mathématiques » (S16,774-779). Il donne l'exemple un exemple où la pensée mathématique n'est selon lui pas essentielle : « On peut développer des sites Web qui vont faire tout ce qu'un site Web peut faire aujourd'hui sans même avoir nécessairement de pensée mathématique » (S16,755-757).

Le sujet 16 affiche une vision empirique de l'abstraction : « On ne peut pas mettre la main dans l'ordinateur pour sortir une variable, ça n'existe pas, donc oui il y a un gros niveau d'abstraction qui est nécessaire » (S16,639-641). Selon lui, « la pensée abstraite, chez les jeunes, ça vient quand même après un certain âge » (S16,1130-1131). Le sujet 16 évoque quelques exemples de ce qu'il considère comme des niveaux d'abstraction en programmation informatique :

> Est-ce que je me mets à la place de l'utilisateur puis j'essaie de voir comment est-ce que lui va percevoir le programme? Est-ce que je le pense en termes de la racine pure du problème que j'ai puis je lui trouve une solution, c'est un autre niveau. Est-ce que je regarde sur les risques qu'il y a? Est-ce que je le regarde d'un point de vue de l'optimisation? Est-ce que je le regarde d'un point de vue de généralisation? (S16,678-685)

Il énonce ensuite :

> Un programmeur efficace va devoir être capable de jongler ou de combiner tous ces points de vue là pour arriver à sa solution pour un problème donné […] il n'a pas le choix de faire cohabiter tous ces niveaux-là un moment donné. (S16,688-692)

Le sujet 16 affirme :

> Il faut vraiment justement, être capable de simuler un raisonnement dans sa tête, être capable de faire semblant d'être à la place de l'ordinateur, d'être capable de penser justement en termes de variables ou de penser en termes d'objet, alors que c'est complètement intangible. (S16,634-638)

Le sujet 16 considère que la pensée abstraite peut s'inférer dès qu'un individu utilise du papier et un crayon pour réfléchir à un problème :

> Dès que tu fais ça [mettre sa pensée sur papier], moi je considère que tu es rendu en [pensée] abstraite, même si c'est pour réécrire des lignes de code ou pour mettre des choses, tu travailles dans ta tête, tu n'es plus juste en train d'essayer de faire faire quelque chose à l'ordinateur puis de le dompter pour qu'il fasse ce que tu veux, tu es vraiment toi en train de mettre de l'ordre dans tes idées puis de remettre de l'ordre dans le processus que tu veux implanter. (S16,941-948)

Le sujet 16 donne une définition de l'algorithmique : « L'algorithmique, c'est la structuration de notre pensée, l'algorithmique, ce n'est pas dire quoi faire à l'ordinateur, c'est nous, comprendre comment on veut le dire à l'ordinateur » (S16,1022-1024). En parlant de ses étudiants, il affirme : « [ils] n'ont pas de misère à comprendre si je leur donne un code, ils n'auront aucune difficulté à lire un code et à comprendre qu'est-ce que le code fait »

(S16,1042-1045). Selon lui, les difficultés surviennent « quand on leur demande à eux de prendre ta solution à ton problème, comment est-ce que toi tu vas la présenter à l'ordinateur » (S16,1046-1048).

Le sujet 16 considère l'algorithmique comme une compétence pouvant se mobiliser dans différents contextes : « Pour moi l'algorithmique comme méthode de résolution de problèmes, c'est présent partout, je vois des algorithmes dans tout ce que je fais tous les jours » (S16,1056-1058). Selon lui, cet apprentissage de l'algorithmique est plus important que l'apprentissage de la programmation.

Le sujet 16 donne l'exemple d'utilisation de méthodes concrètes comme la réutilisation de codes pour construire un programme :

> Il va y avoir de l'essai-erreur, qu'il va y avoir de la recherche justement tantôt comme je disais sur les forums ou sur ce qui s'est déjà fait, il y a beaucoup de réutilisation, c'est même l'intérêt de la programmation qui fait en sorte que c'est... construire un programme. (S16,875-880)

Sur la pratique de l'essai-erreur spécifiquement, il affirme : « On peut très bien arriver, prendre un problème, s'asseoir devant l'ordinateur, puis faire des choses, mais un moment donné, si le problème est complexe, il va falloir quand même juste prendre le temps puis le faire, l'exercice d'abstraction du problème » (S16,935-939).

Le sujet 16 affirme :

> Pour être un bon programmeur, ça prend les deux [mathématiques et logique], mais quelqu'un pourrait programmer sans faire de mathématiques. […] on ne pourrait pas se passer de la logique par contre, j'aurais de la misère imaginer quelqu'un qui a de la misère à mettre en ordre une séquence d'étapes, ce que j'appellerais de la logique ou de faire un minimum de raisonnement logique, je pense qu'il n'aurait pas mal plus de misère. (S16,795-802)

4.16.3 Engagement affectif

Le sujet 16 évoque que, dans le cas d'un programmeur, « tu n'as pas d'excuse de bloquer sur un problème, fouille un peu sur internet, tu vas trouver une solution à ton problème, il y a du monde qui sont déjà passés par là » (S16,840-842). Il évoque l'existence de « communautés derrière les équipes de développement ou derrière les langages qui vont t'aider » (S16,843-844).

Dans le cadre de sa maîtrise, le sujet 16 s'est intéressé au « cycle de développement d'un système d'information » (S16,343-344). Il évoque « l'approche de maquettage direct ou le maquettage rapide » (S16,349) et la décrit comme une méthode où une solution doit être proposée le plus rapidement possible au client afin de l'aider à identifier ses besoins.

Le sujet 16 évoque que le débogage peut être une source de frustration en programmation informatique : « Du débogage, c'est l'exemple [de frustration] le plus... qui me vient en tête en premier » (S16,816-817). Il compare

cela à la correction d'un texte en français : « C'est comme corriger un texte en français ou des choses comme ça, un moment donné c'est juste... ça marche, mais ce n'est pas fini » (S16,818-820). Il réagit à l'énoncé en lien avec les frustrations disant que les frustrations, « c'est le quotidien de la programmation » (S16,713-714). Il ajoute que « il y a du monde qui vont se décourager ou qui vont affirmer ou qui vont décider "la programmation, ce n'est pas pour moi" » (S16,717-719).

Le sujet 16 a appris la programmation informatique dans le cadre de sa formation. Il affirme qu'il avait des attentes par rapport à la programmation informatique et affirme s'être dit : « Je suis tanné de faire juste ce que l'ordinateur me dit que je peux faire, puis je veux prendre le contrôle de la machine puis lui dire moi-même [ce] que j'ai besoin qu'il fasse » (S16,69-73). Il n'a jamais envisagé faire une carrière « strictement en programmation » (S16,298) bien que cela fasse partie de son métier de façon périphérique (par exemple, aujourd'hui il enseigne la programmation informatique). Il évoque aussi des motivations personnelles disant qu'il peut « investir des journées dans un problème de programmation » (S16,251-252). Il décrit ce que lui apporte la programmation informatique ainsi : « Même s'il [le programme] ne marche pas, ce n'est pas grave, tu es juste satisfait d'avoir pensé à un problème comme ça puis de t'être consacré à un problème comme ça » (S16,252-255).

4.17 Analyse individuelle du sujet 17

4.17.1 Engagement comportemental

Le sujet 17 a été initié à l'informatique en général via l'ordinateur familial. Il évoque « Quelques ordinateurs, quelques portables » (S17,34-35) et affirme que « Maintenant je suis plus avec les tablettes » (S17,35-36). Pour la programmation informatique, il y a été initié au cégep dans une technique en informatique : « On a plus appris des langages, langage C, Visual Basic aussi » (S17,58-59). Il nomme des ordinateurs avec lesquels il a travaillé lorsqu'il était plus jeune : « Avec le Basic qu'il y avait directement sur le Commodore 64, c'est comme ça que j'ai appris principalement » (S17,40-42).

Le sujet 17 affirme que les langages de programmation peuvent s'utiliser à différents niveaux : « On prend ces langages-là, ces outils-là, au niveau où est-ce qu'on peut, puis un moment donné à force de travailler, on peut arriver à d'autres niveaux » (S17,496-499). Le sujet 17 raconte à plusieurs reprises des anecdotes en lien avec l'utilisation de Raspberry Pi. Il affirme qu'il « trouve qu'on revient à un niveau plus de le mettre dans les mains de monsieur madame tout le monde » (S17,258-259). Selon lui, cet aspect d'accessibilité de la technologie est important car « à quelque part, quand on n'a plus [...] la pensée qu'on a cette capacité-là, […] bien on perd notre liberté, on la laisse à quelqu'un d'autre » (S17,260-264).

Le sujet 17 définit l'ordinateur disant que « il y a plusieurs niveaux à ça, il y a des couches de la communication, des bases de données, traitement de texte, et cetera » (S17,433-435). Il évoque ensuite que « entre le Basic puis l'assembleur, il y a des grosses différences, mais de comprendre tous ces concepts-là puis de les appliquer, on n'est pas du tout au même niveau-là » (S17,331-334).

Le sujet 17 utilise l'exemple de la peinture du pont de Québec pour illustrer le potentiel de la robotique basée sur Raspberry Pi et Arduino :

> Le pont de Québec, je trouve que c'est désastreux ce qui arrive avec [la peinture], puis ce que je me rends compte, c'est qu'il pourrait facilement avoir une faculté de robotique ici qui fonctionnerait avec des petits appareils comme ça pour faire du développement. (S17,201-205)

Le sujet 17 dresse une analogie avec le sport ou le spectacle disant, au sujet de la programmation informatique : « C'est sûr qu'au départ, […] il n'y a personne qui est Wayne Gretzky en partant […], c'est normal […] d'avoir des frustrations au départ, en persévérant » (S17,642-645).

Le sujet 17 fait état d'un test ou d'une étude dont il a été informé : « Il y a un test qui a été fait […] quand on fait juste taper un rythme, mais qu'on connaît la chanson, on ne comprend pas que l'autre personne n'est pas capable de [...] l'entendre juste en tapant » (S17,742-745). Il se sert de cet exemple pour affirmer que « quand on ne comprend pas que l'autre personne ne comprenne pas parce que nous on l'a compris, puis on trouve ça facile, là c'est plus difficile d'enseigner ou de faire comprendre à l'autre personne » (S17,748-751). Le sujet 17 affirme : « Je suis un kinesthésique, donc ça me prend la troisième dimension pour bien comprendre les choses » (S17,1039-1040).

4.17.2 Engagement cognitif

Le sujet 17 parle des mathématiques et de la logique disant que « c'est deux sortes d'intelligence qui sont souvent reliées, mais pas nécessairement exclusives ou nécessairement associées » (S17,692-694). Il affirme ensuite que « il y a beaucoup de programmation qui n'est pas du tout mathématique » (S17,704-705). Le sujet 17 s'en remet à la théorie des intelligences multiples pour expliquer que certains individus ont de la facilité en mathématiques et d'autres non : « Il y aurait, il semblerait, 8 ou 9 types d'intelligence, c'est évident qu'on ne les a pas tous au même niveau » (S17,366-368).

Le sujet 17 affirme qu'il y a plusieurs aspects à l'abstraction, notamment l'aspect « système d'exploitation où est-ce qu'on a des choses qui sont vraiment machines, c'est... comment on met l'information sur le disque, comment on communique avec l'écran » (S17,523-526). Il évoque aussi d'autres niveaux d'abstraction en lien avec la « communication avec un ordinateur, avec un réseau » (S17,527,528). À l'intérieur d'un même langage, il considère qu'il peut y avoir « des niveaux de complexité ou d'interaction » (S17,540-541). Il évoque le cas de

la recherche informatique qui selon lui « prend un niveau d'abstraction particulier » (S17,856) et donne l'exemple de l'intelligence artificielle.

Questionné sur la place de l'essai-erreur en programmation informatique, le sujet 17 affirme que « procéder par essai-erreur, oui, puis il ne faut pas hésiter à le faire » (S17,826-827). Il donne l'exemple de son Raspberry Pi en ajoutant : « Je ne peux pas le briser » (S17,828-829).

Le sujet 07 établit une distinction entre la logique et les mathématiques disant que « on peut être très logique […] par exemple il y a beaucoup d'avocats qui sont très logiques dans leur façon d'être séquentiel ou de liens entre les choses, mais ce n'est pas nécessairement au niveau mathématique qu'ils le sont » (S17,685-689). Il affirme que la logique est moins évidente pour certaines personnes, mais il ajoute : « Je crois que c'est une partie quand même importante de la programmation » (S17,374-375). Il ajoute que « il y en a qui ont l'impression qu'on a une intelligence finie, […] il y en a d'autres qui croient qu'on peut se développer, je fais partie de l'école de ceux qui [pense que les gens] peuvent se développer » (S17,377-382).

4.17.3 Engagement affectif

Le sujet 17 évoque que « il y a des gens qui ont cette capacité, qui connaissent tellement leur domaine qu'ils sont capables d'aller à d'autres niveaux puis de poser des questions […] au développeur, puis de dire "non ça ne fait pas ce [dont] j'ai besoin" » (S17,814-818). Il ajoute que « ce n'est pas parce qu'ils ne sont pas capables de le faire eux-mêmes, parce qu'ils n'ont pas les outils de développement en arrière, qu'ils ne sont pas capables de le réfléchir » (S17,818-821).

Le sujet 17 raconte que lors de sa technique en informatique au cégep, « c'est là que j'ai plus appris vraiment les bonnes pratiques » (S17,55-56). Il ajoute qu'il a encore « des petits côtés amateurs » (S17,57) en riant. Il évoque l'existence de communautés entourant l'utilisation de Raspberry Pi : « C'est incroyable les communautés qu'il y a là-dessus » (S17,143-144), puis évoque l'existence de Raspberry Pi magazine. Le sujet 17 nomme l'existence de Raspberry Jam « qui sont des sessions justement pour regrouper des gens qui sont enthousiasmés par ça, puis de faire soit des projets de programmation ou de robotique » (S17,162-164). Le sujet évoque « ce côté-là […] de communauté qui permet de développer des logiciels incroyables, Wikipédia entre autres » (S17,568-570). Il ajoute que « c'est relativement facile de s'adjoindre à ces équipes-là » (S17,570-571).

Au sujet de ses motivations à programmer, le sujet 17 affirme que « c'est un petit peu comme la science-fiction, c'est tout le potentiel que ça a, c'est-à-dire de voir qu'est-ce qu'on peut faire avec ça » (S17,197-199). Il développe le lien avec la science-fiction en disant : « J'aime bien la science-fiction puis tout ça, mais ça me permet de le concrétiser, tu sais ça le rend disponible, ou accessible je dirais plutôt » (S17,140-142). Il affirme

qu'il programme « par passion » (S17,97) et « des fois c'est professionnel » (S17,97-98). Il ajoute : « J'ai toujours ce côté amateur de vouloir m'améliorer, me développer » (S17,99-100).

