Apprendre à programmer: comment les enseignants justifient-ils le choix d'un outil didactique?

Etienne Vandeput, Julie Henry

Étienne Vandeput[a] & Julie Henry[b]

a. Institut Universitaire de Formation des Enseignants (IUFE), Genève, Suisse
etienne.vandeput@hotmail.com
b. Centre de recherche PReCISE – NAmur Digital Institute (NADI) – Université de Namur, Belgique
julie.henry@unamur.be

Apprendre à programmer
Comment les enseignants justifient-ils le choix d'un outil didactique ?

Résumé
De nombreux outils sont utilisés dans un contexte d'enseignement de la programmation. Mais comment ces derniers sont-ils choisis ? Quels sont les critères sur lesquels les enseignants appuient leur choix ? À travers une série d'entretiens semi-dirigés menés auprès d'enseignants d'horizons différents (secondaire, supérieur universitaire et non universitaire), des éléments de réponse sont apportés. Ainsi, un outil n'est pas choisi parce qu'il facilite l'apprentissage, mais plutôt parce qu'il est actuel, parce qu'un supérieur le souhaite ou parce qu'il est utilisé dans le monde du travail. De même, aucune réflexion d'ordre didactique n'est envisagée pour aider l'étudiant à surmonter ses difficultés. Le constat pessimiste dressé par cet article invite à envisager des recommandations à prendre en compte dans la perspective d'une introduction à la pensée informatique dans l'enseignement obligatoire.

Mots clés : enseignement de la programmation, apprentissage de la programmation, choix d'un outil, didactique

1 Introduction

À l'école obligatoire, on a très tôt voulu enseigner la programmation informatique. Mais le manque d'expérience, de repères, les carences en matière de formation des enseignants et la complexité de l'exercice ont eu raison de cette première orientation. Le vent de la pédagogie s'est assez

rapidement mis à souffler dans la direction des usages, voire de l'aide à l'apprentissage. Ce vent semble avoir tourné, du moins momentanément. Il revient dans la direction prise au tout début de l'apparition de l'informatique à l'école, mais pour d'autres raisons. L'une des principales est sans doute que, comme le prévoyaient les optimistes, tout le monde finirait un jour par utiliser les technologies numériques, mais que comme pouvaient l'imaginer les pessimistes, les utilisateurs ne pourraient guère progresser en matière d'exploitation efficace et de comportements prévoyants (sécurité en matière de vie privée, par exemple). Pour endiguer ce courant, la solution que d'aucuns proposent aujourd'hui est donc d'apprendre aux jeunes à coder arguant que cet apprentissage les rendra moins vulnérables aux effets pervers de la vague numérique. Cet argument, rejeté au début des années 80, est aujourd'hui mis en avant avec la nécessité de développer chez les jeunes une pensée qualifiée d'informatique.

La programmation n'a évidemment pas cessé d'être enseignée dans les facultés d'informatique et à la Haute École[1]. C'est actuellement à ce niveau d'études qu'une majorité d'étudiants la découvrent. C'est un défi que leurs enseignants relèvent avec des fortunes diverses. À l'heure de cet hypothétique retour au code, il nous a paru intéressant d'observer les préoccupations didactiques des enseignants dans la formation à la programmation à la fin du secondaire et au début de l'enseignement supérieur. Nous nous sommes focalisés sur un élément particulier : le choix des outils didactiques. Nous précisons, bien entendu ce que nous entendons par outils didactiques. L'idée est de tirer des enseignements utiles dans le contexte où cet enseignement de la programmation est de plus en plus souhaité à l'école obligatoire dans de nombreux pays.

2 Objet de l'étude

Sans prendre position, dans un premier temps, sur le bien-fondé ou non d'un retour au code et sans contester que la programmation soit et reste sans doute à jamais au cœur de l'informatique indépendamment de l'évolution

1 Dénomination utilisée en Belgique pour désigner des études supérieures non universitaires.

galopante de celle-ci, notre étude s'intéresse aux choix didactiques opérés par les enseignants qui sont censés enseigner les bases de la programmation. Nous espérons ainsi pouvoir tirer des leçons utiles aux enseignant(e)s qui peuvent être amené(e)s à reproduire l'expérience à un niveau inférieur. Nous limitons volontairement notre étude à la programmation impérative à cause du lien étroit qu'elle entretient avec l'algorithmique et sachant que c'est essentiellement à ce paradigme de programmation que s'intéressent les projets actuels à l'école obligatoire.

