Le code va changer

Olivier Alexandre

RESET

Recherches en sciences sociales sur Internet

11 | 2022
Codes. L'informatique comme elle s'écrit
Dossier

Le code va changer

Les développeurs de la Silicon Valley face aux incertitudes de la programmation

The code will change. Silicon Valley developers address the uncertainties of programming

OLIVIER ALEXANDRE

https://doi.org/10.4000/reset.3498

Résumés

Français English

Reposant sur une enquête par entretiens, observations et l'exploitation d'un corpus de quarante enquêtés travaillant dans la Silicon Valley, cet article traite des stratégies adoptées par les développeurs pour faire face à l'incertitude de la programmation informatique en tant qu'activité professionnelle. Il porte sur la structuration temporelle de cette activité, l'incertitude se retrouvant à trois échelles de temps: les sessions de programmation, les projets et les carrières. Pour faire face aux aléas et impasses au cours des sessions de travail, les développeurs font appel à des renforts. À l'échelle du projet, ils s'appuient sur des médiations. Enfin, pour conjurer l'incertitude au sein d'une industrie caractérisée par la complexité et la volatilité des valeurs, ils cherchent à initier des tournants dans leur carrière professionnelle. À travers l'analyse de ces trois types de stratégies (renforts, médiations et tournants), cet article propose un cadre d'analyse temporelle du travail des développeurs.

Based on in depth-interviews, observations and a corpus analysis of forty developers working in Silicon Valley, this article deals with the uncertainty of computer programming as a professional activity. The article focuses on the temporal structure of this activity and the uncertainty of the outcome from the developers perspective for their work, project and career. To reduce this uncertainty, developers call on support personnel during their programming sessions. To develop their projects, they rely on mediations. Finally, to ward off uncertainty in an industry characterized by complexity and volatility, they try to initiate turning points in their professional careers. Through the analysis of these three types of strategies (support personnel, technological mediations and turning points), this article proposes a framework for the temporal analysis of developers'work.

Entrées d'index

Mots-clés : développeurs, Silicon Valley, incertitude, contrôle, renfort, médiations, tournant, temps, projection

Keywords: developers, Silicon Valley, uncertainty, control, support, mediations, turning points, time, projection

Texte intégral

Introduction

1 Dans la Silicon Valley, le travail, l'entreprise, le professionnalisme et la carrière constituent le cœur des préoccupations, des valeurs et des systèmes narratifs (Alexandre, Coavoux, 2021) d'une industrie où le « *software* » est devenue le domaine central[1]. Toutefois, l'appréhension de cet univers dans les termes de la sociologie des professions ne va pas de soi. En effet, les travailleurs du code, particulièrement les développeurs auxquels cet article s'intéresse, ne peuvent s'appuyer sur des standards en termes de formation, d'organisation professionnelle et la réglementation. En premier lieu, il n'existe pas de système de formation unifié. Dans la Silicon Valley, les codeurs professionnels déclarent avoir appris à coder par différents biais : autodidaxie, apprentissage via un ami ou un membre de la famille durant l'enfance ou l'adolescence, programmes de formation scolaires et parascolaires, passage par les départements de *Computer Science*, programmes de formation allant de deux mois au postdoctorat. Cette multiplicité des formations est d'autant plus frappante que les professionnels de la Silicon Valley sont issus pour moitié d'un autre pays que les Etats-Unis (Saxenian, 1999). Outre l'absence de standard académique, nombre de développeurs déclarent ne pas avoir appris la programmation via des systèmes et cycles de formation de quatre à six ans, instaurés suivant les principes des écoles d'ingénieur héritées du 19e siècle (Grelon, 1997). Deuxièmement, ces travailleurs du code mettent à distance les logiques d'organisation corporative. Les démarcations entre différents « territoires » professionnelles, observées notamment par A. Abbott dans les hôpitaux (Abbott, 1988), n'y ont pas d'équivalent, les frontières entre développement, analyse de données, design, entrepreneuriat, développement de produit demeurant relativement lâches dans les entreprises technologiques nouvellement créées. Les travailleurs passent parfois de l'une à l'autre de ces tâches au cours d'une même journée. Il est également fréquent de les voir passer d'une fonction à l'autre au fil de leurs carrières, un développeur devenant, entrepreneur, puis investisseur, etc. L'évolution, l'agilité et la fluidité y sont régulièrement présentées comme les fondements de la culture de travail (Becker, 1999), à l'opposé du système des juridictions nées de rapports de force entre corps professionnels rivaux, par exemple entre les médecins et les homéopathes à la fin du 19e siècle dans l'Etat de New York (Abbott, 2003). Troisièmement, en raison d'une évolution relativement rapide des technologies, les titres (*software engineer, software developer, etc.*) et spécialités (langages de programmation, *hadoop*, iOS, Androïd, etc.) ne font pas l'objet d'un contrôle par une institution ou une organisation. La dynamique de croissance de l'industrie explique l'inflation des dénominations au fil des années : « *software developers* » (SD), « *dev ops* », « *software engineers* », « *data analysts* », « *machine learners* », « *software architects* » ; et des grades : SD junior (sans expérience), SD I (avec diplôme dans l'informatique), SD II (avec x années