Le sujet 17 discute du plaisir qu'il a à faire de la programmation informatique. Il décrit une de ses motivations :

> C'est tout le plaisir qu'on peut retirer de ça, […] créer quelque chose de, entre guillemets, toutes pièces, c'est sûr qu'il y a des kits qui existent puis ça simplifie la... déjà de le faire de ses mains puis de se rendre compte... on est habitué beaucoup dans notre monde à avoir tout cuit dans le bec, de pouvoir créer des choses comme ça puis d'aller plus loin. (S17,216-222)

4.18 Analyse individuelle du sujet 18

4.18.1 Engagement comportemental

Sur l'apprentissage de la programmation informatique, le sujet 18 affirme : « J'ai plus appris la programmation dans mes cours universitaires, c'est à partir de la première année universitaire où j'ai commencé par des algorithmes, apprendre à programmer en C, en C++, en Fortran » (S18,40-44). Il a par la suite utilisé « des logiciels comme Matlab où on utilise la programmation » (S18,45-46). Questionné sur ses premiers contacts avec l'informatique en général, il affirme : « À partir du secondaire déjà, je ne sais pas si je peux appeler ça programmation, bon il y avait certaines activités qu'on faisait un peu comme pour nous initier aux algorithmes » (S18,62-65).

Le sujet 18 décrit comment il envisageait la programmation à ses débuts : « Pour faciliter la vie et pour régler des choses de façon rapide, c'est comme ça que je percevais la programmation » (S18,96-98). Il réitère l'idée de rapidité : « On faisait des compilations en utilisant l'ordinateur pour réaliser rapidement certaines tâches » (S18,127-128). Il affirme :

> J'ai constaté que le passage d'un langage à un autre […] ce que moi j'ai constaté c'était juste comme on parle de langage français, anglais, [...] chacun a ses bases, ses mots, son dictionnaire et autre chose, mais le fondement est pareil. (S18,103-108)

Il ajoute que « le fondement est pareil, la façon d'aborder […] c'est à peu près la même chose » (S18,108-111).

Le sujet 18 raconte l'expérience de l'informatique qu'ont ses nièces :

> Mes nièces ont des tablettes, elles avaient ces tablettes-là [quand] elles avaient deux ans, il y avait certains petits éléments qu'ils (sic) devaient tout construire par exemple, tu sais pour construire une maison, ils avaient des différentes questions-là, tu vois, elles devaient aller étape par étape. (S18,668-673)

4.18.2 Engagement cognitif

Le sujet 18 donne un exemple d'utilisation de la programmation qu'il fait avec le logiciel Matlab : « Pour faire la triangularisation des matrices, donc on a des matrices là […] avec les nombres à l'intérieur, il faut mettre ça de façon triangulaire » (S18,398-401). Il utilise cet exemple pour supporter l'idée que « il ne faut pas avoir peur des mathématiques » (S18,403-404), mais ajoute que « tu n'es pas obligé d'avoir un niveau très avancé dans les mathématiques mais tu ne dois pas avoir peur des mathématiques et de la logique » (S18,404-406).

Le sujet 18 décrit l'implication de la base deux pour illustrer un niveau d'abstraction en informatique : « Par exemple généralement beaucoup de choses en informatique sont basées sur la base 2, le zéro et le un […] c'est un peu comme un niveau d'abstraction » (S18,333-337). Il ajoute que « pour pouvoir programmer, […] ça peut impliquer d'aller un peu plus souvent dans l'abstrait » (S18,338-340).

Le sujet 18 affirme avoir été initié à l'algorithmique sans ordinateur alors qu'il était au secondaire : « Dans mon cas particulier, on n'utilisait pas les outils [informatiques], donc on nous initiait juste de façon manuelle à programmer » (S18,76-78). Il donne l'exemple d'activités mathématiques où il était question de « créer un algorithme qui permet par exemple de trouver les résultats d'une suite » (S18,69-70).

Au sujet de l'essai-erreur, le sujet 18 affirme : « Moi ça m'est arrivé de procéder par essai-erreur lorsque j'ai cherché certaines choses ou que je voulais vérifier certains éléments, je fais des essais-erreurs puis je vois ce qui marche » (S18,519-522). Le sujet 18 affirme que « [si] tu n'as pas les connaissances abstraites que ça nécessite, essai-erreur, [...] ça va te mener nulle part » (S18,507-509).

En regroupant les mathématiques et la logique, le sujet 18 affirme : « Je m'étais rendu compte que plus ça évoluait [la programmation], plus ça demandait aussi de connaissances mathématiques, et aussi de la logique » (S18,390-392). Il se base sur cette observation pour affirmer : « Je suis vraiment d'accord qu'il ne faut pas avoir peur de la logique mathématique et être prêt à faire face à des frustrations » (S18,392-394).

4.18.3 Engagement affectif

Le sujet 18 raconte avoir déjà vécu des frustrations notamment lorsqu'un programme ne fonctionne pas : « Pour faire la triangularisation des matrices et là, quand tu compiles, ça bloque » (S18,443-444). Il ajoute que « il faut que tu revoies dans le programme qu'est-ce qui a marché, là souvent c'est juste peut-être une virgule ou un point que tu n'as pas mis quelque part » (S18,444-447). Selon lui, ce genre de situations, « ça peut te frustrer un peu » (S18,448).

Le sujet 18 raconte un changement dans ses motivations au fil du temps : « Quand j'étais au secondaire, aux premières années universitaires, je n'avais pas de motivation en particulier, mais actuellement par exemple j'ai

d'autres motivations qui vont en dehors par exemple du cadre scolaire » (S18,201-204). Il donne un exemple actuel : « Je suis en train de vouloir apprendre à programmer dans SAS, SAS c'est un logiciel [de] statistiques » (S18,205-206). Il résume ses motivations à programmer ainsi : « Je l'ai appris dans le cadre scolaire, et maintenant je suis motivé à l'apprendre seul par rapport à certains objectifs peut-être même professionnels et certains objectifs de recherche actuellement » (S18,237-241).

5 Discussion

Suite à l'analyse des entretiens réalisés auprès des dix-huit sujets, nous proposons ici une discussion visant à répondre à la question de recherche et à ses sous-questions. Cette discussion reprend des éléments abordés dans le cadre théorique et propose des explications supplémentaires à partir de la convergence observée entre les sujets que nous avons rencontrés. Pour rappel, la présente étude est exploratoire et ne prétend pas proposer des modèles explicatifs universels qui s'appliqueraient également à toute personne pratiquant l'activité de programmation informatique. Un retour systématique sera fait sur les trois sous-questions de recherche qui concernent l'engagement dans ses trois dimensions : comportementale, cognitive et affective. Par la suite, une synthèse de ces trois retours permettra de proposer une réponse à la question de recherche principale : Quelles sont les conditions supportant l'engagement dans l'activité de programmation informatique ? Finalement, ce sont les limites de la présente étude ainsi que les perspectives futures qui seront évoquées.

5.1 Réponse à la sous-question de recherche portant sur les manifestations de l'engagement comportemental

En vue de proposer des conditions qui supportent l'engagement comportemental dans l'activité de programmation informatique, nous nous affairerons ici à discuter des manifestations de ce type d'engagement chez nos sujets. Tel que précisé dans notre définition de l'engagement, l'engagement comportemental s'observe directement par des événements ou anecdotes. Dans ce contexte, il n'est donc pas question de la perception des individus ni du regard qu'ils portent sur des événements antérieurs, mais bien de la nature de ces événements tels qu'ils sont racontés.

5.1.1 Programmer pour soi-même ou programmer pour autrui

Les analyses des entretiens ont permis de dégager deux types de contextes dans lesquels la programmation informatique est pratiquée : (1) la programmation informatique réalisée à des fins personnelles, pour un projet à l'initiative de l'individu qui le réalise, ou bien (2) la programmation informatique réalisée pour autrui. Ces contextes sont différents notamment au niveau des raisons qui poussent une personne à faire de la programmation : la programmation informatique pour autrui fait intervenir des raisons extérieures à la personne, mais elle peut aussi faire intervenir de nouvelles raisons personnelles qui n'existeraient pas si la programmation était réalisée à des fins personnelles. Par exemple, plusieurs sujets ont affirmé que la réalisation de projets de programmation informatique pour autrui était une source de valorisation personnelle (p. ex., les sujets 01, 07 et 09).

Nous avons observé que la programmation pour autrui ne concerne pas que les contextes professionnels où le programmeur ou la programmeuse doit répondre aux besoins d'un client ou d'un employeur : elle englobe aussi tout projet, amateur ou professionnel, réalisé pour autrui. Par exemple, des sujets ont raconté avoir réalisé des programmes informatiques pour des membres de leur famille ou des amis (p. ex., des jeux vidéos ou des sites Web). Dans ces cas, des contraintes liées à la compréhension du besoin d'autrui interviennent au même titre que dans le cas de la programmation à des fins professionnelles. Dans le contexte de la programmation pour soi, plusieurs sujets ont évoqué le plaisir de créer, le sentiment de contrôle sur les outils qu'ils utilisent ainsi que les défis personnels qu'ils se donnaient eux-mêmes.

Nous avons observé à partir des 18 cas d'étude que l'importance donnée à la différence entre la programmation pour soi-même ou pour autrui varie entre les sujets. Pour certains, cette différence aurait peu d'incidence. Par exemple, le sujet 14 considère que c'est presque la même chose, la seule différence étant qu'il est parfois plus minutieux lorsqu'il programme pour lui-même. Pour d'autres comme le sujet 07, cette différence résonne dans toute la démarche, tout au long du processus et pas seulement en amont au moment de la compréhension du problème. Nous identifions donc que le destinataire de la programmation informatique, qui peut être soi-même ou autrui, est un paramètre qui peut initier différents rapports au savoir à prendre en compte afin de décrire ou d'inférer l'engagement comportemental. Dans le cas de la programmation informatique dont la seule finalité est de répondre aux exigences d'un travail scolaire, un cas évoqué par quelques sujets (p. ex., la sujet 05), nous considérons que le destinataire est autrui.

5.1.2 Apprentissage hétérostructuré ou autostructuré de la programmation informatique

Les propos de nos sujets en lien avec la façon dont ils ont appris la programmation informatique nous amènent à mobiliser les distinctions de Prévost (1994) entre l'autoformation et l'hétéroformation que nous avons discutées dans le cadre théorique.

La majorité de nos sujets ont rapporté avoir appris en empruntant chacune de ces voies en différents moments de leur parcours et dans des proportions variables. Les sujets 02, 03, 12, 13, 14 et 18 ont vécu une formation que nous considérons comme presque exclusivement hétérostructurée, alors que le sujet 10 a vécu une formation que nous considérons comme presque exclusivement autostructurée. Les autres sujets ont vécu une formation à la fois hétérostructurée et autostructurée, selon le contexte.

Pour plusieurs sujets, l'apprentissage en contexte scolaire a été présenté comme une façon de combler des lacunes qu'ils avaient eux-mêmes constatées dans leur pratique. En ce sens, bien qu'il soit structuré par un programme scolaire, il s'inscrit dans une démarche d'apprentissage autostructurée. Par exemple, le sujet 03 affirme être retourné à l'université car il avait constaté des lacunes en mathématiques et qu'il cherchait à les

combler. Le sujet 09 affirme que certains apprentissages, par exemple des règles de logique, sont enseignés explicitement en contexte scolaire. Dans le cas de l'apprentissage scolaire, l'engagement de nos sujets s'observe principalement par leur choix d'adhérer à un programme de formation intégrant l'apprentissage de la programmation informatique. Il s'agissait principalement de programmes collégiaux ou universitaires, les quelques exceptions étant les cours optionnels offerts au niveau secondaire. En lien avec la problématique, nous constatons que l'engagement comportemental consistant à entamer des démarches pour s'inscrire à un cours de programmation informatique ne pourrait pas s'observer si cet apprentissage était rendu obligatoire pour tous et toutes. Nous conclurons en disant que nos sujets sont tous engagés dans une démarche d'autoformation, un concept qui…

> … sert à évoquer des pratiques de formation que l'on pourrait situer sur un continuum tendu entre les pratiques les plus hétérostructurées, telles que l'enseignement individualisé ou l'EAO [enseignement assisté par ordinateur], et les pratiques les plus autostructurées, telles que les pratiques autodidaxiques. (Albero, 2002, pp. 467–468)

5.1.3 La programmation informatique comme finalité ou comme outil : vers un rapport au savoir de création

Chaque sujet que nous avons rencontré a appris et utilisé plus d'un langage de programmation. Il a été observé que certains d'entre eux ont manifesté plus d'engagement avec certains langages, notamment en raison du type de réalisation que chacun permet. Par exemple, les sujets 04, 09, 10 et 11 ont relaté avoir utilisé des langages de programmation Web afin de produire des sites Web qui leur servaient de moyen de communication. Dans ce cas-ci, il apparaît que l'engagement est suscité par l'objet de réalisation plutôt que par les langages. Pour d'autres, les langages de programmation en tant que tels peuvent représenter un motif d'engagement. Par exemple, le sujet 16 affirme avoir appris le langage Python par pur loisir sans autre finalité.

Nous établissons ainsi une distinction entre la programmation informatique réalisée de façon instrumentale et la programmation réalisée pour elle-même. Cette distinction s'observe de différentes façons dans les entretiens. L'expression « informatique d'affaires » (S14,53) utilisée par le sujet 14 permet peut-être d'expliquer cette distinction entre une informatique plus fondamentale et une informatique plus appliquée, laquelle serait instrumentale à quelque chose d'autre. Cela s'approche des propos du sujet 13 lorsqu'il établit une distinction entre les personnes qui sont « orientées [vers l']utilisateur » (S13,435-436) et celles qui sont « orientées [vers le] développeur » (S13,437). Cette distinction pourrait expliquer la divergence de points de vue entre les sujets quant au rapprochement entre la programmation et l'art. Certains des sujets qui font ce rapprochement parlent de la programmation comme d'une finalité notamment en racontant réaliser des projets sans but précis, pour leur seul loisir (p. ex., le sujet 16). Le sujet 04 s'oppose à ce rapprochement, alors que les sujets 01, 07, 12, 13 et 15 le font notamment sur la base de la démarche de création, de l'aspect visuel du code et des optimisations

possibles. L'idée de la programmation entendue comme finalité se rapproche aussi des travaux de Paloque-Bergès (2009) qui présentent plusieurs façons d'envisager le code informatique comme une expression artistique. Si la programmation est entendue comme outil plutôt que comme finalité, alors la valeur créative se situera ailleurs que dans le code, dans le rendu visuel d'un film par exemple. Étant donné les dimensions affectives pouvant être associées à l'art, nous discuterons davantage de ce rapprochement en lien avec l'engagement affectif.

Ainsi, nous constatons que les sujets qui envisagent la programmation informatique comme une finalité entretiennent tous un rapport au savoir de création. Pour les autres qui l'envisagent davantage comme un outil de création plutôt que comme une finalité, le rapport au savoir peut être soit de l'ordre de la création ou bien de la consommation en fonction de ce que la personne réalise à l'aide de la programmation informatique. Du point de vue de l'activité de programmation informatique, si elle est utilisée pour réaliser des programmes peu originaux dans le but d'accomplir une tâche, sans investissement personnel et émotif, alors le rapport au savoir est davantage de l'ordre de la consommation. Cela n'exclut pas que d'un autre point de vue, la personne puisse entretenir un rapport au savoir de création – par exemple du point de vue de la communication par un site Web. Le programme informatique derrière ce site Web peut avoir une faible valeur créative, ce qui n'exclut pas que l'usage que la personne fait de ce site Web peut être fortement créatif. La personne est alors consommatrice d'outils de programmation informatique et les utilise d'une façon non créative dans le but de supporter autre chose qui pourra être envisagé de façon créative. Cette idée rejoint Romero (2018) pour qui le potentiel pédagogique de la programmation est plus grand dans des activités créatives. Cela rejoint aussi le modèle de Romero, Laferriere et Power (2016) selon lequel les activités technocréatives peuvent susciter un engagement variable suivant qu'elles sont de l'ordre de la consommation, de l'interaction, de la création, de la co-création ou de la co-création participative. Si elle est envisagée comme outil davantage que comme finalité, la programmation informatique peut, ou non, faire intervenir ce rapport créatif : cela dépend si la personne envisage les programmes qu'elle construit comme des expressions artistiques ou non et si elle utilise ces programmes comme support à une autre activité créative.