2.1 Public-cible

L'étude a été réalisée en Communauté française de Belgique sur un public-cible constitué d'une dizaine d'enseignants d'université ou de Haute École qui enseignent un cours de base en programmation. Dans le souci de compléter notre information, nous avons également interrogé un enseignant du secondaire supérieur[2], titulaire d'un cours censé donner le goût, l'envie de programmer à des élèves qui se destinent à ce type d'études. Précisons que nous ne parlons pas ici de « sensibilisation » à l'acte de programmation comme on pourrait le faire à travers des expériences liées à l'informatique débranchée ou à travers les expériences constructivistes que permettent certains univers graphiques, mais bien d'un enseignement très objectivé de la programmation.

2.2 Outils didactiques

Nous nous intéressons directement aux outils didactiques choisis pour atteindre les objectifs d'enseignement établis, non sans questionner d'abord les enseignants sur ces objectifs et sur la difficulté subjective d'enseigner/apprendre la programmation. Nous avons donné au mot « outil » l'acception la plus générale qui soit, permettant ainsi aux enseignants de s'exprimer le plus largement possible. On trouvera donc derrière ce mot, le langage, mais aussi ce qui permet de produire du code : un environnement de développement ou un simple éditeur de texte (dans les situations où on se refuse à toute assistance de génération du code, par exemple),

2 Adolescents de 16 à 18 ans.

des diagrammes, des schémas, voire des objets réels qui pourraient participer à l'élaboration d'une meilleure compréhension. Nous nous intéresserons surtout à ce qui motive ce choix. Nous pensons qu'apprendre à programmer se détache difficilement d'un vrai langage, même si nous n'excluons pas de remettre cela en question s'il s'avère que certains se contentent d'un langage de description ou d'un pseudo-langage. Et donc, en particulier, la motivation du choix du langage est un élément important de notre analyse.

3 Questions de recherche

Au départ, notre étude souhaite essentiellement fournir des informations concernant l'importance relative des outils dans le choix de l'enseignant. Nous avions aussi l'ambition de trouver des réponses à des questions telles :

- pourquoi privilégier un pseudo-langage, un langage, un environnement de développement ?
- quelles sont les priorités accordées (programmation structurée, modélisation, structures de données. . .) dans le cadre d'un tel enseignement et pourquoi ?
- y a-t-il des outils qui facilitent la compréhension de certains concepts, qui favorisent certains comportements attendus du « bon programmeur » ?
- y a-t-il des outils qui perturbent la compréhension de certains concepts, qui favorisent certains comportements peu souhaités de la part du « bon programmeur » ?

Mais ce que nous avons découvert nous oblige à faire preuve d'une certaine humilité et montre, en tous cas, que ces questions passent souvent au second plan devant les urgences auxquelles les enseignants doivent faire face, notamment en ce qui concerne la gestion du contexte dans lequel ils évoluent. Nous donnons des détails dans les conclusions et perspectives.

4 Méthodologie

La préoccupation est double : (1) comprendre le choix des enseignants ; (2) les questionner sur les concepts considérés comme délicats à maîtriser par leurs élèves/étudiants et ceux qui leur paraissent compliqués à enseigner. L'idée est également de leur demander d'expliciter les forces et les faiblesses des outils de leur choix dans ces apprentissages/enseignements.

Pour cela, nous avons mené une série d'entretiens semi-dirigés auprès de dix enseignants : un enseignant du secondaire, cinq enseignants travaillant au sein d'une haute école et quatre enseignants, au sein d'une université[3]. Un guide d'entretien détaillé nous a permis d'entamer la conversation à partir de questions plutôt générales et de puiser dans des questions de détails si ceux-ci n'apparaissaient pas dans le discours de l'enseignant(e). L'établissement de ce guide d'entretien n'a pu se faire sans une certaine garantie de bonne couverture des réponses possibles. Pour le concevoir, nous nous sommes inspirés de certaines études (Lahtinen, Ala-Mutka & Järvinen, 2005 ; Milne & Rowe, 2002) qui s'intéressent à la perception subjective des difficultés liées à la compréhension des concepts, mais aussi d'autres études (Good, 2011 ; Guzdial, 2004 ; Pears *et al.*, 2007 ; Sorva, Karavirta & Malmi, 2013) qui proposent une analyse personnelle des outils ou encore de celles qui établissent des taxonomies de ces outils (Kelleher & Pausch, 2005). Les informations tirées de ces études nous ont permis d'orienter le questionnaire pour qu'en quelques questions et sous-questions bien calibrées, nous puissions établir rapidement la cohérence du choix de l'outil dans le contexte décrit. L'entretien était axé sur quatre questions principales :

- Quelles difficultés essentielles identifiez-vous dans l'apprentissage/l'enseignement de la programmation ?
- Quels outils didactiques utilisez-vous pour enseigner la programmation ?
- Quels objectifs poursuivez-vous dans votre cours de programmation ?