d'expérience), SD III (responsable de projet), *SD senior* (responsable d'équipe) », *software manager* », jusqu'au *Chief Technical Officer* et *VP engineering* plus hautes fonctions technique au sein des entreprises de la région. De nouvelles spécialisations apparaissent et disparaissent, au même rythme que les technologies. Dans ce contexte, être développeur recouvre une double signification : développer des logiciels et développer de manière continue ses compétences, ses expériences et ses savoir-faire.

2 Cette focalisation sur les pratiques explique en partie que les travaux de référence des *hacker studies*, tirés en partie d'enquêtes réalisées dans la péninsule de San Francisco (Levy, 1984 ; Flichy, 2001 ; Turner, 2006 ; Coleman, 2013 ; Chrysos, 2015 ; O'Mara, 2019) ont peu évoqué les enjeux professionnels de l'activité de programmation, au profit de thèmes tels que la liberté (Levy, 1984 ; Coleman, 2013), le plaisir (Turkle, 1984), les échanges désintéressés (Kelty, 2008), l'idéal communautaire (Turner, 2006) ou la collaboration (Berrebi-Hoffmann et al., 2018). Ce mode d'appréhension renvoie aux systèmes de valeur des développeurs. Comme l'a souligné Paris Chrysos (Chrysos, 2015), ces derniers mettent en avant leur état d'esprit, les projets qu'ils souhaitent réalisés, ce qu'ils savent faire ou veulent apprendre, bien plus que par leurs emplois, leurs diplômes. Dans la tradition des « *hackers* », le fait même d'envisager le développement informatique comme une profession constituerait une forme dégradée du développement logiciel dont les fondements sont l'imagination, la rêverie, le détournement et l'utopie concrète (Chrysos, 2015). Dès lors, la question qui se pose, sur laquelle porte cet article, est de savoir comment les développeurs assurent le contrôle d'une activité faiblement professionnalisée[2] ? Les *software studies*, en dépit de leurs apports, laissent en suspend cette question, puisque, des trois sous-courants identifiables aux sciences sociales, le premier s'attache à l'étude des concepteurs de logiciel dans le cadre d'une histoire plus générale de l'informatique (Mahoney, 2011) ; le second réinscrit le logiciel dans l'étude plus globale des médias pour souligner ses spécificités (Manovich, 2001) ; et le troisième met en lumière la manière dont le logiciel recompose les modes d'organisations sociales au-delà des programmes et de leurs concepteurs (MacKenzie, 2006). Qu'ils s'agissent d'histoire, d'études des médias ou de sociologie, la question de l'organisation et du contrôle professionnels de l'activité de programmation y est donc un angle mort. L'hypothèse soutenue dans l'article est la suivante : en l'absence d'institutions externes (formations, juridictions, corporations), les développeurs recourent à des modes de contrôle internes, intégrés au cours de l'action. La logique de ces contrôles internes se comprend à partir des propriétés temporelles du développement logiciel.

3 En effet, le travail des développeurs consiste à se projeter dans le temps. Toutefois, cette projection est particulièrement difficile à organiser, pour trois raisons : la fragilité de la production, l'évolution de l'environnement technologique et les changements organisationnels. Les développeurs sont confrontés quotidiennement à une multitude d'aléas et d'imprévus. Indépendamment de leur niveau de compétences, ils doivent faire face à des bugs, des conflits de version, des failles de sécurité, sans savoir à l'avance s'il s'agit d'un bref contretemps ou d'une impasse définitive. En outre, leur environnement technologique ne cesse d'évoluer. Ils doivent s'approprier régulièrement de nouveaux langages, cadres de programmation, standards, architectures, infrastructures, paradigmes, à une vitesse alignée sur la loi de Moore (Lécuyer, 2020)[3]. Or, ce processus continue d'apprentissage est entravé par des contraintes organisationnelles : restructuration d'équipes, changements de stratégie, rachats ou cessations d'activité, qui sont monnaie courante dans la Silicon Valley. Pour ces trois raisons, les développeurs savent que les projets dans lesquels ils s'engagent ont toutes les chances de ne pas aboutir. Dans le langage de l'économie, les développeurs doivent donc composer avec une « incertitude », une notion distincte selon P. Knight de celle de