La Figure 3 de la page 101 présente une synthèse de nos observations sur l'engagement comportemental. Elle illustre trois axes à partir desquels peuvent être décrits les contextes dans lesquels la programmation informatique est pratiquée par nos sujets. Ces axes doivent être envisagés comme des continuums. Ainsi, certains contextes se situent quelque part entre l'apprentissage hétérostructuré et l'apprentissage autostructuré, entre un rapport au savoir de création et un rapport de consommation, et entre la programmation pour soi-même et la programmation pour autrui. Prenons l'exemple d'une personne qui crée un programme informatique visant à gérer automatiquement l'éclairage de son domicile. Dans ce contexte, le destinataire de la programmation informatique est la personne elle-même; le projet est réalisé pour soi. Le programme informatique est ici

présenté comme un outil plutôt que comme une finalité, car il permet de réaliser un système d'éclairage informatique. Malgré cela, il est possible que la personne envisage son programme informatique comme une création, par exemple si elle passe du temps à en soigner le code, si elle cherche à l'optimiser et si elle en est personnellement fière. Cela témoignerait d'un rapport de création plutôt que d'un rapport de consommation. Autrement, si la personne n'envisage la programmation informatique que comme un outil permettant de parvenir au système d'éclairage automatique, elle se situerait davantage du côté de la consommation – son programme étant alors davantage un assemblage d'instructions peu original.

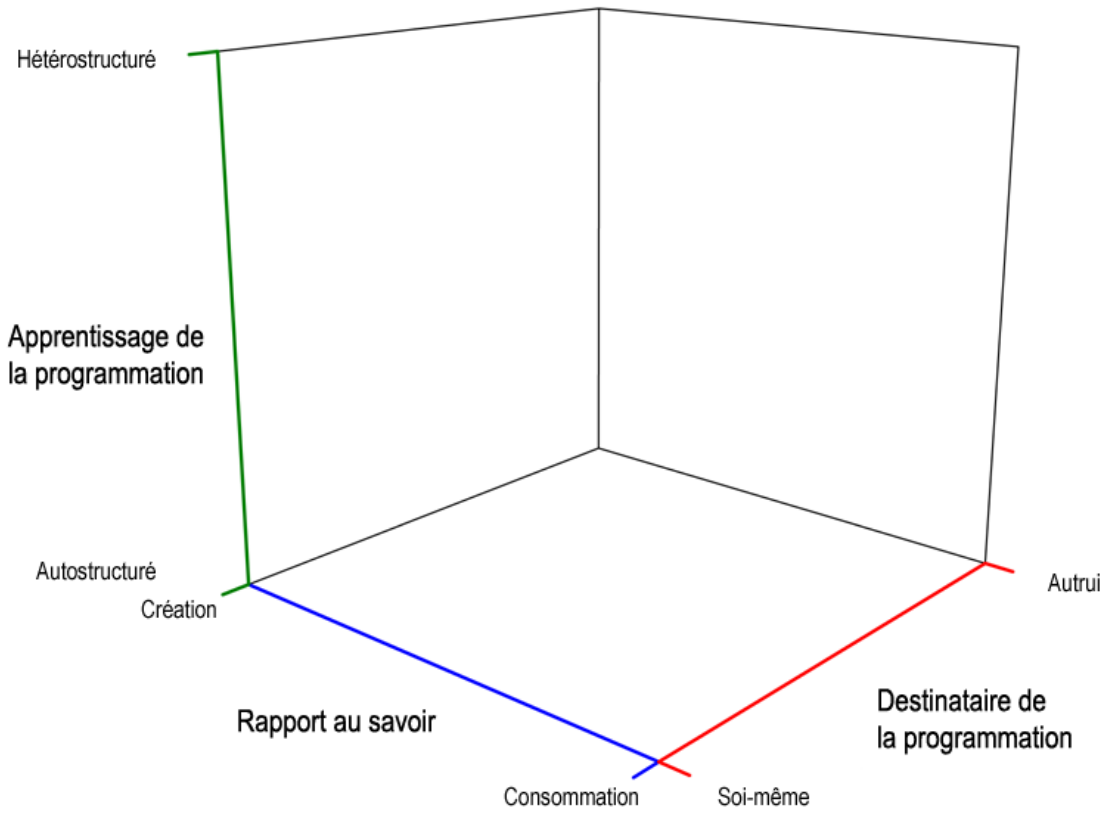

Figure 3. Trois axes pour décrire la pratique de la programmation informatique

5.1.4 L'adoption de différentes postures épistémologiques par nos sujets

À plusieurs reprises et sous différentes formes, nous avons observé une variété de postures épistémologiques dans les propos de nos sujets. À des fins d'appui empirique, il nous apparaît essentiel d'en donner ici quelques exemples.

Le sujet 04 affirme que le code informatique ne sert à rien en soi. En faisant cette affirmation, il considère que le code est un moyen pour parvenir à quelque chose plutôt qu'une finalité. Il se prononce contre l'apprentissage pour tous et toutes de la programmation informatique en tant que finalité. Toutefois, en tant que moyen pour

comprendre les interactions entre les personnes via des outils numériques de communication, il soutient que la programmation peut avoir un intérêt pour tous et toutes. Cela rejoint en partie les propos d'autres sujets voulant que l'informatique en tant que telle ne soit pas grand-chose jusqu'à ce qu'elle soit appliquée à un domaine de connaissance. Dans ce cas-ci, il nous apparaît que le sujet a modifié sa posture épistémologique au fil du temps face à la programmation informatique. Le début de son parcours, tel qu'il nous l'a raconté, laisse comprendre qu'il a déjà pratiqué la programmation en tant que finalité alors qu'il travaillait à la création de sites Web. En ce sens, les situations-problèmes auxquelles il était confronté impliquaient de facto le recours à la programmation informatique. Nous considérons ainsi qu'il adoptait une posture passive face à certains paramètres de la situation-problème, ne remettant pas en cause le recours à la programmation informatique. Depuis quelques années, nous considérons qu'il a modifié cette posture passive face au problème pour développer un rapport dans lequel il remet en question les éléments donnés du problème, comme le choix de recourir ou non à la programmation informatique. Il attribue ce changement à sa découverte de la discipline du design qui lui a fait prendre conscience des écosystèmes, de leur imbrication et des interactions qui s'y passent. À plusieurs reprises, le sujet 04 parle de « révélations » (S04,827) pour expliquer les causes de son passage de la programmation informatique vers le design. Cela nous amène à renforcer l'idée selon laquelle il a modifié la posture épistémologique à partir de laquelle il envisageait les problèmes qu'il résolvait par la programmation informatique auparavant.

La pratique de la programmation informatique par le sujet 05 s'inscrit dans le mouvement *Maker*. La programmation est instrumentalisée pour cheminer vers la construction d'artefacts (avec le nanoordinateur Raspberry Pi et le microcontrôleur Arduino). Dans ce contexte, une posture épistémologique est induite par la situation en raison de la culture de création de ce mouvement. Cette posture peut s'entremêler avec d'autres postures, par exemple celle induite par le langage de programmation utilisé qui impose un autre lot de contraintes à prendre en considération. Le choix de ce langage est alimenté par des paramètres ou contraintes qui émergent en adoptant la posture épistémologique induite par la situation.

Nous constatons que le sujet 06 avait déjà une attitude favorable à l'apprentissage de la programmation informatique alors qu'il était à l'école primaire. Lorsqu'il évoque la curiosité qui motivait ses actions, ses propos rejoignent ceux de la sujet 05 pour qui le besoin d'un apprentissage doit être ressenti par l'apprenant ou l'apprenante. Le sujet 06 évoque aussi un contexte parfois favorable à cet apprentissage, parfois défavorable (il nomme, par exemple, que le personnel de son école secondaire n'était pas en mesure de l'aider dans ses projets). Cela peut peut-être s'expliquer par une forme de résilience entendue comme une attitude menant à des actions pourtant découragées par le milieu. Il importe de préciser que le sujet fréquentait aussi des milieux où l'informatique était encouragée comme la cellule familiale et certains de ses amis, par exemple. Il apparaît ici que le sujet entretenait au moins dès le primaire un rapport actif face au savoir et qu'il avait déjà conscience

de sa capacité d'action sur son propre apprentissage. Par rapport actif face au savoir, nous entendons qu'il s'autorisait à adopter chacune des quatre postures épistémologiques en vue de créer des solutions à des problèmes.

Ces quelques exemples de l'adoption de postures épistémologiques variées par nos sujets nous amènent à proposer qu'une part de l'activité de programmation informatique consiste en une forme de médiation entre différents rapports au savoir qui peuvent intervenir parfois simultanément. Ainsi interprétée, la programmation pourrait donner lieu à un engagement comportemental associé à la mouvance d'un individu entre les postures épistémologiques pour aborder un problème. Cette agilité peut peut-être davantage s'expliquer par l'activité cognitive de l'individu. La prochaine section vise ainsi à répondre à la sous-question de recherche portant sur les manifestations de l'engagement cognitif. À partir des résultats observés chez nos sujets, nous tenterons de décrire l'activité cognitive qui a lieu lorsqu'un individu aborde un même problème depuis différentes postures épistémologiques.

5.2 Réponse à la sous-question de recherche portant sur les manifestations de l'engagement cognitif

Nous avons présenté l'engagement cognitif disant qu'il s'observe plus particulièrement par le recours à des stratégies personnelles d'autorégulation. Plus spécifiquement, étant donné que les définitions de la pensée informatique convergent vers le processus d'abstraction, nous avons eu recours au concept d'abstraction réfléchissante pour étudier ce que peut être le travail cognitif en programmation informatique. Les analyses ont permis de construire une proposition touchant à l'engagement cognitif : la démarche créative en programmation informatique implique une interaction entre action et réflexion.

5.2.1 La marge créative en programmation informatique

Le sujet 07 amène une idée nouvelle dans le cadre de nos analyses, celle que le programmeur ou la programmeuse doit simultanément réfléchir de différentes façons (il parle de « top-down » (S07,543) et « bottom-up » (S07,543)[7]). Ces propos, présentés en analyse individuelle pour le sujet 07, nous apparaissent comme fédérateurs de ce que d'autres sujets ont exprimé au niveau de la résolution de problèmes en programmation informatique. Par exemple, le sujet 03 a décrit la programmation en binôme comme une démarche où une personne écrit le code alors qu'une seconde révise au fur et à mesure tout en gardant en tête

[7] Les termes *top-down* et *bottom-up* sont utilisés ici pour reprendre les propos du sujet 07. Ils ont une acception différente en psychologie cognitive où *top-down* désigne ce qui vient de l'individu et *bottom-up* ce qui vient de l'environnement. Nous nous efforçons dans la suite du texte de nous distancer de ces termes pour éviter la confusion.

la vision globale du projet. Le sujet 07 suggère qu'un programmeur ou une programmeuse qui travaille seul doit adopter ces deux points de vue simultanément. Le sujet 07 va plus loin et affirme que l'objectif d'une telle démarche vise à ce que les deux points de vue, *top-down* et *bottom-up*, se rencontrent pour créer une solution au problème.

Ce que le sujet 07 décrit comme une façon de penser *top-down* a été évoqué par d'autres sujets sous différentes appellations; principalement il s'agit de l'analyse conceptuelle d'un problème réalisée en amont de l'écriture du code informatique. Notons que cette réalisation en amont peut survenir d'une manière itérative, ce pour quoi elle peut aussi être réfléchie comme étant en aval. Nous proposerons l'expression *réflexion descendante* afin d'illustrer que cette façon de réfléchir, poussée à l'extrême, consiste à penser un problème de façon entièrement intellectuelle, sans appui empirique. Cette réflexion descendante s'apparente ainsi à une vision rationaliste de la connaissance, c'est-à-dire qu'elle donne lieu à une représentation de la situation issue du raisonnement de l'individu.

Ce que le sujet 07 décrit comme une façon de penser *bottom-up* a aussi été évoqué de plusieurs façons par d'autres sujets. Plusieurs ont avancé le fait que l'écriture de code informatique (et la compilation ou l'exécution de ce code) les aidait à améliorer leur compréhension du problème et de la solution (p. ex., les sujets 03, 06 et 07). Cette idée rejoint la proposition plus générale de Jonassen (2006) selon laquelle un ordinateur peut agir comme un outil cognitif notamment au niveau de la modélisation d'un problème. Comme cette façon de penser correspond à l'action d'écrire du code informatique, et que cette action fait émerger une nouvelle compréhension de la situation, nous proposerons l'expression *action ascendante* pour la décrire. Cette action ascendante correspond ainsi à une vision empirique de la connaissance, c'est-à-dire que la compréhension du problème passe par l'exécution ou la compilation de code informatique, résultat de l'action d'un individu dans ce cas-ci. Dans cette perspective, nous pouvons considérer cette action ascendante comme une manifestation d'abstraction empirique.

Les entretiens font émerger non seulement deux façons de penser, mais surtout une interaction soutenue entre les deux tout au long de la démarche créative de programmation informatique. Cela correspond aux observations plus générales de Papert (1992) pour qui l'usage de l'informatique implique une interaction entre l'abstrait et le concret. Cette alternance entre la réflexion et l'action se fait en parallèle avec une alternance entre les manipulations abstraites et les manipulations concrètes de la création. Ainsi, l'abstraction réfléchissante, processus cognitif, est constamment alimentée par l'abstraction empirique, issue du résultat de l'action du sujet. C'est ce que nos sujets ont évoqué en disant que la rétroaction de l'ordinateur lors de la compilation ou de l'exécution d'un programme les aidait à mieux comprendre une situation.

Si nous acceptons le point de rencontre entre la réflexion descendante et l'action ascendante comme une création répondant à un besoin, alors nous proposons que cette création soit le fruit de deux postures épistémologiques incompatibles. La première, celle de la réflexion descendante, s'approche du rationalisme en ce qu'elle propose une interprétation de la situation dirigée exclusivement par la pensée du programmeur ou de la programmeuse. La seconde posture épistémologique, celle de l'action ascendante, relève de l'empirisme, car elle est guidée exclusivement à partir du « tangible » (S07, 840), résultat de l'action dans ce cas-ci. Nous entendons ces postures épistémologiques comme incompatibles, en tout ou en partie, en raison des différents rapports au savoir que nous avons présentés dans le cadre théorique. Cette tension semble génératrice de l'émergence de la marge créative de l'activité de programmation.