3 Pour garantir l'anonymat des enseignants interviewés, chacun d'eux s'est vu attribuer un nom fictif. Ce dernier, associé à un extrait présent dans cet article, permet notamment de mentionner le genre de l'auteur. Son niveau d'enseignement et son ancienneté sont également précisés.

- En quoi les outils choisis vous permettent-ils de rencontrer les objectifs fixés et les difficultés supposées ?

Pour chacune de ces questions, nous disposions d'une série de sous-questions nous permettant de compléter l'information lorsque celle-ci manquait de détails. À titre d'exemple, les sous-questions de la première question étaient :

- Quels sont les concepts (notions) complexes à enseigner ?
- Quels sont les concepts (notions) difficiles à maîtriser (comprendre) ?
- Pourriez-vous classifier ces difficultés, en traduire l'importance ?
- Au-delà des concepts, quelles sont les savoir-faire à acquérir ?
- Quels sont ceux dont l'acquisition n'est pas triviale ?

5 Analyse de données

Nous traiterons ces données en plusieurs points. À chaque fois, nous tenterons de suggérer des solutions didactiques.

5.1 Outils et motivation

Le premier concerne le rapport que peut entretenir l'étudiant, non pas à la programmation elle-même, mais aux méthodes et aux règles de bonne pratique que requiert cette activité. En effet, les étudiants ne perçoivent pas facilement l'intérêt de s'imposer une discipline de travail. La nécessité d'obtenir rapidement un résultat, et singulièrement un programme, prime sur le souci de qualité, voire de bon fonctionnement de celui-ci.

> […] leur demander d'utiliser un outil formel pour prouver qu'un programme fonctionne… je ne pense pas qu'ils comprennent l'intérêt. (Antoine, université, moins de 5 ans)

Ce n'est que dans les situations de constat manifeste des problèmes (le programme ne fonctionne pas ou ne donne pas les résultats attendus) que celles-ci peuvent trouver une part de justification. Et ce constat ne se fait

pas, par exemple, lorsque le programme semble fonctionner avec les données choisies, ce qui renforce la difficulté.

> […] ceux qui sont convaincus de savoir déjà programmer nous disent « ça sert à rien » jusqu'au jour où ça ne marche plus. (Antoine, université, moins de 5 ans)

Face à cette difficulté, on peut imaginer deux remèdes. Le premier est évidemment d'habituer l'étudiant, beaucoup plus tôt dans son cursus, à ce genre d'attitude, à ce savoir-être. Cela plaide évidemment en faveur d'une introduction de la programmation à l'école, sous une forme ou sous une autre, pourvu qu'elle inclue cette dimension de rigueur. Les enseignants eux-mêmes le suggèrent prétextant, sans doute à juste titre, devoir agir trop tard et avec des moyens inadaptés.

> C'est un peu gênant si nous, dans le supérieur, on reprend des choses qui sont du niveau primaire […] (Claire, haute école, plus de 15 ans)

Ainsi, des outils qui apparaissent comme pertinents pour apprendre à programmer se révèlent inadaptés à l'âge mental des étudiants.

> Je crois que nos étudiants seraient vexés si on choisissait des outils qui sont clairement à destination des petits et qu'on les utilisaient avec eux […] ils se diraient « mais ils nous prennent pour qui ceux-là ». (Claire, haute école, plus de 15 ans)

Le second remède, qui est d'ailleurs tout aussi valable dans la première situation, consisterait à choisir des problèmes (défis) types qui conduisent très rapidement à l'échec si cette rigueur n'est pas de mise. Mais les entretiens laissent penser que les enseignants ont peu l'occasion d'y réfléchir.

D'autres commentaires nous font penser que, dans des contextes où les aspects techniques sont très présents, la programmation est mieux acceptée peut-être parce qu'elle constitue une activité plus intellectuelle.

> Moi, je donne des cours de théorie aussi : réseau, etc. et je trouve que la programmation passe beaucoup mieux que ces cours-là parce que les élèves sont plus intéressés. (Nathalie, haute école, plus de 20 ans)

Lorsqu'ils pensent activité de programmation, beaucoup d'étudiants la considèrent comme une occasion de mettre en œuvre des projets qui leur plaisent. Certains outils sont alors perçus comme des entraves ou des éléments perturbateurs.