« risque » (Knight, 1921). Selon l'économiste, le risque est relatif à une distribution d'états connus et probabilisable, tandis que l'incertitude est le produit d'une situation où la distribution d'états futurs est inconnue et impossible à connaître[4]. Pour y faire face, les développeurs recourent à ce que H. White nomme des « prises » ou « moyens de contrôle » : « le contrôle consiste à trouver des prises parmi les entités environnantes. Ces prises sont une position qui permet de s'orienter en relation avec d'autres entités »[5] (White, 2008 : 1). Après une présentation des données qui ont servies à l'analyse (I), cet article traite des stratégies des développeurs de la Silicon Valley pour établir des prises et des moyens de contrôle. Il traite de trois types de stratégies, relatives à des trois séquences d'action : les sessions de programmation (court terme), les projets (moyen terme) et les carrières (long terme). Durant les sessions de programmation, les développeurs sollicitent des renforts pour faire face aux imprévus qu'ils rencontrent (II). Pour mener à bien leurs projets, ils s'appuient sur des médiations (Hennion, 2015) afin de limiter les coûts temporels de production et d'information, et s'orienter plus rapidement et efficacement (III). Afin d'échapper aux aléas et à l'arbitraire des carrières, ils tentent enfin d'initier des « tournants » (Abbott, 2001) dans leur trajectoire (IV). En mobilisant des renforts, en recourant à des médiations et en cherchant à réaliser des tournants, les développeurs tentent ainsi de conserver une prise sur une activité marquée par une irréductible incertitude.

Méthodes et données. Contourner la dimension privée du code

4 Cet article repose sur une enquête de terrain réalisée dans la Silicon Valley entre 2015 et 2018. Comme la plupart des travaux sur la programmation, l'enquête s'est heurtée à la difficulté d'observer systématiquement et durablement l'activité de programmation, en raison de son invisibilité, de sa technicité (je n'avais pas reçu de formation préalablement dans ce domaine) ou du fait qu'elle s'opère le plus souvent au sein de la région dans des lieux privés : espaces domestiques, entreprises, lieux d'enseignement, etc. Cet exercice privé du code s'est révélé problématique pour trois raisons. Tout d'abord, la littérature sur les *hacker studies* ont privilégié l'étude du « libre », de ses pratiques (Murillo, 2020) et de sa philosophie (Broca, 2013) au détriment des logiciels propriétaires et des logiques marchandes (Alcaras, 2020), et en dépit de l'intégration par les entreprises du référentiel hacker (Alexandre, 2021). Ce différentiel de traitement explique la rareté des considérations méthodologiques sur la manière de dépasser les contraintes relatives à l'observation dans les espaces privés. Or, les restrictions d'accès aux espaces de travail ont limité les possibilités d'observation du code en train de se faire, notamment en raison des conditions d'accès aux entreprises – besoin d'y être invité – et des restrictions juridiques, puisque dans les entreprises « matures » de la Silicon Valley, l'entrée sur les sites est conditionnée systématiquement à la signature d'un accord de non-divulgation (*No Disclosure Agreement* – NDA).

5 Le statut d'affilié à Stanford m'a toutefois permis d'accéder à des séances de formations, des *hachatons*, des *meetups*, un *bootcamp* et trois conférences professionnelles de développeurs. Les données collectées se sont révélées relativement difficiles à exploiter, notamment du fait de l'impossibilité de conserver des données de programmation ou en raison des contraintes d'enquête au cours de ces événements. Toutefois, ces premières observations ont été l'occasion d'établir des contacts avec des développeurs, puis d'en rencontrer d'autres par le jeu des interconnaissances, travaillant pour des entreprises situées à des niveaux de développement variés, soit suivant les catégories de financement en vigueur dans la Silicon Valley (croisés avec les