La solution à un problème se situe alors quelque part entre le point de départ de l'action ascendante et celui de la réflexion descendante. Nous proposons que les différences observées entre les sujets quant à la place de l'analyse et à la place du bricolage peuvent s'expliquer par la variété d'approches possibles par rapport à ces deux points de départ : une personne peut réaliser la réflexion descendante de façon plus soutenue avant de passer à l'action, alors qu'une autre peut tout de suite passer à l'action afin d'alimenter sa réflexion.

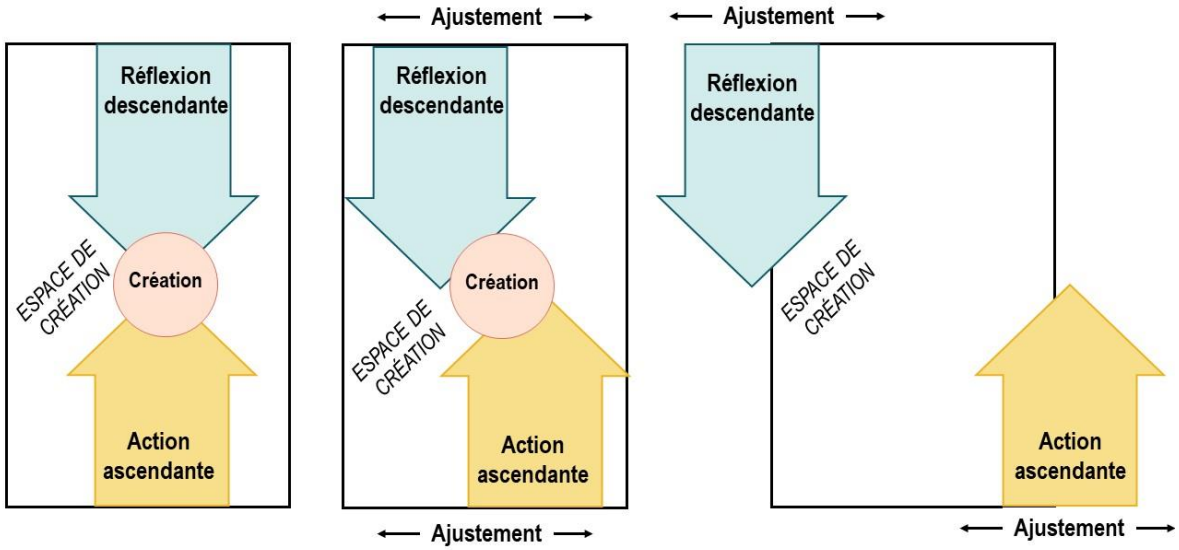

Figure 4. La marge créative en programmation informatique

La Figure 4 illustre la marge créative en programmation informatique telle que nous la comprenons à la lumière de nos analyses. Elle implique deux activités simultanées du sujet : (1) une réflexion descendante qui part de la raison et de l'analyse d'un problème, puis (2) une action ascendante qui part de l'écriture de code informatique et de son exécution. Le point de rencontre entre ces deux activités constitue la création et se situe quelque part dans l'espace de solution, c'est-à-dire à l'intérieur des critères d'acceptation de la solution (p. ex., tel que discuté par le sujet 09). La rencontre entre la réflexion descendante et l'action ascendante peut être partielle, ce qui fait apparaître un espace d'ajustement de l'action et de la réflexion, dans lequel l'individu doit naviguer afin de construire la création. Finalement, la réflexion descendante ou l'action ascendante peuvent être éloignées au point où même des ajustements ne permettent pas leur rencontre. Dans ce cas, la solution est impossible à l'intérieur de l'espace de solution, il faut soit reprendre l'action et la réflexion ou revoir les critères d'acceptation de la solution. En réponse à la sous-question de recherche portant sur l'engagement comportemental, nous avons proposé que la programmation peut être envisagée comme une activité de médiation. Nous rattacherons la marge créative à ce que nous avons dit à ce sujet en ajoutant que cette médiation s'opère par un travail d'ajustement entre la réflexion descendante et l'action ascendante.

5.2.2 Les champs conceptuels en programmation informatique

Les entretiens réalisés nous ont amené à constater que l'activité de programmation informatique peut être déclinée en de nombreuses variantes, rendant difficile l'identification de champs conceptuels associés à la programmation informatique tout entière. Les langages peuvent varier et avoir une incidence sur la

compréhension des concepts nécessaires à la démarche créative, de même pour les différents paradigmes de programmation et les finalités recherchées par la personne. Pour cette raison, notre étude exploratoire ne permettait pas d'identifier des champs conceptuels. Les questions abordées avec les sujets n'étaient pas suffisamment précises pour permettre d'identifier de façon systématique les concepts qu'ils et elles mobilisaient en pratiquant l'activité, et encore moins les champs conceptuels. La recherche nous conduit à affirmer qu'il serait précipité de définir des concepts communs à toute forme de programmation informatique. Malgré tout, cette étude exploratoire nous amène à considérer que ces concepts communs peuvent exister étant donné que certains d'entre eux apparaissent dans le discours de la majorité des participants comme les boucles, les itérations, les structures conditionnelles ou bien les variables. La démarche créative que nous avons présentée pourrait servir de base pour construire des activités créatives de programmation informatique requérant la compréhension de ces concepts. L'étude des créations ainsi réalisées, combinée à une verbalisation de la démarche par la personne, pourrait permettre d'identifier des champs conceptuels composés de formes langagières et non langagières, d'invariants opératoires et de concepts-en-acte.

5.3 Réponse à la sous-question de recherche portant sur les manifestations de l'engagement affectif

5.3.1 La convergence des motivations en programmation informatique

Plusieurs de nos sujets ont évoqué l'importance des motivations intrinsèques afin de persister dans l'apprentissage de la programmation informatique. En ce sens, la sujet 05 parle du « besoin d'aller plus loin » (S05,202) ressenti par l'individu, une idée qui est reprise par de nombreux autres sujets en différents termes. Plusieurs ont aussi discuté des motivations extrinsèques notamment en contexte professionnel où le développement d'un produit est orienté par les besoins d'un client ou d'un employeur. En ce sens, les sujets 01, 03, 12 et 15 ont insisté sur l'importance de comprendre ce besoin pour créer une solution qui répond aux critères d'acceptation. Nous dirons ces motivations organisationnelles puisqu'elles relèvent de l'organisation et non de l'individu.

Plusieurs analyses nous amènent à considérer qu'une des conditions à l'engagement en programmation informatique est qu'il doit y avoir convergence ou à tout le moins cohérence entre les motivations individuelles, qui peuvent être dites intrinsèques, et les motivations organisationnelles, qui peuvent être dites extrinsèques. Cela pourrait expliquer pourquoi le sujet 07 parle du besoin d'amadouer le programmeur ou la programmeuse en vue de l'amener à collaborer à des intérêts qui ne sont pas les siens. Lorsque les motivations organisationnelles sont les mêmes que les motivations individuelles, alors il y a convergence et dans ce contexte, nous constatons que nos sujets ont persisté dans leur emploi, scolarité ou tâches, par exemple.

Lorsque les motivations individuelles et organisationnelles sont différentes sans toutefois entrer en conflit, alors il peut y avoir engagement aussi. Nous considérons que c'est le cas du sujet 12 qui explique qu'il dispose d'une marge créative ou d'un espace de liberté à l'intérieur des balises fournies par l'organisation pour laquelle il travaille. Dans d'autres cas, où il y a confrontation des motivations individuelles et des motivations organisationnelles, nous constatons que certains sujets ont été amenés à changer de tâches ou de fonction. Par exemple, le sujet 15 a expliqué avoir quitté ses fonctions de programmeur pour occuper un poste de gestionnaire afin d'être impliqué dans le processus décisionnel plus qu'il ne l'était en tant que programmeur.

Nous rappelons que la présente étude consistait en des entretiens individuels réalisés auprès de programmeurs et programmeuses informatiques. La rencontre de nombreux acteurs – autres que les programmeurs et programmeuses – des organisations permettrait de parvenir à une meilleure compréhension de l'interaction entre les motivations personnelles et organisationnelles et d'obtenir d'autres points de vue que ceux de nos sujets.

5.3.2 Le rapport à l'erreur et à la complexité en programmation informatique

Les analyses des entretiens nous ont amené à considérer que nos sujets entretenaient des rapports affectifs différents face à la complexité et à l'erreur. Plusieurs d'entre eux ont évoqué que ce rapport avait changé au fil du temps et de leurs expériences. Par exemple, nous constatons que le rapport à l'erreur du sujet 09 s'est transformé avec l'expérience. Au début de son parcours, le sujet décrit l'erreur comme source de frustration. Puis, par la suite, il la décrit comme quelque chose de positif permettant d'améliorer la solution qu'il crée à un problème. Il donne plusieurs exemples en lien avec les erreurs de compilation ou d'exécution d'un code informatique. Encore une fois, cela rejoint l'idée générale de Jonassen (2006) selon laquelle l'usage de l'informatique peut devenir un outil cognitif.

Lorsqu'ils ont été questionnés sur les frustrations en programmation informatique, nos sujets se classent de façon dichotomique. Pour certains d'entre eux, les frustrations font partie intégrante de la programmation informatique. D'autres, à l'inverse, considèrent que les frustrations dépendent de l'émotivité d'un individu. Certains suggèrent alors qu'une personne vivant des frustrations n'est peut-être pas au bon endroit en programmation informatique. Cela laisse une question ouverte : la frustration en programmation informatique émane-t-elle des situations ou des individus qui la pratiquent ? À partir des idées de Weinberg (1998) sur la place de l'ego en programmation informatique, nous proposerons que cela dépend de la proximité émotive qu'une personne entretient avec les programmes qu'elle réalise. À ce sujet, Weinberg affirme :

> Même si [les programmeurs] sont détachés des personnes, ils sont attachés à leurs programmes. En fait, leurs programmes deviennent souvent des extensions d'eux-mêmes […]. A priori, il peut sembler que le jugement de l'ordinateur est indiscutable, et si ce l'est, l'attachement d'un programmeur envers ses programmes peut avoir de sérieuses conséquences sur son estime personnelle. Quand l'ordinateur trouve un bogue dans son programme, le programmeur peut raisonner environ comme cela : « Ce programme est erroné. Ce programme fait partie de moi, une extension de moi-même […]. Je suis erroné. » (p. 53-54, traduction libre)

Ce profil peut correspondre à certains de nos sujets (p. ex., les sujets 07 et 08), notamment ceux qui ont exprimé par des anecdotes avoir été affectés personnellement par des difficultés rencontrées ou le jugement de leurs pairs en lien avec des programmes qu'ils réalisaient. Il ne correspond toutefois pas à tous nos sujets avec la même force. Par exemple, le sujet 01 considère ses programmes comme des créations originales, mais affirme explicitement ne pas vivre de frustrations liées à des difficultés rencontrées.

Nous comprenons que le sujet 06 a développé au fil du temps une attitude d'ouverture face à la complexité des problèmes en informatique. Cela rejoint l'idée de tolérance à l'ambiguïté qui consiste à « travailler dans des contextes où toutes les informations et consignes ne sont pas disponibles » (Romero, 2016, para. 6). Cela peut être mis en parallèle avec ce que d'autres sujets ont évoqué au sujet des frustrations. Par exemple, l'entretien avec le sujet 05 nous amène à considérer que la découverte de la complexité d'un problème peut mener, entre autres, à (1) des frustrations ou (2) un ajustement des représentations du problème, ce qui pourrait correspondre aux processus d'assimilation ou d'accommodation (Piaget, 1977) évoqués dans le cadre théorique. Cela suggère que la compréhension abstraite d'un problème passe d'abord par une attitude d'ouverture face à la complexité. Nous constatons qu'à défaut d'une telle attitude, l'individu peut vivre des frustrations. Plusieurs sujets ont parlé de ces frustrations comme des obstacles à la persévérance. Nous proposons que de telles frustrations peuvent aussi être des obstacles à la transformation du rapport au savoir vers un rapport de création plutôt que de consommation.

Nous en venons donc à proposer que les frustrations en programmation informatique puissent être des réactions émotives symptomatiques – ou non – d'une attitude de fermeture face à la complexité d'un problème. En ce sens, il y aurait intolérance à l'ambiguïté. Cette proposition doit être considérée avec prudence, car nos analyses convergent vers l'idée que les frustrations peuvent aussi être attribuables à l'émotivité des individus plutôt qu'à cette attitude de fermeture. Cela nous amène donc à proposer que les frustrations en programmation informatique ne suffisent pas à conclure à une attitude d'ouverture ou de fermeture face à la complexité, mais qu'elles peuvent justifier une investigation approfondie. Par exemple, la compréhension du manque d'engagement affectif chez un individu pourrait être enrichie par l'étude de la nature des frustrations qu'il vit, s'il y a lieu.

5.3.3 Les interactions interpersonnelles en programmation informatique

Nos sujets ont discuté des interactions interpersonnelles qui avaient lieu dans le cadre de leur pratique de la programmation informatique. Ces interactions touchent au travail d'équipe, à l'utilisation de la programmation pour communiquer avec des personnes ou à la participation à des communautés de pratique.

Le travail d'équipe a été évoqué par plusieurs de nos sujets. Plusieurs ont souligné que les structures organisationnelles imposent parfois de séparer la programmation informatique en plusieurs sous-tâches pour répondre à des impératifs de division du travail. Certains comme les sujets 01 et 07 nous amènent à considérer que cette division du travail peut dénaturer la tâche de programmation informatique en lui retirant une forme de cohérence interne qui est une source de motivation. Par exemple, le sujet 03 est motivé par le fait qu'il doive regarder un problème sous plusieurs angles différents, un élément qui peut être altéré ou dénaturé si le problème est découpé par avance. Certains ont aussi présenté le travail d'équipe comme une possible source de frustration (p. ex., le sujet 14). Toutefois, les analyses nous amènent à considérer que le travail d'équipe n'est pas une source de frustration spécifique à la programmation informatique; il s'agit plutôt d'une possible source de frustration qui peut s'appliquer au travail d'équipe dans plusieurs activités. Il nous apparaît utile de distinguer ici l'équipe de travail, l'organisation et la communauté de pratique. L'équipe de travail implique de travailler sur un projet collectif, orienté à la fois par des motivations personnelles et organisationnelles, alors que la communauté de pratique exerce son influence par l'adhésion volontaire d'un individu. En ce sens, les contraintes reliées au travail d'équipe émergent en adoptant la posture induite par la situation davantage qu'en adoptant celle induite par la communauté de pratique.

L'analyse de l'entretien avec le sujet 09 amène à envisager le code informatique comme un outil sémiotique permettant de communiquer avec d'autres membres d'une communauté de pratique. De cette façon, un groupe de personnes peut parvenir à une compréhension négociée de certains concepts. Par exemple, le sujet 09 indique qu'une des bonnes pratiques en programmation consiste à écrire un code suffisamment clair pour être compris par quelqu'un ayant un bagage similaire à la personne qui l'écrit. Dans cet exemple, le concept de *clarté* a été négocié entre des individus dans une communauté de pratique ou de façon plus globale dans une culture informatique. Plusieurs de nos sujets comme les sujets 01, 03, 13 et 15 amènent aussi l'idée que des outils de visualisation peuvent aider à communiquer des concepts entre des personnes ayant des bagages différents ou des connaissances dans des domaines différents. L'utilisation de ces outils, parmi lesquels les langages comme

UML[8], pourrait être étudiée davantage à partir de l'engagement tel que défini par Naps et al. (2003). Pour l'apprentissage de la programmation chez les enfants, il existe des langages ou des outils de programmation visuelle comme le langage Scratch. Ces langages revêtent certaines propriétés des langages de modélisation en raison des aides graphiques comme l'utilisation de blocs et de formes variés.