> Avant qu'ils ne conçoivent le programme en C, un des professeurs de labo (donc sur machine), demande aux étudiants de réaliser sur papier, le programme en pseudo-code. Moi je ne suis pas aussi rigoureux parce que je vois que ça les embête. (Tony, haute école, plus de 20 ans)

5.2 La question du choix

Rappelons que nous souhaitions donner à la notion d'outil l'acception la plus large possible et y inclure tous les éléments qui peuvent constituer une aide ou un support à l'apprentissage de la programmation. On s'intéresse donc à des outils matériels autant qu'à des outils logiciels et en particulier, au(x) langage(s) qui permettent de transformer le fruit d'une réflexion algorithmique en un programme exécutable. Il est à remarquer que certains concepts, voire certaines étapes de la programmation sont vus par certains comme des outils.

> Les spécifications sont à la fois compétences et outils. Même chose pour invariants… c'est plutôt un outil qui est censé aider les étudiants à bien faire leurs boucles […] (Antoine, université, moins de 5 ans)

Mais nous les avons exclus de la réflexion en tant que tels. Le moins que l'on puisse dire, c'est qu'ils sont d'une très grande variété. On ne retrouve pas vraiment de consensus à ce propos. Les outils choisis sont en lien avec la stratégie d'enseignement, voire les impératifs de formation. Parmi les outils évoqués, le plus dépouillé en quelque sorte est la feuille de papier. Elle témoigne d'une volonté délibérée de l'enseignant de faire réfléchir les étudiants. Ce témoignage montre que l'attitude de celui-ci n'est pas toujours le repli face à un outil austère.

> Le meilleur outil de l'informaticien c'est la feuille de papier, le crayon et la gomme. Ça reste dans l'optique « je réfléchis avant de coder ». La plupart des exercices sont faits sur papier et ils ont des séances de laboratoire qui leur permettent de se confronter à la machine. (Nicolas, université, plus de 5 ans)

À l'autre bout du spectre, on trouve le langage. Certains considèrent qu'apprendre à programmer, c'est maîtriser un langage de programmation.

> Moi dans les cours, je n'utilise pas d'outils intermédiaires, je n'utilise pas de représentations graphiques, de structures algorithmiques ou quoi que ce soit d'autre, je reste dans le concret, c'est le code. (Cédric, université, plus de 10 ans)

Et puis, il y a ceux qui ne mettent pas vraiment de frontières entre algorithmique et langage.

> En première année, ils ont un cours d'algorithmique fort orienté vers le C. Les exemples, c'est déjà presque de l'écriture C. (Nicolas, université, plus de 5 ans)

Entre les deux, on observe presque autant de conceptions de l'enseignement de la programmation que d'enseignants. Certains font travailler les étudiants dans des environnements de développement sophistiqués. Le choix est guidé de diverses manières.

> Pour Eclipse, j'ai eu le choix. Le directeur m'a dit : « tu peux utiliser soit Eclipse, soit Netbeans ». Il y a une liberté au niveau du choix des outils. (Tony, haute école, plus de 20 ans)

Ce type de choix effectué plus volontiers dans les Hautes-Écoles que dans les Universités trouve souvent sa justification dans la proximité de la formation avec le monde du travail qui est d'ailleurs parfois partie prenante de cette formation.

> Je suis allé voir ceux qui étaient les plus demandés sur le marché. Je les ai proposés au directeur qui a demandé aux consultants externes ce qu'ils en pensaient et ils ont validé le choix. […] ce ne sont pas les profs en interne qui ont choisi les outils, c'est une proposition faite qui a été validée par les extérieurs, par les gens du métier. (Tony, haute école, plus de 20 ans)

Pour beaucoup d'enseignants de Haute École, le choix des outils ne peut se faire sans un regard vers les entreprises. Il est plus important que l'étudiant connaisse l'outil, même s'il ne s'en sert pas de manière optimale.

> De toute façon, on est quand même bien obligé de passer par les outils les plus courants actuellement. On ne peut pas faire autrement. (Nathalie, haute école, plus de 20 ans)

Quand un choix est fait par la communauté des enseignants, il est parfois revu à la lumière des commentaires des sociétés susceptibles d'engager ces étudiants.

> C'est vrai qu'on se disait qu'avec le C++, une fois qu'ils ont fait ça (*sic*), ils peuvent passer partout. On avait aussi des retours des sociétés qui disaient « écoutez, le C# est quand même fort utilisé et quand ils viennent chez nous, ils ne le connaissent pas ». Donc, nous avons décidé il y a trois ans de changer et de passer au C#. (Nathalie, haute école, plus de 20 ans)

La préoccupation est moins présente à l'Université, où l'objectif est davantage tourné vers la rigueur que vers un choix de langage. Néanmoins, lorsque le cours s'adresse à des étudiants dont la finalité des études n'est pas l'informatique, on peut la retrouver sous des justifications assez proches.