informations présentées sur Crunchbase, une plateforme compilant des données sur les entreprises dans les nouvelles technologies) : des entreprises en *seed* (amorçage, le plus souvent sans employé), en serie A (phase d'optimisation consistant à développer les leviers de la traction, ce qui implique des recrutements techniques, avec quelques dizaines d'employés), en serie B (consistant à se focaliser sur le développement commercial, avec une équipe comprise entre 50 et 100 personnes), en serie C (il s'agit de perfectionner rapidement l'activité dans une perspective de retour sur investissement, nombre d'employés entre 100 et 500), des licornes (entreprises valorisées à plus d'un milliard de dollars à l'occasion de nouvelles levées de fonds, nombre d'employés entre 500 et 5000) et de grandes entreprises (cotées en bourse, plusieurs milliers d'employés). J'ai réalisé sur cette base une première série d'entretiens avec trente-huit professionnels dont la programmation informatique constituait l'activité principale (voir tableau 1).

T1. Tableau de la première vague d'entretiens avec des développeurs

Genre	**Hommes**	**Femmes**
Nb	35	3

Age	**-20 ans**	**21-30**	**31-40**	**41-50**	**51-60**	**+60**
Nb	2	10	15	9	1	1

Statut	***Software engineer***	***Manager***	***CTO***	***CEO***	***Data analysts***	***Academics***
Nb	12	5	12	5	2	2

Organisation	**« *Bootstrap* »**	**+5**	**+50**	**+100**	**+1000**	***Universities***
Nb	3	5	6	8	14	2

Nationalité	**US**	**Europe**	**Afrique**	**Asie**	**Moyen-Orient**	**Caraïbes**
Nb	16	15	2	2	2	1

6 Au cours de ces entretiens, il a rarement été possible d'évoquer en détail les projets sur lesquels les développeurs travaillaient, particulièrement pour les salariés des grandes entreprises tenus par contrat à conserver secret le contenu de leur travail. Il m'a ainsi été demandé à plusieurs reprises de suspendre l'enregistrement ou de maintenir la confidentialité des informations communiquées. Ces limites et contraintes ont appelé une stratégie alternative de recherche.

7 À partir des premiers matériaux rassemblés, j'ai ainsi constitué un corpus d'enquêtés à partir des activités de l'association while42, qui proposait un espace de sociabilité, de partage d'expérience et de formation à des ingénieurs informatiques et programmateurs d'origine française à San Francisco. Cette association comptait une cinquantaine de membres en 2015, année de sa fondation. Bien que d'accès limité (puisqu'il est nécessaire d'être un développeur pour y accéder), le suivi des activités de cette association à partir d'observations sur Internet (site de l'association, comptes Facebook de ses membres, blogs, comptes LinkedIn et Twitter) a permis de constituer une base de données sur quarante développeurs. On a réuni dans la base des informations sur leur formation, leur parcours professionnel et migratoire avant 2015, en 2015, date de la création de l'association, et depuis 2015 jusqu'en 2021. A été renseigné les compétences et projets réalisés par les enquêtés sur cette période. Dix entretiens complémentaires ont été réalisés avec des membres de ce groupe d'enquêtés. Le matériau mobilisé dans

cet article est tiré de ces trois sources : première phase d'entretiens, base de données constituée à partir du corpus d'enquêtés et entretiens avec des membres du corpus. Leur analyse a fait apparaître un constat en partie paradoxale : celui de la centralité du travail des développeurs dans la Silicon Valley et la fragilité de leur production.

Programmer : du drame aux renforts dans le cours de l'action

8 Dans cette première section, on s'intéressera à la manière dont les développeurs sont envisagés comme une catégorie de professionnels centrale dans la Silicon Valley ; une centralité qui va pourtant de pair avec l'incertitude qui pèse sur leur production. Comme évoqué en introduction, cette dernière s'avère particulièrement difficile à planifier car soumise à de constants aléas, que les développeurs vivent fréquemment comme des « drames » (Hughes, 1996). Ils font alors appel à des renforts (Becker, 1988) pour faire face aux difficultés rencontrées.