La participation à des communautés de pratique s'observe dans les entretiens avec la plupart de nos sujets. Parfois, ces communautés de pratique sont plus explicites (p. ex., les communautés de logiciel libre auxquelles participe le sujet 07) alors que parfois elles sont plus implicites (p. ex., le sujet 06 évoque trouver des réponses à ses questions sur le site StackOverflow). Cela peut s'expliquer par la théorie des communautés de pratique de Wenger (1998) : certains individus se situent davantage au centre de la communauté et sont actifs dans la vie de la communauté alors que d'autres se situent en périphérie et sont davantage passifs dans leur participation à la communauté. Comme nous l'avons proposé, la communauté de pratique peut induire une posture épistémologique que le programmeur doit adopter afin de faire émerger des contraintes à considérer dans la création d'une solution. Ces contraintes doivent être harmonisées avec d'autres, par exemple celles de la situation. Le sujet 15 donne l'exemple de contraintes en lien avec les limites financières imposées par une organisation. Il nous apparaît pertinent de questionner le poids relatif de chacune de ces postures épistémologiques face aux autres selon les contextes; dans le cas évoqué par le sujet 15 il nous apparaît que le poids relatif de la situation-problème est plus élevé que celui de la communauté de pratique étant donné que la décision finale revient à l'organisation plutôt qu'à l'individu.

Nous proposons que l'action de médiation du programmeur ou de la programmeuse dans un environnement peut être compromise lorsqu'il ne lui revient pas de déterminer comment les contraintes émergeant de différentes postures épistémologiques s'agencent entre elles. Cette implication de l'individu dans la modélisation du problème constitue un outil pour la création d'une solution. Toutefois, nos analyses nous amènent à considérer que cet outil n'est pas toujours à disposition de l'individu selon la structure décisionnelle de l'organisation. Selon plusieurs sujets, cela peut être une source de frustration. Cela rejoint aussi les propos de Weinberg (1998) selon qui une équipe de programmation peut éprouver des difficultés si le gestionnaire ne partage pas l'expertise de ses membres :

[8] Le langage UML (Unified Modeling Language) est un langage visuel de schématisation utilisée en informatique pour représenter des solutions informatiques au niveau conceptuel. Il est fréquemment utilisé pour clarifier le besoin avec un client ou une équipe de travail.

> Quand un nouveau gestionnaire qui n'est pas programmeur prend en charge une équipe de programmation, des problèmes peuvent arriver à moins que le gestionnaire reconnaisse explicitement sa méconnaissance des aspects techniques. […] Même si une équipe peut finir par respecter un gestionnaire qui admet ouvertement son manque d'expérience en programmation, ils ne peuvent que ridiculiser quelqu'un qui prétend en avoir et qui au final s'avère incompétent. (p. 80)

5.4 Synthèse de la discussion

En guise de synthèse à cette discussion des résultats, nous proposerons ou enrichirons deux interprétations de l'activité de programmation informatique appuyées par nos résultats : la programmation informatique comme art ou la programmation informatique comme activité de médiation ontologique. Ensuite, nous identifierons des points de convergence ou d'opposition entre nos observations et certaines études relevées dans la problématique et le cadre théorique. Finalement, nous fournirons une réponse à la question de recherche à la lumière de la discussion.

5.4.1 La programmation informatique comme art

La programmation informatique (et l'informatique de façon plus générale) et l'art sont parfois abordés de façon complémentaire dans le domaine de l'éducation. L'approche *STEAM*[9] par exemple vise à aborder de façon multidisciplinaire les sciences, la technologie, l'ingénierie, les arts et les mathématiques.

Plusieurs de nos sujets ont établi un rapprochement entre la programmation et l'art; c'est le cas des sujets 01 et 07 par exemple. D'autres ont se sont explicitement prononcés contre un tel rapprochement comme c'est le cas des sujets 04 et 10. Ici, nous considérons que ces propos reflètent des perceptions individuelles de la programmation informatique et qu'en ce sens ils ne sont pas à considérer comme contradictoires. Le sujet 12 envisage la programmation comme un art et la pratique comme telle, alors que ce n'est pas le cas pour le sujet 04. Cette contradiction en apparence peut s'expliquer par les propos du sujet 13 lorsque ce dernier affirme que, historiquement, la programmation informatique était d'abord un loisir et une passion pour certains, alors que d'autres y ont vu un potentiel commercial et lucratif. Ces deux façons d'envisager la programmation informatique sont autant de manifestations de rapports au savoir qui peuvent être associées à ce que nous avons dit au sujet de la programmation informatique comme outil ou comme finalité. Les sujets qui l'envisageaient comme finalité ont exprimé un rapprochement avec l'art et l'ont présentée comme un moyen d'expression de soi, alors que ceux qui l'envisageaient comme outil n'ont pas proposé ce rapprochement ou l'ont explicitement nié. Nous constatons chez plusieurs sujets que ces deux usages de programmation peuvent s'observer simultanément

[9] *STEAM* est un acronyme issu de l'anglais qui signifie *Science*, *Technology*, *Engineering*, *Arts* et *Mathematics.*

chez un même individu, celui de l'art et de l'expression associée à la programmation informatique comme finalité, et celui de la résolution d'un problème pour autrui associée à la programmation informatique comme outil. Ces deux façons d'envisager la programmation informatique peuvent également être liées à la concordance des motivations personnelles et des motivations organisationnelles, laquelle se présente comme une condition à l'engagement chez nos sujets.

5.4.2 La programmation informatique comme activité de médiation ontologique

Nos analyses des entretiens nous amènent à considérer la programmation informatique comme une activité dans laquelle des contraintes interviennent. Les sujets ont souligné que ces contraintes peuvent représenter des limites à la solution créée; ils ont aussi illustré qu'elles peuvent représenter des règles en vue de modéliser un problème ou une solution. Ces contraintes apparaissent à l'individu lorsqu'il adopte différentes postures épistémologiques pour regarder une même situation. Quatre postures épistémologiques émergent des analyses, chacune faisant intervenir un ensemble de contraintes : (1) la posture induite par la situation, (2) la posture induite par l'individu, (3) la posture induite par le langage de programmation utilisé et (4) la posture induite par la communauté de pratique. Selon le contexte, ces postures peuvent ou non se superposer les unes aux autres. Ainsi, il peut arriver que dans certains contextes, une personne ait de la difficulté à adopter certaines de ces postures en raison de caractéristiques de la situation auxquelles elle n'est pas habituée.

La posture épistémologique induite par la situation est celle qui permet d'analyser un problème ou un besoin en vue de créer une solution par des activités d'abstraction et de modélisation. Dans le cas de la programmation informatique réalisée pour autrui, la posture induite par la situation est celle qui permet d'aborder le domaine d'affaires ou le besoin exprimé. Cela signifie que cette posture ne fait pas systématiquement intervenir des contraintes d'ordre technologique. Par exemple, le sujet 03 nous a donné l'exemple d'un système de facturation d'un restaurant : les besoins sont de l'ordre de l'efficacité, de l'administration ou de la restauration plutôt que de l'ordre de la technologie. Il peut ensuite adopter la posture induite par la situation en vue de faire émerger les contraintes qu'elle impose à la création d'une solution. Certains sujets nous amènent à considérer que, prise isolément, cette posture ne conduit pas systématiquement à la création d'un programme informatique. Par exemple, le sujet 04 a réitéré à plusieurs reprises que la programmation informatique n'était pas toujours la bonne solution.

La posture épistémologique induite par l'individu est celle qui fait intervenir les motivations personnelles d'un individu, ses représentations ainsi que ses attentes. Nous inférons cette posture à partir des motivations relevées dans les analyses, par exemple celle qui concerne l'intérêt personnel à comprendre comment l'informatique fonctionne. La posture de l'individu peut être comparée aux attentes perçues par l'élève dans le modèle d'interprétation des activités cognitives de l'élève de DeBlois (2000). Par exemple, certains sujets ont

souligné avoir pensé comprendre le besoin d'un client dans un premier temps, mais que dans un deuxième temps ils réalisaient avoir mal compris. Cela peut s'expliquer par une projection des attentes de l'individu sur la situation-problème, de la même façon qu'un élève peut projeter ses attentes sur un problème mathématique. Ces attentes agissent ensuite comme des contraintes supplémentaires imposées par l'individu lui-même, parfois inconsciemment. La posture de l'individu apparaît prépondérante par rapport aux autres. Par exemple, le programmeur tel que décrit par le sujet 07 semble incapable de compromis sur sa propre posture épistémologique, peut-être en raison d'une centration excessive sur lui-même. C'est le propre de l'épistémologie génétique (Piaget, 1970) que de questionner cette capacité du sujet à se détacher de son environnement pour adjoindre à son existence des perspectives qu'il ne contrôle pas.

La posture épistémologique induite par le langage de programmation utilisé est celle qui permet de voir les contraintes imposées par des langages ou par un paradigme de programmation informatique. Certains sujets ont affirmé que les différences entre les langages vont au-delà des changements syntaxiques, rendant certaines solutions possibles dans un langage et impossibles dans un autre. Par exemple, les propos du sujet 11 amènent à considérer que le découpage d'un programme informatique varie en fonction du paradigme de programmation. Cela rejoint les propos d'autres sujets, par exemple ceux du sujet 09 lorsque ce dernier évoque qu'une personne qui programme doit réaliser des abstractions et ce, peu importe le langage de programmation. Il souligne toutefois que la forme de ces abstractions varie d'un langage à l'autre.

La posture épistémologique induite par la communauté de pratique est celle qui permet de voir les contraintes imposées par la tradition, les règles de l'art ou le jugement de ses pairs. Nous inférons cette posture à partir des propos de certains sujets concernant l'existence de pratiques meilleures que d'autres ou bien l'existence de principes de base comme l'évitement des répétitions. Pour certains sujets, la communauté de pratique consiste en des lieux d'échange virtuels entre personnes faisant de la programmation informatique, comme des forums de discussion sur le Web. Plusieurs sujets ont discuté des communautés construites autour des logiciels libres, certains affirmant que ces communautés contribuent à normaliser les pratiques. Pour d'autres, la communauté de pratique s'observe aussi dans des lieux physiques, par exemple dans le cadre du mouvement *Maker* où des espaces créatifs permettent à des gens de se rassembler en vue de créer des artéfacts physiques et numériques. C'est le cas des sujets 05 et 17.

Afin de décrire la programmation informatique en rassemblant les idées exprimées par nos sujets, nous proposons qu'elle soit considérée comme une activité créative dans laquelle l'individu doit tenir compte d'une variété de contraintes. Ces contraintes émergent en adoptant les quatre postures épistémologiques que nous avons décrites précédemment. Cette activité exige de la créativité dès lors que les contraintes imposées par ces différentes postures épistémologiques peuvent entrer en contradiction les unes avec les autres. Par

exemple, certains sujets ont évoqué qu'il était parfois impossible de respecter les pratiques valorisées par la communauté de pratique en raison de courts délais à respecter pour créer un programme informatique. Le délai imparti est une contrainte émergeant de la posture épistémologique induite par la situation, alors que le respect des pratiques éprouvées est une contrainte émergeant de la posture induite par la communauté de pratique. Le respect intégral des deux contraintes étant impossible, le programmeur ou la programmeuse agit comme médiateur afin de trouver un compromis dans le respect de ces multiples contraintes. L'activité de programmation informatique peut donc être considérée comme une activité de médiation. Cette idée est appuyée par le fait que plusieurs sujets ont affirmé avoir eu à tenir plusieurs rôles simultanément dans leur travail. Par exemple, certains d'entre eux affirment avoir agi à titre de gestionnaire de projet, d'analyste et de programmeur ou programmeuse. Ces multiples rôles ont impliqué d'adopter différentes postures épistémologiques afin de regarder une même situation.

Les propos de certains sujets nous amènent à considérer que ces quatre postures épistémologiques peuvent être adoptées par une personne pour agir dans des ontologies variées. Par exemple, dans le cadre de la programmation informatique pour un client, la posture induite par la situation met en lumière l'ontologie du domaine d'affaires. La posture de l'individu s'inscrit dans une ontologie, une réalité propre à chaque individu. En ce sens, non seulement les contraintes qui en émergent varient d'un individu à l'autre, mais la façon dont sont perçues et considérées ces contraintes varient aussi. La posture induite par le langage de programmation utilisé fait émerger des contraintes qui sont considérées comme immuables par nos sujets, à moins qu'il soit question de l'élaboration d'un langage de programmation. Finalement, la posture induite par la communauté de pratique fait intervenir les expériences de partage vécues par l'individu ainsi que la valorisation qu'il donne à ses pairs et à leurs pratiques. Nous dirons donc cette médiation *ontologique*, car elle fait intervenir des constructions de différents domaines de connaissance ou d'affaires chacun pouvant traduire différentes visions du monde et pouvant être abordé avec une posture épistémologique particulière.

La Figure 5 illustre notre définition de la programmation informatique comme un processus de médiation ontologique. La programmation consiste alors à créer une solution en tenant compte d'une variété de contraintes, lesquelles émergent lorsque l'individu adopte chacune des quatre postures épistémologiques.

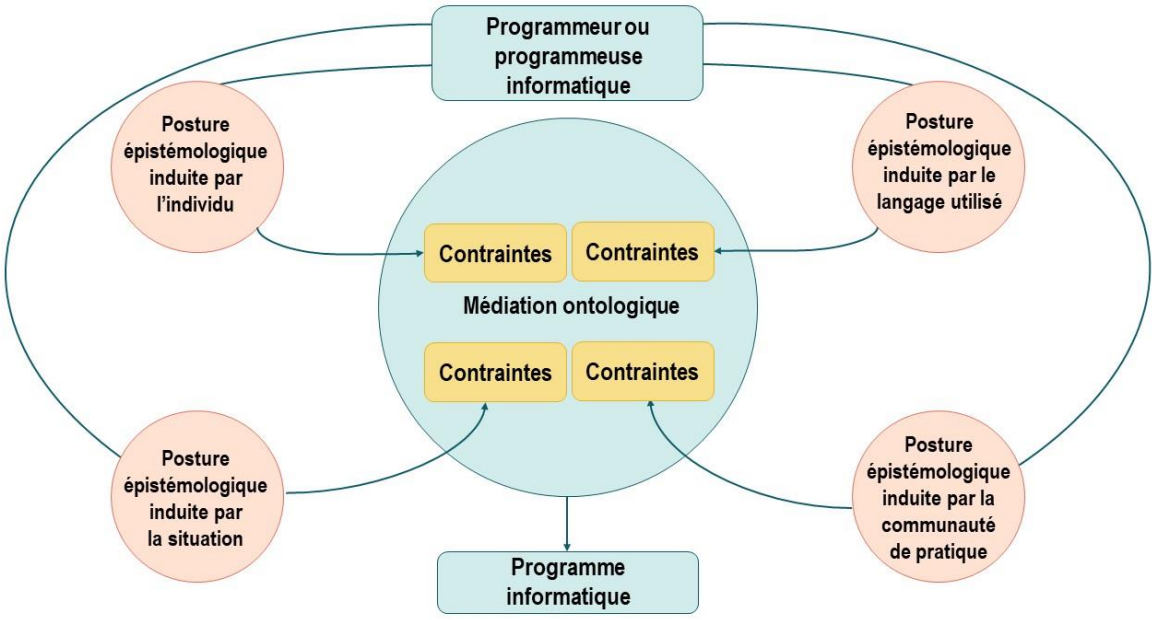

Figure 5. La programmation informatique comme activité de médiation ontologique

5.4.3 Points de convergence ou d'opposition avec certaines études précitées

Nos observations nous ont conduit à considérer, comme Wing (2006, 2008), que l'abstraction est un processus prépondérant dans l'activité de programmation informatique et un point d'entrée pertinent pour l'étudier. Toutefois, contrairement à Wing (2006, 2008), nous considérons que l'abstraction est difficilement observable et que ses manifestations ne sont pas objectives. Les entretiens nous amènent à constater que les couches d'abstraction servant à décrire un programme informatique ou le processus qui a permis de le construire peuvent être différentes selon les points de vue. En ce sens, nous rejoignons DeBlois (2000) qui considère que la compréhension du processus d'abstraction ne peut pas être réalisée seulement par l'étude de la production d'un élève. En plus de la production (par exemple, un programme informatique), cette compréhension doit se développer par une démarche d'investigation et d'explicitation du processus créatif mis en œuvre.