> […] je pense qu'au final le C garde son intérêt, notamment pour les ingénieurs civils puisque c'est de toute façon une exigence des électriciens. Ceux qui iront en électricité ou même en méca ont besoin du C à un moment donné pour les systèmes embarqués etc. Donc autant voir du C. (Nicolas, université, plus de 5 ans)

Sans se préoccuper de l'acte de programmation lui-même, le regard posé est déjà celui d'une informatique utile.

> Le langage C est un choix imposé par la Faculté de Sciences. La formation en math est très appliquée et donc, à partir de la troisième année, ils sont censés utiliser pas mal d'outils informatiques, pas mal de librairies, etc. Une bonne partie des librairies qu'ils utilisent est écrite en C. Et donc, il y avait le souhait de voir du C dès le début pour justement enlever une difficulté chez les étudiants. (Cédric, université, plus de 10 ans)

Quand des difficultés surgissent, il est fréquent que les enseignants changent d'outils. Mais ils sont désemparés quand les changements qu'ils opèrent ne donnent pas les résultats escomptés. Ils constatent alors que la motivation n'est pas nécessairement liée à la rigidité potentielle de ceux-ci.

> Pendant des années, on a fait du Pascal et on a cru que les étudiants n'accrochaient pas, parce qu'ils avaient l'habitude d'un monde informatique très convivial. Et nous, on leur proposait une petite fenêtre sous DOS, en noir, où les accents ne passaient pas correctement ou bien c'était beaucoup de chipotage. On s'était dit que c'était ça qui freinait les étudiants. Donc on a voulu faire quelque chose de plus joli en faisant du HTML. On voyait la même matière qu'avant au travers du JavaScript : les boucles, les conditionnelles. . . et ça n'a rien changé du tout. C'est toujours aussi difficile alors qu'on a essayé de faire des choses beaucoup plus jolies. Ça n'a pas été la solution. (Claire, haute école, plus de 15 ans)

Au bout du compte, il semble que les outils les plus élémentaires (papier, crayon, gomme) soient choisis par ceux qui veulent encourager la réflexion. Les langages et environnements de développement sont surtout choisis pour que l'étudiant y soit accoutumé dans le monde du travail ou de formation qui l'attend sans qu'ils lui permettent nécessairement d'acquérir de bons réflexes de programmation. Une chose semble certaine, le choix du langage n'est pas vraiment guidé par le souci d'apprendre à programmer.

5.3 Le rapport des outils à la didactique

Au vu de ce qui précède, on peut se demander quelle place occupe l'intérêt didactique des outils dans le choix de ces derniers. D'autre part, les contraintes qui pèsent sur l'enseignement de la programmation dans l'enseignement supérieur ne semble plus pouvoir laisser beaucoup de place à cette préoccupation, ce qui serait un argument de poids dans la nécessité d'enseigner à tout le moins l'algorithmique dans l'enseignement secondaire. Nous traitons de cette question dans les conclusions et les perspectives. Nous analysons ici, dans le discours des enseignants, s'ils établissent un lien entre les outils et leur rôle facilitateur ou non de l'enseignement de la programmation. Cette question est à mettre en rapport avec la représentation que se font les enseignants des problèmes liés à cet enseignement. Nous la décrivons en deux ou trois points.

Il y a d'abord la manière dont ils jugent leur public. Elle est parfois dure et sans nuance.

> […] il y a des étudiants très bons, les 30 % qui réussissent… eux, je pense qu'on peut faire n'importe quoi avec eux, ça marchera toujours. (Claire, haute école, plus de 15 ans)

Elle peut traduire un certain fatalisme.

> […] il y a des élèves qui sont doués pour ça et il y en a qui le sont moins. (Étienne, secondaire supérieur, plus de 20 ans)

Elle est parfois mieux identifiée sans qu'on laisse entrevoir de solution au problème sinon celle du temps qui passe.

> Enseigner la programmation reste quelque chose de difficile puisque, soit les étudiants qui perçoivent directement comment faire, ils ont la logique et puis on a en a qui restent bloqués. […] j'attends que mes étudiants mûrissent. Il faut leur laisser le temps. (Julie, haute école, plus de 15 ans)

Il y a la manière de traiter les problèmes lorsqu'ils se présentent. Devant un constat totalement négatif, on s'oriente fréquemment vers des modifications d'outils, de stratégies qui ne s'avèrent pas nécessairement payantes, les raisons du découragement ou du manque d'intérêt étant parfois mal identifiées.