9 Au sein de la Silicon Valley, les développeurs assurent majoritairement des tâches de conception, souvent sous la forme d'applications et de nouvelles fonctions. Les développeurs sont ainsi souvent désignés par les termes de « *builders* », « *makers* », « *architects* », « *doers* » ou « *entrepreneurs* ». Ils sont reconnus et rémunérés sur le marché du travail en conséquence : le salaire moyen d'un *software developer* au premier trimestre 2021 à San Francisco s'élevait à 122 000 dollars par an, 92 000 pour un *data analyst*, 66 000 pour un *graphic designer* selon le site professionnel de référence Glassdoor. Cette valorisation des développeurs au sein de la Silicon Valley contraste avec le statut de leurs homologues dans d'autres pays ou foyers industriels. Un employé d'une grande entreprise de la région ayant travaillé à Paris considère ainsi :

> « En France, quand tu dis que tu fais de l'informatique, les gens te prennent un peu de haut. C'est vu comme… l'ingénieur informaticien. Les gens te voient comme le symbole de toutes les personnes qui n'ont pas réussi à les aider dans le service IT des boites où ils sont passées. On parle d'IT, parce que c'est une fonction support ; et pas là où la valeur se crée. Ici, c'est l'inverse : l'informatique est pas une fonction support mais là où la valeur se crée. C'est une vraie industrie informatique. Si je faisais du marketing, j'irais chez l'OREAL. Si je faisais du traitement des eaux, j'irais chez Veolia ; si je voulais faire du conseil, j'irais chez Accenture. Mais je fais de l'informatique, donc je veux être chez Google, Facebook, Apple, des boites dont le cœur, c'est l'informatique » [traduit par l'auteur]. (*Software developer II*, homme, 27 ans, grande entreprise)

10 Une comparaison des effectifs des différentes sous-catégories de travailleurs technologiques entre les régions de San Francisco et de New York illustre la densité et la concentration des rôles de conception dans la Silicon Valley. Alors que la région de New York constitue le second foyer des professionnels du numérique du pays, la répartition entre catégories de professionnels du code fait apparaître une domination des *developers*, *architects* et *scientists*, là où les métiers de maintenance et les fonctions supports de l'informatique (*administrators*, *analysts*, *supports*) prédominent dans la région de New York.

Tableau 2. Comparaison des professionnels du code entre la région de San Francisco et de New York (en gras, les valeurs >)

Sub-Groups	Bay Area	New York area

Application developers	**53,050**	39,970
Systems Software developers	**48,440**	11,140
Computer Network Architects	**6,200**	4,860
Research scientists	**2,900**	420
Use support	18,350	**21,760**
System administrators	14,760	**15,270**
Computer programmers	12,400	**15,880**
System analysts	11,410	**24,740**
Web developers	5,970	**7,140**
Network support	5,490	**6,130**
Database administrators	2,890	**4,780**
Security analysts	2,290	**4,400**

Source : Bureau of Labor Statistics (2015)

11 À partir des concepts forgés par Howard Becker au sujet des mondes de l'art, on peut ainsi qualifier le développement logiciel comme une activité « cardinale » au sein de la Silicon Valley. Cette centralité s'explique principalement par le « savoir-faire technique, les aptitudes sociales et le bagage intellectuel » (Becker, 1988 : 238) qui rendent les développeurs indispensables à la chaîne de production. Cependant, l'activité de conception de programmes informatiques s'avère particulièrement incertaine.

Le développement logiciel, une activité dramatique

12 Le travail de conception d'un logiciel se révèle complexe. Il s'agit de développer une interface de programmation, proposer et tester de nouvelles fonctions, constituer des bases de données, les structurer, opérer des extractions (Dagiral, Peerbaye, 2012), utiliser efficacement des interfaces systèmes pour appareiller les fichiers et les programmes, procéder au débogage, mettre en place des systèmes de sécurité pour l'ensemble des services, s'assurer de la conformité du programme avec les standards en vigueur, prendre en charge les tâches de support technique, le tout en collaborant avec d'autres développeurs, des designers, des spécialistes des données, des *product managers*, des commerciaux, des entrepreneurs, des investisseurs et des usagers. Les développeurs de la Silicon Valley doivent composer, parfois quotidiennement, avec cette multiplicité d'intervenants, tout en veillant à la cohérence de l'assemblage de leur logiciel.