Comme Turkle et Papert (1990), nous avons constaté que la programmation informatique amène les individus à travailler dans des contextes où le pluralisme épistémologique est valorisé. Un même problème peut être regardé depuis différentes postures épistémologiques. En complément, nous avons proposé que ce pluralisme épistémologique peut s'étudier davantage en envisageant la programmation informatique comme une activité de médiation ontologique. Nous avons proposé un modèle où quatre perspectives permettent de faire émerger des contraintes autrement imperceptibles.

Voogt et al. (2015) avaient souligné que la pensée informatique est le plus souvent observée dans des contextes de programmation informatique et que cela ne suffit pas à appuyer le caractère transversal d'une telle compétence. En parallèle avec le modèle d'interprétation des activités cognitives de l'élève de DeBlois (2000), nos observations nous ont conduit à considérer que la transversalité de la pensée informatique peut être observée dans un contexte de programmation informatique, mais seulement si la personne est amenée à verbaliser et contextualiser les choix qu'elle a réalisés dans la création d'un programme informatique. La personne doit être capable de discuter depuis différentes perspectives (celles de la situation-problème, de sa personne elle-même, de la communauté de pratique et du langage utilisé).

Plusieurs références du cadre théorique nous amenaient à considérer que la programmation doit être pratiquée avec une grande marge créative pour favoriser l'engagement (p. ex., Romero, Davidson et al., 2016; Romero, Laferriere et al., 2016). Tous nos entretiens ont convergé vers cette interprétation. Nous ajoutons que la marge créative offerte, tributaire du contexte plutôt que de la personne elle-même, n'est toutefois pas suffisante pour garantir l'engagement. La personne doit aussi percevoir cette marge créative et doit s'envisager personnellement comme créatrice.

5.4.4 Réponse à la question de recherche

Nous proposons ici une réponse systématique à la question de recherche qui guide ce projet de recherche : Quelles sont les conditions supportant l'engagement dans l'activité de programmation informatique ?

L'interprétation des résultats nous amène à proposer trois conditions qui peuvent supporter l'engagement dans l'activité de programmation informatique. Premièrement, pour s'engager dans une activité de programmation informatique, un individu doit développer sa capacité à agir en contexte de médiation ontologique en adoptant consciemment différentes postures épistémologiques pour regarder une même situation. Ces différentes postures amènent à modéliser la situation à partir de l'abstraction de certaines de ses composantes comme les boucles d'itération. Ces composantes sont ensuite organisées pour créer une solution qui répond à des contraintes qui émergent par l'adoption de différentes postures épistémologiques. Ces postures permettent différents regards en raison de la grande diversité des contextes dans lesquels elles peuvent être adoptées. En ce sens, nous considérons que le défaut de varier les postures épistémologiques peut nuire à l'engagement comportemental et à l'engagement cognitif.

Deuxièmement, pour s'engager dans une activité de programmation informatique, un individu doit disposer d'un espace de création suffisamment grand pour qu'il puisse exprimer sa créativité. Nous avons discuté de l'équilibre à maintenir entre les contraintes émergeant des différentes postures épistémologiques dont celle induite par la situation. En effet, dans certains cas, les contraintes peuvent être données et inflexibles, rendant difficile la

médiation ontologique. Cela suggère que le programmeur ou la programmeuse doit aussi être impliqué non seulement dans la construction de la solution, mais aussi dans la construction de l'interprétation du problème. Nous proposons que le défaut d'un espace de création laissant place à sa créativité pourrait réduire l'engagement cognitif et l'engagement comportemental des apprenants et apprenantes dans le cadre d'activités d'apprentissage de la programmation à l'école.

Finalement, pour qu'une personne s'engage dans une activité de programmation informatique, il doit y avoir convergence ou cohérence entre les motivations à agir de la personne qui programme et celles de la personne ou de l'organisation pour qui le programme informatique est réalisé, à défaut de quoi l'engagement affectif être affaibli.

Ces résultats éclairent le mouvement d'intégration de la programmation informatique à l'école primaire et secondaire et outillent les acteurs des systèmes éducatifs en offrant une meilleure compréhension de ce qu'est la programmation informatique du point de vue de la personne, par exemple l'élève, qui la pratique. Notre recherche nous conduit à affirmer qu'un enseignant ou une enseignante qui souhaite amener ses élèves à réaliser des activités de programmation informatique gagnerait à développer une compréhension de l'activité d'un point de vue humain. Apprendre à programmer n'est peut-être pas suffisant pour développer des compétences professionnelles de conception pédagogique, de pilotage d'activités d'apprentissage et d'accompagnement des élèves en lien avec la programmation informatique. Ce sont ces compétences professionnelles qui permettent aux enseignants et enseignantes d'intervenir, par exemple auprès d'élèves en difficulté d'apprentissage. Les manifestations de l'engagement dans la programmation informatique que nous avons observées nous amènent à considérer que ces compétences professionnelles ne peuvent pas être mobilisées sans que les enseignants et enseignantes n'en comprennent les implications sur l'apprentissage.

6 Conclusion

6.1 Rappel de la problématique et de la méthode

La problématique qui a motivé cette étude partait du mouvement d'intégration de la programmation informatique à l'école pour toutes et tous, lequel trouve a fait émerger des questionnements autant au niveau de la recherche que de la pratique enseignante. Au-delà de la présente étude, l'apprentissage pour tous et toutes de la programmation s'inscrit dans une dynamique beaucoup plus large de transformation de la société par l'omniprésence du numérique dans toutes les sphères de la vie. L'arrivée de la programmation informatique à l'école, comme discipline ou comme moyen pédagogique, se fait en continuité avec les transformations de l'école qui ont lieu depuis le XIXe siècle visant à concevoir l'apprentissage du point de vue de l'individu qui le réalise et non du point de vue de l'enseignant ou de l'enseignante qui le planifie. De même, l'intégration de la programmation à l'école n'est pas neutre, elle est influencée par une variété de pouvoirs orientés par autant de valeurs.

Cette intégration de la programmation informatique aux activités des milieux scolaires ne peut faire l'économie d'une compréhension riche de ce qu'elle est par rapport à l'être humain qui la pratique. En guise de contribution à ce projet, nous avons proposé une étude exploratoire portant sur les conditions qui supportent l'engagement en programmation informatique. L'étude a consisté à analyser des entretiens auprès de programmeurs et programmeuses d'expérience. Nous avons assumé qu'ils ont déjà été engagés d'une façon ou d'une autre aux niveaux comportemental, cognitif et affectif. Nous avons volontairement orienté nos travaux vers la compréhension des dynamiques de rapport au savoir qui prennent place dans la programmation informatique, notamment afin d'éclairer l'engagement comportemental. Nous nous sommes affairés à comprendre l'engagement cognitif en décrivant les situations dans lesquelles nos sujets ont fait de la programmation informatique.

6.2 Rappel des résultats

Les analyses des entretiens nous ont amenés à proposer des modèles explicatifs que nous avons situés par rapport à différentes théories de la connaissance, de la pédagogie ou de l'informatique. Parmi ces modèles, nous avons proposé qu'un individu qui fait de la programmation informatique agit comme médiateur ou médiatrice ontologique. Le travail de programmation auquel il participe l'incite sinon l'oblige à adopter différentes postures épistémologiques pour regarder un même problème et à identifier des contraintes à considérer dans la création d'une solution originale. Ces postures s'inscrivent dans des ontologies différentes, ce pourquoi nous considérons que l'individu doit parvenir ultimement à se distancer de chacune d'elles afin de construire une

interprétation effective de la situation. L'engagement comportemental observé chez nos sujets nous a conduit à identifier des paramètres à considérer afin de décrire la pratique de la programmation informatique : le type d'apprentissage (autostructuré ou hétérostructuré), le destinataire de la programmation (soi-même ou autrui) et le rapport au savoir (création ou consommation). La variété des propos récoltés, en lien avec l'engagement en programmation informatique, nous amène à considérer que cette intégration doit s'inscrire dans une vision holistique des transformations de l'école. En ce sens, elle devrait tenir compte des impératifs des économies contemporaines (besoins de main d'œuvre, transformation de l'emploi, citoyenneté numérique), mais aussi et surtout des dimensions comportementale, cognitive et affective inhérentes à la programmation informatique.

6.3 Limites de la présente étude

Les sujets ayant participé à la présente étude ont tous et toutes au moins cinq ans d'expérience en programmation informatique. Nos sujets ont été recrutés via une liste de diffusion regroupant les membres de la communauté universitaire de l'Université Laval, à Québec. La quasi-totalité de nos sujets sont scolarisés au niveau universitaire, or la programmation informatique peut être pratiquée sans égard au niveau scolaire. À cet effet, les analyses ne suffisent pas à généraliser les propositions à l'ensemble des individus pratiquant la programmation informatique. Elles ouvrent des pistes pour la compréhension de l'engagement dans l'activité de programmation informatique, mais ne permettent pas de comprendre en profondeur cet engagement lorsqu'il est question d'élèves placés en situation d'apprentissage au primaire ou au secondaire.

Les entretiens ont été réalisés par une seule et même personne, de même pour les retranscriptions et la codification. Cette personne, l'auteur de ce mémoire, a une expérience professionnelle et amateure de la programmation informatique d'environ 15 ans. Ainsi, malgré un effort de rigueur, de distanciation et de neutralité (notamment au moment des entretiens), elle aborde la question de recherche avec la subjectivité inhérente à toute activité interprétative. Cela représente une limite méthodologique étant donné que l'interprétation des résultats n'a pas fait l'objet d'une validation systématique par plusieurs individus.

La présente étude assume que toutes les formes de programmation informatique ont en commun des formes cohésives entre elles. Or nos sujets nous ont amenés à remettre en question la cohésion entre ces formes. Nous avons orienté notre discussion notamment autour d'une description d'un rapport au savoir supportant l'engagement en programmation sans nous intéresser aux différences qui peuvent intervenir entre différentes formes de programmation (par exemple, le traditionnel continuum entre la programmation de haut niveau et la programmation de bas niveau). Or, l'activité de programmation informatique est peut-être trop générale pour à elle seule capter les rapports au savoir qu'elle implique. Les sciences informatiques ont découpé cette activité de différentes façons qu'il serait intéressant d'explorer en vue d'enrichir l'interprétation des rapports au savoir.

6.4 Perspectives futures

Nous avons évoqué à quelques reprises des études s'intéressant aux difficultés vécues par les programmeurs ou programmeuses novices. À cet égard, notre étude sur les conditions supportant l'engagement en programmation nous conduit à nous questionner sur son corollaire, le non engagement, autrement dit les conditions qui peuvent nuire à l'engagement en programmation informatique. L'étude des difficultés en programmation informatique nous apparaît comme un terreau fertile pour enrichir notre compréhension de la programmation en tant qu'activité humaine surtout dans le contexte d'un déploiement en milieu scolaire. Considérant le mouvement grandissant d'apprentissage de la programmation pour toutes et tous dès l'enfance, il est à prévoir que des difficultés vont émerger chez les élèves au cours des prochaines années et ce, à tous les niveaux scolaires. La préoccupation des systèmes éducatifs pour la réussite du plus grand nombre impliquera forcément de développer une compréhension des origines de ces difficultés et des conditions dans lesquelles elles surviennent. Cette compréhension pourrait s'avérer importante pour accompagner les professionnels de l'éducation qui doivent intervenir auprès des enfants; elle permettra de bâtir des interventions didactiques adaptées aux particularités de l'activité de programmation informatique, s'il y a lieu.

Rappelons que la présente étude se veut exploratoire et interprétative, elle ne visait pas à la généralisation. Toutefois, les modèles théoriques dégagés lors de la discussion des résultats pourront éventuellement appuyer au niveau théorique la construction de devis quantitatifs visant à tester si les conditions supportant l'engagement peuvent être généralisées à des groupes.

7 Bibliographie

Albero, B. (2002). L'autoformation en contexte institutionnel : entre la contingence et l'utopie. Dans *Université ouverte, formation virtuelle et apprentissage* (pp. 459–483). Paris, France: L'Harmattan.

Azevedo, R. (2005). Computers environments as metacognitive tools for enhancing learning. *Educational Psychologist*, *40*(4), 193–197. https://doi.org/10.1207/s15326985ep4004

Balanskat, A. et Engelhardt, K. (2015). *Computing our future*. Bruxelles, Belgique: European Schoolnet.

Barba, L. A. (2016). Computational Thinking: I do not think it means what you think it means. Repéré le 23 mai 2018 à http://lorenabarba.com/blog/computational-thinking-i-do-not-think-it-means-what-you-think-it-means/

Bélanger, J.-P., DeBlois, L. et Freiman, V. (2014). Interpréter la créativité du raisonnement dans les productions d'élèves en mathématiques d'une communauté d'apprentissages multidisciplinaires interactifs. *Éducation et Francophonie*, *XLII*(2), 44–63.

Bosch, N., D'Mello, S. et Mills, C. (2013). What emotions do novices experience during their first computer programming learning session? Dans *Proceedings of the 16th International Conference Artificial Intelligence in Education* (pp. 11–20). https://doi.org/10.1007/978-3-642-39112-5-2

Brenner, M. E. (2006). Interviewing in Educational Research. Dans J. L. Green, G. Camilli et P. B. Elmore (dir.), *Handbook of Complementary Methods in Education Research* (pp. 357–370). Washington, États-Unis: American Educational Research Association.

Brookfield Institute. (2018). Digital Literacy + Coding Pilot. Repéré le 3 mai 2018 à http://brookfieldinstitute.ca/pilots-prototypes/digital-literacy-coding-pilot/

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, *1*(2), 67–69.

Cloutier, P. (2018). Codage informatique à l'école: éviter les erreurs du passé. Repéré le 15 mai 2018 à https://www.lesoleil.com/actualite/education/codage-informatique-a-lecole-eviter-les-erreurs-du-passe-0c9b71ce642840bd18d1e6648973f97e

DeBlois, L. (1992). Trois approches visant la compréhension abstraite logico-mathématique de la numération positionnelle chez des enfants en troubles d'apprentissage. *Instantanés Mathématiques*, *23*(3), 6–14.