> Aucun outil ne nous satisfait actuellement. […] nous cherchons toujours pour les étudiants plus faibles et qui sont très bons dans les autres matières… Nous avons déjà changé souvent nos méthodologies parce que nous étions persuadés que le Pascal ne fonctionnait pas parce que ce n'était pas joli (*sic*). […] le parcours est long et ça passe par des apprentissages qui sont nettement moins sexy. (Claire, haute école, plus de 15 ans)

Dans les meilleurs cas, on se rend compte que les outils sont limités, mais c'est plutôt rare et la manière d'exprimer la difficulté ne laisse pas forcément entrevoir de manière de contourner ces limites.

> Quand elle *(NDLA : ma collègue)* travaille en pseudo-code, il y a des choses qu'elle sait faire, que je ne sais pas faire en C. (Julie, haute école, plus de 15 ans)

Parfois on devine tout de même une possibilité de contourner la difficulté.

> On peut déjà faire pas mal de choses, pas tout malheureusement, […] avec Algobox et donc quand on programme, enfin quand on traite des problèmes mathématiques, en général on arrive à les coder après par Algobox. Il y a parfois certaines résolutions où je leur dis « là écoutez c'est la limite de l'outil, vous ne pourrez pas le faire tel quel avec Algobox », mais voilà. (Anne, haute école, moins de 5 ans)

Lorsque les avantages d'un outil sont mis en évidence, c'est plutôt pour dégager des facilités d'usage ou un intérêt pédagogique, même si on devine qu'il y a aussi des bénéfices du côté de la facilité de compréhension et de l'aide à la programmation.

> […] on passe très vite à un pseudo-code vraiment en français car mes étudiants ont du mal parfois avec l'anglais, […] un pseudo-code classique « tant que. . . faire » avec des choses très compréhensibles et alors qu'au début, je les orientais vers le Pascal, finalement je trouve que ça amène des difficultés et pas beaucoup d'avantages alors je travaille maintenant avec Algobox qui est très très facile à utiliser et qui est très adapté. (Anne, haute école, moins de 5 ans)

En gros, les enseignants ne choisissent pas un outil parce qu'il va les aider à lever une difficulté, mais plutôt parce qu'un autre outil ne semble pas donner de bons résultats ou parce qu'ils pensent que le fait de ne pas l'avoir rencontré risque de constituer un obstacle dans la vie professionnelle future de leurs étudiants.

La question du rapport des outils à la didactique semble perdre son sens dès lors que la question de la didactique n'est pas posée. La programmation s'enseigne sur base d'une expérience personnelle, mais on sent l'absence

cruelle de réflexion sur la manière de solutionner les problèmes de compréhension bien qu'on le souhaite ardemment. Ce manque est compréhensible dès lors que la formation des informaticiens n'inclut pas la possibilité de suivre une filière didactique et donc d'identifier un lieu où cette réflexion, cette recherche peut avoir lieu.

Bref, il est difficile de qualifier les outils dont on nous a parlé de didactiques, dès lors qu'il semblent davantage générer des problèmes que d'apporter un support aux difficultés des étudiants à gérer l'activité de programmation.

6 Conclusion

Les outils sont choisis en fonction de toute une série de critères de laquelle la facilité d'apprentissage semble absente. On choisit un outil parce que son dépouillement est considéré comme formateur, parce qu'une Faculté le souhaite au nom d'intérêts propres, parce que le monde du travail le demande, etc. Nous n'avons pas entendu parler d'un outil choisi parce que, du point de vue de sa conception, il facilitait l'apprentissage de la programmation. Que du contraire, certains langages comme Pascal qui ont longtemps été considérés comme pédagogiques sont rejetés parce que trop vieux et se voient préférer des langages plus actuels, même si ceux-ci présentent des caractéristiques qui pourraient être considérées comme invalidantes. Par exemple, l'absence de typage avec Python a été soulignée sans que cela paraisse être une difficulté.