13 La programmation, comme le note B. Rieder, ne tolère pas les approximations (Rieder, 2006 : 243). Or, la complexité des entités en présence tend à augmenter le nombre d'erreurs potentielles : erreurs de calcul, mauvais alignement des lignes et de valeurs, conflits entre différentes versions et programmes, problèmes d'extractions, mauvaise structuration de la base, non mise en conformité avec les standards récemment actualisés par telle ou telle organisation de référence, etc. Ces problèmes, parfois rétroactifs, exposent les développeurs à d'incessants coups d'arrêts, qui peuvent se transformer en impasses. Pour ces raisons, le travail de programmation peut être

jugé fastidieux (Button, Sharrock, 1995). Les enquêtés décrivent leur activité comme temporellement coûteuse, intellectuellement ardue et émotionnellement éprouvante. N. Auray évoquait à ce sujet l'« insatisfaction face aux flux d'informations prescrits » et la frustration née de « l'intention de résoudre une énigme » (Auray, 2012 : 34), tout particulièrement quand le programme vient à « planter » sans raison apparente ou qu'un problème reste sans réponse.

14 Le quotidien des développeurs est hanté par l'échec. Échec technique en premier lieu. Une analyse des forums de support aux développeurs de Facebook et Google Map révélait ainsi que le niveau de résolution ne dépasse pas respectivement 10 et 15%[6]. Mais la résolution technique ne signifie pas la fin des problèmes pour les développeurs. Dans la Silicon Valley, l'abondance des solutions et la concurrence entre programmes rend le destin des logiciels très aléatoire, y compris lorsque les développeurs opèrent à partir d'une place forte de la région. Un ancien employé de Facebook avance ainsi que 90% des nouvelles fonctions et produits développés par l'entreprise sont abandonnés faute d'utilisateurs (Martinez, 2016). À ce verdict commercial s'ajoute le caractère éphémère des cycles de vie des entreprises dans la région, dont plus de 90% ne dépassent pas les cinq ans d'activités. Incertitude au niveau du logiciel, incertitude des usages, incertitude sur la durée de vie des entreprises, on caractérisera pour ces raisons l'activité des développeurs par une loi d'impuissance : celle d'un un ratio échecs-réussites de l'ordre de 90/10, soit pour 10% de réussites (des projets, des produits et des entreprises) 90% d'échecs. Ce ratio s'avère particulièrement frustrant et stressant pour les développeurs.

> « J'ai commencé tout seul, je restais à mon appartement, c'est super dur comme mode de vie, je veux plus le refaire, j'ai trouvé ça vraiment déprimant… Tu passes tes journées, tes week-end, sans retour de personne, à te demander si ça va marcher… Et ça plante, ou tu suis différentes pistes, qui vont nulle part parce que quelqu'un l'a déjà fait, tu galères… Tu es là à batailler sur ton app, sans argent, sans sortie. Tu te demandes ce que tu vas faire, si tu vas y arriver, à quoi ça sert… En plus, San Francisco coute cher, beaucoup de gens ici font de l'argent beaucoup plus facilement… » [traduit par l'auteur]. (Ancien développeur-entrepreneur, homme, 34 ans, devenu investisseur)

15 Ces aléas sont susceptibles d'être vécus par les développeurs comme des « drames », au sens donné à ce mot par E. Hughes. Le sociologue soulignait que les drames au travail avaient pour origine plusieurs facteurs : changement technologique, évolutions organisationnelles, différences de temporalités des parties prenantes, dissymétrie de ressources et différences d'objectifs (Hughes, 1996). Selon l'auteur de *The Sociological Eye* : « une partie du drame réside dans le fait que ce qui est travail quotidien et répétitif pour l'un est urgence pour l'autre. » L'extrait d'entretien suivant illustre la manière dont les développeurs se retrouvent au cœur d'un rapport asymétrique au temps, les différentes entités et parties prenantes (ici un grand groupe de médias, des utilisateurs, des sponsors, et une équipe de développeurs) poursuivant des objectifs et des séquences de temps n'étant pas alignés.

> « Avec la télévision, pendant la pub, les gens font autre chose. Les annonceurs payent, mais il y a un gros problème d'attention, les études montrent que 80% des gens font autre chose. On a développé une solution pour que les gens restent devant leur écran. Et Turner [ndlr : TBS, filiale de Warner media] nous donne notre chance, direct pour les *play-offs* de baseball avec 20 millions de téléspectateurs, une des plus grosses audiences de l'année. Le contrat, c'est 800K, 21 matchs, sur 3 semaines. Le problème, c'est qu'on avait que l'IOS. Donc on fait des API spéciale pour Turner, et on sous-traite pour faire la version Android, par des colombiens, des super gars, font ce qu'ils peuvent mais tout était buggé, c'était la catastrophe. Pendant, avant et après les matchs, des spots passent pour