DeBlois, L. (1994). Le rôle des représentations mentales dans le développement de l'abstraction réfléchissante. Dans *Commission internationale pour l'étude et l'amélioration de l'enseignement des mathématiques*. Toulouse, France: Université Paul Sabatier.

DeBlois, L. (1996). Une analyse conceptuelle de la numération de position au primaire. *Recherches en Didactique des Mathématiques*, *16*(1), 71–128.

DeBlois, L. (1997). Quand additionner ou soustraire implique de comparer. *Éducation et Francophonie*, *XXV*(1), 102–120.

DeBlois, L. (2000). Un modèle d'interprétation des activités cognitives pour des élèves qui éprouvent des difficultés d'apprentissage en mathématiques. Dans *Actes du colloque Constructivismes: usages et perspectives en éducation* (pp. 565–573). Genève, Suisse.

DeBlois, L. (2003). Interpréter explicitement les productions des élèves : une piste... *Éducation et Francophonie*, *XXXI*(2), 176–199. Repéré à http://www.acelf.ca/c/revue/pdf/XXXI_2_176.pdf

DeBlois, L. (2015). Interactions de la classe : tensions entre compréhension et difficultés à apprendre les mathématiques. Dans *Actes de la 37e rencontre annuelle du groupe canadien en didactique des mathématiques* (pp. 171–186).

Descartes, R. (2000). *Discours de la méthode*. Paris, France: Flammarion.

Fortin, M.-F. et Gagnon, J. (2016). *Fondements et étapes du processus de recherche*. Montréal, Canada: Chenelière Éducation.

Fouh, E., Karavirta, V., Breakiron, D. A., Hamouda, S., Hall, S., Naps, T. L. et Shaffer, C. A. (2014). Design and architecture of an interactive eTextbook - The OpenDSA system. *Science of Computer Programming*, *88*, 22–40. https://doi.org/10.1016/j.scico.2013.11.040

Fourez, G. (2004). *Apprivoiser l'épistémologie*. Paris, France: Éditions de Boeck Université.

Fredricks, J., Blumenfeld, P. et Paris, A. (2004). School engagement : potential of the concept, State of the Evidence. *Review of Educational Research*, *74*(1), 59–109.

Goupille, P.-A. (2015). *Technologie des ordinateurs et des réseaux* (9e éd.). Paris, France: Dunod.

Gouvernement du Canada. (2017). Le gouvernement du Canada lance un programme de codage de 50 millions de dollars à l'intention des jeunes Canadiens. Repéré le 3 mai 2018 à https://www.canada.ca/fr/innovation-sciences-developpement-economique/nouvelles/2017/06/le_gouvernement_ducanadalanceunprogrammedecodagede50millionsdedo.html

Green, T. R. G. (1990). The nature of programming. Dans J.-M. Hoc, T. R. G. Green, R. Samurçay et D. J. Gilmore (dir.), *Psychology of programming* (pp. 21–44). Londres, Royaume-Uni: Academic Press Limited.

Guarino, N. (1998). Formal ontology and information systems. Dans N. Guarino (dir.), *Proceedings of FOIS'98* (pp. 3–15). Trento: IOS Press. https://doi.org/10.1.1.29.1776

Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications of the ACM*, *12*(10), 576–580. https://doi.org/10.1145/363235.363259

Hoc, J.-M., Green, T. R. G., Samurçay, R. et Gilmore, D. J. (dir.). (1990). *Psychology of programming*. London: Academic Press Limited.

Hopcroft, J. E. (1987). Computer Science: The Emergence of a Discipline. *Communications of the ACM*, *30*(3), 198–202. https://doi.org/10.1145/214748.214750

Jonassen, D. H. (2006). *Modeling with Technology* (3e éd.). Upper Saddle River, États-Unis: Pearson.

Kant, E. (1944). *Critique de la raison pure*. Paris, France: Presses Universitaires de France.

Kasurinen, J., Purmonen, M. et Nikula, U. (2008). A study of visualization in introductory programming. Dans *Proceedings of the Annual Meeting of the Psychology of Programming Interest Group* (pp. 181–194).

Kelleher, C. (2009). Barriers to programming engagement. *Advances in Gender and Education*, *1*, 5–10.

Kölling, M. (1999). The problem of teaching object-oriented programming. Part 1: Languages. *Journal of Object Oriented Programming*, *11*(8), 8–15. Repéré à http://wiki.bluej.org/papers/1999-09-JOOP2-environments.pdf%5Cnhttp://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.8492

Langevin, P. (1934). *La notion de corpuscules et d'atomes*. Paris, France: Hermann.

Lavoie, M.-A. (2018). Des robots comme outil pédagogique dans les écoles. Repéré le 15 mai 2018 à https://ici.radio-canada.ca/nouvelle/1079506/robots-education-projet-pilote-programmation-informatique-ecoles

Lepage, A. (2018). GraphQDA - GraphQDA is a qualitative analysis software. Repéré à http://graphqda.alepage.net/

Lister, R. (2011). Concrete and other neo-piagetian forms of reasoning in the novice programmer. *Conferences in Research and Practice in Information Technology Series*, *114*, 9–18.

Madni, A. M. (2011). Model Based Systems Engineering with Department of Defense Architectural Framework. *Elegant Systems Design: Creative Fusion of Simplicity and Power*, 305–326. https://doi.org/10.1002/sys.20180

Magnusson, E. et Marecek, J. (2015). *Doing Interview-Based Qualitative Research*. Cambridge, Royaume-Uni: Cambridge University Press.

Mendelsohn, P., Green, T. R. G. et Brna, P. (1990). Programming Languages in Education: The Search for an Easy Start. Dans J.-M. Hoc, T. R. G. Green, R. Samurçay et D. J. Gilmore (dir.), *Psychology of Programming* (pp. 175–200). Londres, Royaume-Uni: Academic Press Limited.

Ministère de l'Éducation et de l'Enseignement supérieur. (2018). *Plan d'action numérique en éducation et en enseignement supérieur*. Repéré à http://www.education.gouv.qc.ca/fileadmin/site_web/documents/ministere/PAN_FINAL.pdf

Naps, T. L., Rodger, S., Velázquez-Iturbide, J. Á., Rößling, G., Almstrum, V., Dann, W., … McNally, M. (2003). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, *35*(2), 131–152. https://doi.org/10.1145/782941.782998

Not, L. (1979). *Les pédagogies de la connaissance*. Toulouse: Privat.

Paloque-Bergès, C. (2009). *Poétique des codes*. Paris, France: Édition des archives contemporaines.

Papavlasopoulou, S. (2016). On the relationship between Engagement, Learning, and Creativity in Computer

Science: Toward better understanding and improved methods. https://doi.org/10.1109/ICALT.2016.53
Papert, S. (1980). *Mindstorms*. New York, États-Unis: BasicBooks.
Papert, S. (1992). *The children's machine*. New York, États-Unis: BasicBooks.
Parent, S. (2014). De la motivation à l'engagement. *Pédagogie Collégiale*, *27*(3), 13–16.
Pea, R. D., Soloway, E. et Spohrer, J. C. (1987). The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics*, *9*(1), 5–30.
Piaget, J. (1970). *L'épistémologie génétique*. Paris, France: Quadrige.
Piaget, J. (1974). *La prise de conscience*. Paris, France: Presses Universitaires de France.
Piaget, J. (1977). *Recherches sur l'abstraction réfléchissante*. Paris, France: Presses Universitaires de France.
Pickering, M. (2011). Le positivisme philosophique : Auguste Comte. *Revue Interdisciplinaire d'Études Juridiques*, *67*(2), 49–67. Repéré à https://www.cairn.info/revue-interdisciplinaire-d-etudes-juridiques-2011-2-page-49.htm
Prévost, H. (1994). *L'individualisation de la formation*. Lyon, France: Chronique Sociale.
Proulx, S. (2018). *Un Québec libre est un Québec qui sait lire et écrire*. Québec, Canada: Septentrion.
Resnick, M. (2007). All I really need to know (about creative thinking) I learned (by studying how children learn) in kindergarten. *Proceedings of the 6th ACM SIGCHI Conference on Creativity & Cognition – C C '07*, (June), 1–6. https://doi.org/10.1145/1254960.1254961
Resnick, M. (2014). Give P's a chance: Projects, Peers, Passion, Play. *Constructionism and Creativity Conference*.
Rochkind, M. (2004). *Advanced UNIX Programming* (2e éd.). Boston, États-Unis: Addison-Wesley Professional. Repéré à https://www.safaribooksonline.com/library/view/-/9780132466004/?ar
Romero, M. (2016). Apprendre un jour, apprendre toujours. Repéré le 17 juin 2018 à http://www.contact.ulaval.ca/article_blogue/apprendre-jour-apprendre-toujours/
Romero, M. (2018). Coder à l'école, ce n'est pas assez. Repéré le 15 mai 2018 à http://www.contact.ulaval.ca/article_blogue/coder-a-lecole-nest-assez/
Romero, M., Davidson, A.-L., Cucinelli, G., Ouellet, H. et Arthur, K. (2016). Learning to code : from procedural puzzle-based games to creative programming. Dans *Proceedings of the 206 International Conference on University Teaching and Education (CIDUI)*. Barcelone, Espagne.
Romero, M., Laferrière, T. et Power, T. M. (2016). The move is on! From the passive multimedia learner to the engaged co-creator. *ELearn*, *2016*(3). https://doi.org/10.1145/2904374.289338
Romero, M., Lepage, A. et Lille, B. (2017). Computational thinking development through creative programming in higher education. *International Journal of Educational Technology in Higher Education*, *14*(1). https://doi.org/10.1186/s41239-017-0080-z
Romero, M. et Lille, B. (2017). La créativité, au coeur des apprentissages. Dans M. Romero, B. Lille et A. Patiño (dir.), *Usages créatifs du numérique pour l'apprentissage au XXIe siècle* (pp. 29–39). Québec, Canada: Presses de l'Université du Québec.
Rushkoff, D. (2010). *Program or be programmed*. New York, États-Unis: OR Books.
Selby, C. C. et Woollard, J. (2013). Computational Thinking : The Developing Definition. *Proceedings of the 2014 Conference of the Special Interest Group on Computer Science Education (SIGCSE)*, 5–8.
Sheard, J., Carbone, A. et Hurst, A. J. (2010). Student engagement in first year of an ICT degree: Staff and student perceptions. *Computer Science Education*, *20*(1), 1–16. https://doi.org/10.1080/08993400903484396
Strawhacker, A. et Bers, M. U. (2015). "I want my robot to look for food": Comparing kindergartner's programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education*, *25*(3), 293–319. https://doi.org/10.1007/s10798-014-9287-7
Turkle, S. et Papert, S. (1990). Epistemological pluralism : styles and voices within the computer culture. *Signs*, *16*(1), 128–157.
University of Chicago. (2017). *The Chicago manual of style* (17e éd.). Chicago, États-Unis: University of Chicago Press.
Vergnaud, G. (1990). La théorie des champs conceptuels. *Recherche en Didactique des Mathématiques*, *10*(2.3), 133–170.

Voogt, J., Fisser, P., Good, J., Mishra, P. et Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, *20*, 715–728. https://doi.org/10.1007/s10639-015-9412-6

Weinberg, G. M. (1998). *The psychology of computer programming* (Silver anniversary éd.). New York, États-Unis: Dorset House Publishing.

Wenger, É. (1998). *Communities of practice. Learning, Meaning and identity*. Cambridge, Royaume-Uni: Cambridge University Press.

Wiggins, G. A. (2006). Searching for computational creativity. *New Generation Computing*, *24*(3), 209–222. https://doi.org/10.1007/BF03037332

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *IPDPS Miami 2008 - Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Program and CD-ROM*, (July), 3717–3725. https://doi.org/10.1109/IPDPS.2008.4536091

Yuan, X., Vega, P., Qadah, Y., Archer, R., Yu, H. et Xu, J. (2010). Visualization Tools for Teaching Computer Security. *ACM Transactions on Computing Education (TOCE)*, *9*(4), 20. https://doi.org/10.1145.1656255.1656258

Annexes

Annexe 1 : Guide d'entretien

Ce guide d'entretien est rédigé à l'attention de l'expérimentateur[10]. Dans ce projet de recherche, l'expérimentateur sera Alexandre Lepage (étudiant à la maîtrise) pour tous les participant.e.s.

Des exemples sont parfois donnés entre parenthèses pour illustrer le type d'informations attendues. Toutefois, ces exemples ne seront en aucun cas donnés aux participant.e.s afin d'éviter de suggérer des réponses. Pour les thèmes 2 et 3, nous n'utilisons pas d'échelle de type Likert (mais l'objectif est similaire). Nous ne cherchons pas à quantifier le niveau d'accord du participant, mais bien à le comprendre en profondeur.

Avant de démarrer l'enregistrement audio et de débuter l'entretien, le ou la participant.e se fera remettre le formulaire de consentement et devra le lire au complet avant d'y apposer sa signature. Il pourra poser des questions à ce sujet à l'expérimentateur. Si le participant ou la participant.e accepte toutes les modalités présentées dans le formulaire et y appose sa signature, on l'informera du démarrage de l'enregistrement audio puis l'entretien pourra débuter. Si le ou la participant.e manifeste son désir d'interrompre l'entretien à un moment quelconque, l'enregistrement sera arrêté et cela mettra fin à l'entretien. On demandera au participant s'il désire que l'enregistrement fait soit supprimé et qu'il ne soit pas utilisé dans l'analyse. S'il le désire, on supprimera, en sa présence, l'enregistrement audio. À défaut d'un consentement explicite à ce que les données partielles soient utilisées, on assumera que le participant refuse et les données seront supprimées et non utilisées.

Thème 1 : Récit de l'expérience en programmation informatique

Objectif : Permettre au participant ou à la participante de raconter le contexte dans lequel il ou elle a appris à programmer. La question de la motivation à programmer est importante, car elle permettra d'interpréter différemment les propos selon s'il s'agit principalement de motivations intrinsèques ou extrinsèques, par exemple. Cela permettra également d'établir des parallèles avec les types de programmeurs évoqués par Weinberg (1998).

1. **De quelle façon avez-vous appris à programmer?**

Note à l'expérimentateur : si le participant ou la participante évoque un apprentissage de la programmation informatique à l'école, s'assurer d'avoir au moins ces informations : le type de cours suivis (ex. : bases de données, réseau, programmation système), le niveau scolaire auquel se donnaient les cours et une appréciation personnelle de cette expérience (pour savoir si l'expérience était positive ou négative, ou nuancée).

2. **Quelles sont vos motivations à programmer? Ont-elles toujours été les mêmes ou ont-elles évolué dans le temps?**

Note à l'expérimentateur : si le ou la participant.e programme dans différents contextes, s'assurer qu'il développe sur chacun de ces contextes en le relançant au besoin (ex. : projets personnels et professionnels). Cela

[10] Le terme expérimentateur désigne l'intervieweur. Il a été remplacé dans le mémoire, mais figure ici pour faire état du guide d'entretien tel qu'il a été accepté par les Comités d'éthique de la recherche avec des êtres humains de l'Université Laval (CÉRUL).

permettra éventuellement de voir s'il existe des motivations sous-jacentes à la personne plutôt qu'au contexte (si, par exemple, elles reviennent dans différents contextes).