L'étude que nous avons menée a été quelque peu polluée par un problème qui est bien identifié. L'étudiant qui entame des études supérieures et qui est confronté à un cours de programmation n'y est pas correctement préparé. Plusieurs phénomènes sont mis en avant : (1) il n'a pas une représentation correcte de la discipline, voire du métier qui peut en découler (Greening, 1998 ; Grover, Pea & Cooper, 2014 ; Grover, Rutstein & Snow, 2016 ; Hewner, 2013 ; Hewner & Guzdial, 2008) ; (2) il a un vécu numérique qui n'est pas nécessairement un capital car les représentations qu'il se fait de certains concepts informatiques sont mauvaises ou biaisées ; (3) et c'est sans doute un élément-clé, il n'a pas vécu (ou si peu), lors de ses études secondaires, de situations d'apprentissage qui lui fassent

correctement percevoir le contexte dans lequel il va être baigné et notamment, celui de systèmes qu'il devra lui-même concevoir et qui fonctionnent de manière formelle, autrement dit très différente de ce qu'il connaît en tant qu'être humain. Ces systèmes exigent rigueur et esprit logique et il semblerait que ces deux dimensions soient beaucoup trop peu développées à l'école secondaire/obligatoire.

Les enseignants du supérieur se retrouvent donc devant des difficultés qu'ils n'ont pas prévues. Face à ces difficultés, et souvent dans l'urgence de devoir obtenir des résultats, ils parent au plus pressé. Nous avons trouvé fort peu d'enseignants qui soient vraiment satisfaits du travail qu'ils font. Mais pour expliquer les difficultés rencontrées, ils cherchent assez souvent des explications dans le passé des étudiants, dans leur vécu comme dans la formation qu'ils ont ou pas reçue. Il est donc remarquable de constater qu'une réflexion d'ordre didactique n'est jamais ou alors très rarement envisagée. L'enseignant n'évoque pas de stratégie pour faire mieux comprendre un concept qui pose problème, pour autant que celui-ci ait été identifié.

> On voit de la théorie avec eux, mais on voit la théorie sur des exemples et on leur donne après plutôt des exercices à faire. (Nathalie, haute école, plus de 20 ans)

On compte davantage sur la répétition des situations, une sorte de drill, pour atteindre la compréhension.

> Il faut donner beaucoup d'exercices pour qu'ils comprennent. (Nathalie, haute école, plus de 20 ans)

Lorsque surgit une difficulté, elle est parfois supprimée et donc évitée ou bien elle donne lieu à un changement d'ordre pédagogique. Faire autre chose, autrement…

> Honnêtement, je ne suis pas certain de maintenir les invariants dans le cours dans les années qui viennent parce qu'au fond, quel est l'intérêt si de toute façon ils ne comprennent pas du tout… est-ce que je ne perds pas mon temps sur ce concept-là ? (Antoine, université, moins de 5 ans)

Même en fin de secondaire, lorsque des cours existent, la difficulté d'apprentissage conduit souvent à un aveu d'impuissance.

> Certains élèves […] inversent la partie gauche et la partie droite de l'affectation. Ça, c'est vrai que j'ai déjà vu plusieurs fois. […] je leur ai déjà dit et ils refont l'erreur

> après, comme si dans leur tête il y avait quelque chose qui ne se passait pas bien… Je ne sais pas trop expliquer. Ça ne se corrige pas toujours aussi facilement que ça. (Étienne, secondaire supérieur, plus de 20 ans)

Le vrai problème ne serait-il pas que l'initiation à la programmation arrive bien trop tard et que les enseignants du supérieur ne bénéficient pas de conditions idéales pour exercer leur métier comme, par exemple, apprendre aux étudiants à concevoir des systèmes efficaces, sachant qu'ils ont déjà une bonne idée de ce qu'est programmer.

> On est fort démuni si on veut faire quelque chose pour adultes qui ressemble quand même à quelque chose pour adultes. Le problème aussi c'est que toutes ces matières-là sont enseignées par des professeurs pour qui ce n'est pas difficile. […] on tombe des nues à voir la difficulté qu'ont les étudiants de comprendre un « si alors sinon ». Pour tous les profs […] c'est quelque chose qui n'a jamais posé problème. Donc il y a une fracture entre le niveau des étudiants et le niveau des professeurs qui leur donnent cours. (Claire, haute école, plus de 15 ans)

Ce constat pessimiste nous invite à envisager, sinon des solutions, une série de recommandations qui pourraient être prises en compte dans la perspective d'une introduction à la pensée informatique dans l'enseignement obligatoire, pour autant qu'elle soit à l'ordre du jour.

7 Perspectives

Il s'avère que les questions posées sur le choix des outils ont souvent été occultées par une énorme difficulté. Beaucoup d'enseignants sont amenés à donner un cours requérant des compétences fondamentales, et d'ailleurs pas forcément liées à l'informatique de manière directe, mais qui sont des compétences requises pour pouvoir développer un cours à ce niveau dans de bonnes conditions. D'une certaine manière, cette situation rappelle celle qu'on a connue voici une trentaine d'années, dans des cours et des formations censés apprendre un usage pertinent des outils progiciels. Alors qu'on visait à développer une méthodologie bien établie de conception des produits, on se heurtait à des problèmes et des besoins beaucoup plus basiques tels un manque de familiarisation avec le clavier, des difficultés à gérer les productions et à les organiser en fichiers et en dossiers, etc. Il en résultait

souvent, pour l'enseignant comme pour l'apprenant, de la frustration, de l'incompréhension et du découragement.