> télécharger l'app ; pendant le match, le présentateur devait dire aux gens de jouer avec l'app, avec des retours en temps réel, et à la fin des matchs, il devait y avoir le top 5 des joueurs de notre jeu. On était fin 2011 et Insta venait de se faire racheter pour 2 ou 3 milliards, on se disait : *ça y est, on est train de prendre l'ascenseur là. C'est génial !!* Tu dors peu, t'es à fond, tu touches plus le sol… Et là, catastrophe : tout s'écroule. Parce qu'une fois passé les 20 000 utilisateurs en simultané, l'architecture n'était pas faite pour supporter le truc. Donc là, nuits blanches totales, on fait venir deux mecs, spécialistes dans le *scaling* qu'on a embauché par la suite. Les mecs prennent des jours de congés, on les paye super cher en mode consulting, hors de prix, trente mille sur l'opération, c'est un forfait ; ils ont fait un rapide audit *: non, mais attendait les mecs, c'est pas du tout les bonnes techno, les technos que vous avez sont pas extensibles, vous avez une base de données relationnelle MySQL et PostGreSQL, ça tient pas la charge…* C'était tellement le bordel qu'ils ont dû passer par un autre mec, qui avait aidé Instagram à *scaler*. Turner est venu nous monitorer, ils ont dû prévenir Samsung, qu'ils avaient fait rentrer comme partenaire dans le truc, avec à chaque match des Samsung galaxie à gagner, avec des annonces à la télé… Donc toute cette expérience, ça a été à la fois génial et chaotique. C'était vraiment les montagnes russes » [traduit par l'auteur]. (CTO, homme, 46 ans, d'une startup, série A)

16 Mais les parties prenantes qui se démultiplient au fur et à mesure du processus, comme décrit cet extrait d'entretien, sont également susceptibles de se transformer en ressource. C'est le cas avec le recours aux renforts.

Recourir à des renforts

17 En effet, pour sortir des impasses qu'ils rencontrent, les développeurs font appel à des « renforts », que Howard Becker définissait comme « toutes les personnes impliquées dans la production de l'œuvre et qui apportent une aide directe ou indirecte à l'artiste qui lui, exerce l'activité cardinale (Becker, 1988 : 27). De ce point de vue, on peut distinguer trois types de renfort : des renforts directs et personnels (collègues, connaissances et pairs), des renforts intermédiaires et contractuels (embauchés pour une mission) et des renforts indirects et collectifs (usagers).

18 Les développeurs recourent incessamment à l'aide et l'expertise de leurs collègues et connaissances personnelles. Elles présentent l'avantage d'être gratuites, rapidement mobilisables et qualifiées. Les entreprises de la région adaptent leur organisation pour faciliter ce type de soutien, en veillant à l'existence et l'accessibilité à des salles de travail collectif, la présence de tableaux, la possibilité d'écrire directement sur les murs avec des craies ou de stylos laissés à disposition, le développement de forums et systèmes de communication internes ou encore en mettant en place des équipes dédiées à des interventions d'urgence (souvent nommées « *special squads* »). Un développeur souligne tout le prix de ces renforts internes :

> « En tant qu'ingé, en termes de savoir-faire, j'ai appris pas mal des gens qui m'entourait, pas le cas dans d'autres pays où j'ai travaillé. C'est hyper agréable quand tu pétes les plombs sur un problème de pouvoir te lever et aller voir une énorme brute. Chez [x], ils ont un truc hyper prétentieux, une équipe qu'ils appelaient la *tiger squad*, quatre ingénieurs, qui n'ont pas d'affectation précise et qui sont chargés d'aider des projets qui se retrouvent bloqués ; il y a toujours des projets qui se retrouvent dans une impasse ou sur un chemin critique, que les mecs sont pas bons, qu'il y a eu des conneries, etc. Dans ce cas-là, on envoie ces gars, qui sont des énormes brutes. Ils sont pas compétents sur tout, mais ont une assise énorme. Et quand je me retrouvais avec mon équipe à péter les plombs, et c'est arrivé plusieurs fois, ben j'allais les voir ; ils se mettaient à côté de moi, boum, boum, me montraient des trucs et là tu dis : *ah ouais…* Et tu retournes bosser. C'est un type de construction de compétences que j'ai jamais vu ailleurs que dans

> la Silicon Valley. Quand tu bosses ailleurs, tu lis des articles… Ici, tu te retrouves à bosser avec les mecs qui ont écrit les articles et qui t'expliquent les trucs directement ; en termes de *hard skills*, c'est super » [traduit par l'auteur]. (*Software developer senior II*, homme, 27 ans, entreprise cotée en bourse)

19 En plus de ce soutien intra-entreprise, les développeurs recourent fréquemment à leur réseau personnel, à des amis ou des connaissances rencontrées dans la région, suivant la logique de *potlach* ou troc d'informations (Ferrary, 2001).