3. **Décrivez la nature de la programmation informatique que vous réalisez.**

Note à l'expérimentateur : s'assurer ici que le ou la participant.e. mentionne au moins les langages avec lesquels il ou elle travaille, le type de programmation (ex. : objet ou procédurale), les habiletés connexes (ex. : gestion de bases de données, administration de réseaux).

Thème 2 : Développement de compétences en programmation

Objectif : Amener le ou la participant.e à s'exprimer sur les stratégies de pensée employées par des programmeurs ou programmeuses. L'ordre et le choix des énoncés sont établis tels que présentés dans ce guide et doivent être respectés au moment de l'expérimentation. Les énoncés s'opposent ou se complètent parfois. Cela peut amener les participant.e.s à modifier leur discours en précisant des idées qu'ils ont précédemment évoquées ou à rectifier des contradictions apparentes qui pourraient survenir dans leur discours. Il n'est pas ici question de piéger les participant.e.s, mais bien de comprendre en profondeur leurs propos.

L'expérimentateur lit l'énoncé sur un ton neutre. Au besoin, il peut le relire afin de permettre au participant.e de s'assurer qu'il ou elle s'est prononcé sur tous les aspects.

Pour chaque énoncé, le participant est appelé à se dire d'accord ou en désaccord, à établir des nuances si nécessaire, à justifier sa position et à enrichir l'énoncé en le développant avec ses propres idées. Le participant n'est pas informé de la source des énoncés et ne doit pas voir l'ensemble des énoncés à l'avance.

1. **« Tout le monde est capable de programmer. »**

Source : construit sur mesure

Justification : l'objectif est d'amener le ou la participant.e à se prononcer spontanément sur cette question, avant d'être exposé à des idées qui exprimeront des positions tranchées à ce sujet. La réponse à cette question sera fort utile à l'analyse de discours, puisqu'elle ne pourra pas avoir été influencée par les énoncés subséquents.

2. **« Penser comme un informaticien, ça veut dire plus que savoir programmer un ordinateur. Ça implique de réfléchir à plusieurs niveaux d'abstraction. »**

Source : traduit de Wing, J. (2006). Computational Thinking. Communications of the ACM, 49(3), 33–35.

Justification : cette idée est abondamment reprise dans la littérature scientifique depuis 2006. Le but est de voir si elle décrit adéquatement la façon de concevoir la programmation pour le ou la participante.

3. **« Ça prend certaines capacités intellectuelles d'analyse pour être un bon programmeur. Il ne faut pas avoir peur des mathématiques et de la logique, et être prêt à faire face à des frustrations. Ce n'est pas tout le monde qui a ces habiletés ou qui a la patience nécessaire pour les développer. Donc la programmation, c'est n'est pas pour tout le monde. »**

Source : traduit d'un commentaire anonyme sur Reddit

Justification : ce commentaire reprend une association qui réduit la programmation informatique au champ des mathématiques. Pour certains, la programmation informatique est multidisciplinaire et a des particularités qui font qu'elle n'est pas que mathématique ; d'autres recherches sont souhaitables pour comprendre et vérifier cette idée (Czerkawski et Lyman (2015).

4. **« Oui, les programmeurs doivent réfléchir de façon abstraite, notamment lorsqu'ils analysent un problème. Mais je ne dirais pas que c'est le coeur de la pensée informatique. Les programmeurs sont des bricoleurs, ils peuvent aussi procéder par essai-erreur si c'est plus rapide, ils utilisent aussi des méthodes très concrètes. Je crois que la programmation, c'est pas juste réfléchir de façon abstraite. »**

Source : construit sur mesure

Justification : ce commentaire est construit principalement à partir d'un ouvrage de Seymour Papert (1993) : The Children's Machine. Il vise à enrichir l'association entre programmation et abstraction en suggérant une nuance. Il est directement relié à la question de recherche : « Quelles peuvent être les balises conceptuelles d'une compétence développée par la pratique programmation informatique ? ». On vérifie donc comment le participant.e adhère à l'idée que le concept d'abstraction n'est pas à lui seul suffisant pour baliser la compétence développée en programmation et, le cas échéant, pourquoi.

Thème 3 : L'apprentissage pour tous de la programmation à l'école

Objectif : Exposer le participant à des idées reçues dans l'actualité des dernières années au sujet de la démocratisation des technologies et en particulier de la programmation informatique. Il est ici attendu que le participant ou la participante puisse se sentir interpelé de différentes façons (ex. : en étant fortement opposé à des vulgarisations trop réductrices de son travail ou au contraire en surenchérissant).

L'expérimentateur lit l'énoncé sur un ton neutre. Au besoin, il peut le relire afin de permettre au participant.e de s'assurer qu'il ou elle s'est prononcé sur tous les aspects.

Pour chaque énoncé, le participant est appelé à se dire d'accord ou en désaccord, à établir des nuances si nécessaire, à justifier sa position et à enrichir l'énoncé en le développant avec ses propres idées. Le participant n'est pas informé de la source des énoncés et ne doit pas voir l'ensemble des énoncés à l'avance.

1. **« Le code, c'est un vecteur de créativité. C'est les bases qui nous permettent de communiquer à l'ère où l'on vit aujourd'hui. C'est important que ces notions touchent nos jeunes dès le plus jeune âge. L'algorithmique, c'est simplement de communiquer des instructions à une machine. À l'école, au même titre qu'on apprend à lire et écrire, tout le monde devrait apprendre à programmer. »**

Source : adapté de http://ici.radio-canada.ca/emissions/la_sphere/2012-2013/chronique.asp?idChronique=416838

2. **« Les besoins les plus fondamentaux des jeunes se situent au niveau de la langue et des mathématiques, et que c'est ce sur quoi l'école primaire devrait se concentrer. Je recommande d'attendre la fin du primaire avant d'introduire l'informatique. »**

Source : adapté de https://ecolebranchee.com/2014/11/17/les-cours-dinformatique-au-debut-du-primaire-pour-ou-contre/

3. **« La technologie, c'est super facile. C'est comme apprendre à utiliser de la pâte à dents. Les compagnies comme Google s'efforcent de rendre leurs produits faciles à utiliser sans qu'on ait à réfléchir. Je ne vois pas pourquoi des enfants ne réussiraient pas à les utiliser lorsqu'ils deviennent plus vieux. L'enseignement est une expérience humaine. Lorsque l'objectif est d'apprendre à lire, écrire, compter, et à réfléchir de façon critique, la technologie est une distraction. L'idée qu'une application sur iPad peut aider à apprendre à lire et écrire est ridicule. »**

Source : adapté et traduit de http://www.nytimes.com/2011/10/23/technology/at-waldorf-school-in-silicon-valley-technology-can-wait.html

Ces questions doivent être posées explicitement au participant si l'expérimentateur juge qu'elles n'ont pas obtenu de réponses directes :

1. **Que pensez-vous de l'apprentissage de la programmation pour tous dès l'école primaire?**

Note à l'expérimentateur : Si le participant est en faveur, ou s'il fait montre de nuances : quelles seraient les façons d'intégrer la programmation informatique à l'école selon vous?

Justification : Cette question pourrait permettre de mieux relier les propos du ou de la participant.e à la littérature scientifique.

2. **Avez-vous appris à programmer à l'école? Si non, auriez-vous aimé cela? Pourquoi?**

Annexe 2 : Courriel de recrutement

Titre du courriel : Recrutement de participant.e.s ayant de l'expérience en programmation informatique

Madame, Monsieur,

Nous sommes à la recherche d'individus de 18 ans et plus ayant au moins cinq ans d'expérience en programmation informatique afin de participer à une étude sur le développement de compétences. Le but de l'étude est de mieux comprendre la nature des compétences en programmation informatique et les mécanismes qui permettent de les développer. L'expérience en programmation informatique peut être amateure ou professionnelle et acquise dans différents contextes. L'étude est ouverte à toutes et à tous, peu importe l'occupation, le domaine d'études ou le type de programmation que vous faites. Vous n'aurez pas à réaliser de tâches de programmation informatique dans le cadre de l'étude.

Votre participation consisterait en un entretien d'une durée approximative de 45 minutes dans lequel vous seriez invité.e à vous exprimer sur trois thèmes : la programmation informatique, le développement de compétences et l'apprentissage de la programmation à l'école. Les entretiens font l'objet d'un enregistrement audio. Tout le matériel de la recherche sera conservé de façon confidentielle et détruit au plus tard deux ans après la fin de la recherche. Les données de recherche agrégées seront conservées pour utilisation ultérieure, mais de façon anonyme. Aucune compensation financière n'est offerte.

Cette étude, intitulée « Étude exploratoire de l'expérience subjective de développement des compétences en programmation informatique »[11], est réalisée dans le cadre du projet de maîtrise de Alexandre Lepage, étudiant à la maîtrise en technologie éducative. Le projet est dirigé par Lucie DeBlois, Ph. D. et codirigé par Margarida Romero, Ph. D.

Si vous êtes intéressé.e et disponible, veuillez contacter Alexandre Lepage par courriel à alexandre.lepage.1@ulaval.ca.

Ce projet a été approuvé par le Comité d'éthique de la recherche de l'Université Laval : No d'approbation 2017-183 / 17-07-2017

[11] Depuis l'envoi de ce courriel, le titre du projet de recherche a changé pour des raisons méthodologiques.

Annexe 3 : Formulaire de données complémentaires

Numéro du participant : _____

Genre : [ ] Masculin [ ] Féminin [ ] Autre

Années d'expérience en programmation : _____

Type de programmation prédominant : [ ] Amateure [ ] Professionnelle

Note : si applicable, possibilité de cocher les deux réponses

Annexe 4 : Glossaire non exhaustif des termes informatiques utilisés

Note : Les définitions sommaires ci-dessous sont données à titre indicatif afin de faciliter la compréhension de la présente étude. Elles n'établissent pas toutes les nuances qui existent.

Arduino : Arduino est un microcontrôleur sur lequel différentes composantes électroniques peuvent être interfacés (p. ex. capteur de masse ou de température). Il est utilisé dans des projets de robotique notamment dans le milieu scolaire.

Assembleur : L'assembleur désigne un ensemble de langages de programmation de très bas niveau destinés à communiquer directement avec un processeur. La programmation en assembleur implique de manipuler directement les plus petites unités de mémoire d'un ordinateur, les bits.

Base de données : Une base de données est un rassemblement de données dont l'organisation a été planifiée. Il existe différents types de base de données, en programmation informatique elles sont souvent utilisées pour assurer la pérennité de données dans un logiciel ou une application.

Boucle : Une boucle est un procédé en programmation informatique qui permet de répéter une séquence d'instructions ou de procédures.

C et C++ : C et C++ sont deux langages de programmation différents, mais apparentés. Ils sont des langages à fort typage largement utilisés encore aujourd'hui, notamment dans la programmation de systèmes d'exploitation.

Condition : Une condition, ou structure conditionnelle, sert à baliser l'exécution – ou non – d'un code informatique. La prise en charge des conditions est généralement considérée comme un fondement de la programmation informatique.

Fonction : Une fonction est un regroupement d'instructions dont l'exécution prend en charge des paramètres définis lors de l'appel de fonction.

HTML : L'acronyme signifie Hypertext markup language. Langage informatique descriptif principalement utilisé pour la création de pages Web. Il n'est généralement pas considéré comme un langage de programmation étant donné qu'il permet de produire un contenu statique.

Intelligence artificielle : Expression désignant la tentative d'amener un ordinateur à prendre des décisions jusqu'ici seulement possibles par l'être humain. Concrètement, l'intelligence artificielle est basée sur des modèles probabilistes alimentés par des données massives. Un des champs d'application populaire est la reconnaissance d'images.

Java : Langage de programmation orienté objet créé par Sun Microsystems en 1996. Il s'agit d'un langage interprété par la machine java, et ainsi il facilite le développement de logiciels multiplateformes. En 2018, il demeure le langage de programmation le plus utilisé.

Javascript : Langage de programmation orienté objet créé par Netscape dans les années 1990 à l'origine pour dynamiser des pages Web. Longtemps utilisé dans la guerre des navigateurs Web, notamment entre Microsoft Internet Explorer et Netscape Navigator, il a souffert d'une standardisation lente – ou du respect de cette standardisation, rendant son utilisation incertaine d'un navigateur à l'autre. Depuis le milieu de la décennie 2000, et notamment avec l'apparition du navigateur Google Chrome, il connaît un essor important notamment via l'apparition de node.js, un interpréteur javascript côté serveur.

Logiciel libre : Un logiciel libre est un logiciel dont le code source est ouvert et modifiable par quiconque y est intéressé. Il peut être redistribué sans contraintes particulières. Différentes licences amènent des nuances sur cette définition, certaines des plus connues étant la licence GNU GPL, la licence MIT et la licence Apache. Un logiciel libre peut être vendu et n'est pas nécessairement gratuit (bien que dans la pratique ce soit souvent le cas).

Logiciel propriétaire : Un logiciel propriétaire est un logiciel dont le code source est protégé par le droit d'auteur. Le logiciel est distribué dans une version compilée, et il est interdit, selon les lois en vigueur, d'étudier son fonctionnement. Son code source est privé.

PHP : L'acronyme signifie Hypertext Preprocessor. Il s'agit d'un langage de programmation principalement utilisé pour la génération de contenus HTML dynamiques, c'est-à-dire pour la confection de pages Web dynamiques. Il est généralement considéré comme un langage de haut niveau à typage faible.

Programmation en binôme ou *pair programming* : Méthode de programmation où deux personnes travaillent simultanément sur un même code informatique, mais chacune avec un rôle différent. Un se charge de l'écriture, l'autre de la validation en temps réel du devis et de l'analyse, du repérage d'erreurs éventuelles.

Programmation orientée objet : Paradigme de programmation qui vise à représenter l'information sous forme d'objets avec des propriétés ainsi que des actions possibles sur ces propriétés. Certains langages de programmation abordent ce paradigme de façon flexible, alors que d'autres l'abordent de façon rigoureuse et contraignent l'utilisateur à tout définir sous forme d'objet. Le paradigme est généralement reconnu comme ajoutant de la rigueur et de la lisibilité à un programme; parfois il alourdit aussi inutilement le développement de programmes simples.

Raspberry Pi : Nano-ordinateur pouvant tenir dans la main mais permettant l'exécution d'un système d'exploitation entier. Il offre sensiblement les mêmes fonctions qu'un ordinateur portable avec des ports USB, un port HDMI, une antenne Wifi, etc. Il est utilisé dans des projets de robotique en raison de la possibilité d'y ajouter des composantes maisons.

Système embarqué : Programme informatique ou ensemble de programmes soumis à des contraintes fortes d'indépendance et de fiabilité (par exemple, le logiciel de bord d'un avion).

Variable : Les variables sont considérées comme un élément central de la programmation informatique, elles servent à stocker et manipuler de l'information. Les langages de programmation gèrent différemment les variables, certains étant très rigoureux sur la façon de les manipuler, d'autres rendant des manipulations complexes invisibles au programmeur (un exemple courant est la conversion automatique d'un nombre en chaîne de caractères).