> Tout ce qui est fichier curieusement aussi ils ont du mal. On s'attendrait *a priori* à ce que les jeunes qui sont réputés capables numériquement s'en sortent… et non, il y en a quand même pas mal qui ne savent pas ce que c'est un dossier, qui savent à peine ce que c'est un fichier et qui confondent les deux. (Antoine, université, moins de 5 ans)

Ce sont ces sentiments qui prévalent aussi dans le discours des enseignants. L'absence de compétences fondamentales (au sens où il est difficile de construire quelque chose sans elles) engendre le plus souvent un désintérêt de l'étudiant pour les activités qu'on lui propose. Parmi ces compétences, on relève essentiellement une capacité d'abstraction qui n'est pas acquise par tous au sortir des études secondaires, mais aussi et surtout, un manque d'habitude à traiter des problèmes ou des tâches de manière systématique, le mot « système » prenant ici tout son sens. Si les jeunes sont habitués aux ellipses, aux raccourcis de langages, aux sous-entendus, ils ont rarement sinon jamais rencontré des situations de travail ou d'apprentissage qui leur imposent une discipline semblable à celle que nécessite la programmation (au sens large) des systèmes informatiques et qui ne peut se justifier que dans la qualité finale d'un produit. Ce manque de discipline ou plutôt de perception correcte de la manière dont fonctionnent ces systèmes est probablement responsable d'un énorme bouchon d'incompréhension qui pourrait sauter en incluant, dans une éducation numérique au secondaire, une série d'activités bien pensées dans ce sens. Et les idées ne manquent pas en la matière. Ce qui manque, c'est probablement l'intégration d'un programme bien élaboré dans un cursus accueillant (entendez : qui offre un nombre d'heures suffisant). Espérons que les choses finiront par bouger dans ce sens.

Cela dit, dans tous les contextes où l'informatique et la programmation sont rendus obligatoires à l'école du même nom, il ne faudrait pas rater cette occasion de faciliter une transition vers des études dans le domaine. Faire des élèves, à ce stade, des programmeurs hors pair est probablement moins urgent que de les « éduquer » à la maîtrise de ces systèmes. Et puis, ce qui n'apparaît pas clairement dans le discours, c'est ce réflexe qui conduit l'enseignant à analyser une situation-problème et à tâcher de la solutionner par une approche didactique différente. Trop souvent, c'est l'incompréhension qui règne et une certaine maladresse dans la manière d'envisager des

solutions. Mais comme nous l'avons dit, c'est probablement à un niveau inférieur que ces difficultés d'apprentissage doivent être abordées.

N'empêche, pour revenir à la question principale de cette étude, que peu d'éléments viennent corroborer l'idée d'un choix d'outils pour leurs qualités didactiques.

Références

Good, J. (2011). Learners at the wheel : novice programming environments come of age. *International Journal of People-Oriented Programming (IJPOP)*, *1*(1), 1–24.

Greening, T. (1998). Computer science : through the eyes of potential students. In *Proceedings of the 3rd australasian conference on computer science education* (pp. 145–154).

Grover, S., Pea, R. & Cooper, S. (2014). Remedying misperceptions of computer science among middle school students. In *Proceedings of the 45th acm technical symposium on computer science education* (pp. 343–348).

Grover, S., Rutstein, D. & Snow, E. (2016). What is a computer : What do secondary school students think ? In *Proceedings of the 47th acm technical symposium on computing science education* (pp. 564–569).

Guzdial, M. (2004). Programming environments for novices. *Computer science education research*, *2004*, 127–154.

Hewner, M. (2013). Undergraduate conceptions of the field of computer science. In *Proceedings of the ninth annual international acm conference on international computing education research* (pp. 107–114).

Hewner, M., & Guzdial, M. (2008). Attitudes about computing in postsecondary graduates. In *Proceedings of the fourth international workshop on computing education research* (pp. 71–78).

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming : A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, *37*(2), 83–137.

Lahtinen, E., Ala-Mutka, K. & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. In *Acm sigcse bulletin* (Vol. 37, pp. 14–18).

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information technologies*, *7*(1), 55–66.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., . . . Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, *39*(4), 204–223.

Sorva, J., Karavirta, V. & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, *13*(4), 15.