20 Au-delà de ces aides directes, les développeurs recourent également à des renforts externes. Ils embauchent notamment des conseillers, consultants ou exécutants sur de courtes missions. Le coût élevé de ces prestations incite les entreprises à externaliser les fonctions supports dès que leurs capacités financières et logistiques le permettent. Un manageur d'une entreprise de plus de 200 employés explicitent ainsi les enjeux de cette forme hybride de renfort entre sous-traitance et « petites mains » de l'informatique (Denis, 2018) :

> « Pour toute la partie support client, l'enjeu est d'être dispo 24/24, 7/7, partout dans le monde, donc on a des gens au Mexique et des gens en Inde. Ils ont des shifts horaires, pas par zonage géographique, l'enjeu étant : te répondre rapidement quel que soit l'origine de la demande. La délocalisation, je l'ai vécue à Google, avec l'Inde ça marchait très mal : décalage horaire, culture, langue, pas de middle management, donc besoin d'un management direct, soit une galère sans nom. Une qualité du travail pas si bonne donc au final, pas rentable. Donc on a oublié, par contre le Mexique : extraordinaire, on a une équipe de cinquante personnes, qui me fait un peu pensé au Google des débuts : super bien formés, dans des facs US, des gens jeunes, ils ont faim, ils sont dans la même zone horaire que nous, sont tous très intelligents, motivés, un vrai régal et on les paye moitié moins cher qu'ici. Ça serait 100 pour un mec dans la vallée, 50 pour un mexicain, 17 pour un indien. On a une structure au Mexique qui se développe bien. Ils font que de la technique : dev, QA, support. Une société intermédiaire s'occupe des recrutements. Et nous on leur donne un guide : faut connaître java quoi, ce qu'ils apprennent à l'école. Et en plus, ils restent et sont fidèles, pas comme ici » [traduit par l'auteur]. (*Software Manager*, homme, 52 ans, dans une startup, série C)

21 Enfin, les développeurs recourent abondamment à la documentation laissée en libre accès et les ressources en ligne offertes via des forums dédiés. Un enquêté évoque leur importance comme suit :

> « Le truc fascinant dans la programmation, c'est que tu n'as pas forcément besoin d'un haut niveau de formation, et franchement, je n'ai pas appris grand-chose à l'université qui m'a aidé en tant que développeur ; ce qui a compté, c'est de pouvoir apprendre en ligne, de lire en ligne, d'utiliser des outils qui sont en ligne. C'est librement accessible. Quand je construisais des applications, je me demandais : quelles sont les limitations de tel ou tel environnement ? La documentation est là. Si tu construits une app pour un produit Apple, Apple met toute la doc sur Internet pour t'aider. C'est chiant à lire, mais c'est là. Tu vas le chercher et ensuite tu construis » [traduit par l'auteur]. (Entretien avec un développeur-entrepreneur, homme, 32 ans, devenu employé d'une entreprise cotée en bourse)

22 Ces trois types de supports (directs et personnels, intermédiaires et contractuels, indirects et collectifs) représentant une forme de *continuum* pour les développeurs. I. Siles a notamment montré au sujet de Twitter comment les processus d'itération au sein d'un réseau extensible de développeurs, interne puis externe à l'entreprise, incluant progressivement les usagers, ont accompagné la croissance de l'entreprise (Siles, 2013). Ces échanges sont l'occasion d'une double opération : la collecte d'informations sur les options techniques et stratégiques qui pourront être suivies dans un environnement donné ; la formation d'alliances avec des entités diverses (organisations, technologies et

professionnels) qui pourront éprouver, évaluer et valider les choix opérés. Cette double opération fait entrevoir l'épaisseur sociale de l'activité de programmation au-delà des discours sur la solitude des développeurs, de l'aridité des observations et du cliché du programmeur courbé sur son écran d'ordinateur. Les développeurs mobilisent en effet constamment des ressources sociales pour solutionner les problèmes qui entravent quotidiennement l'avancement tant de leurs sessions de programmation que de leurs projets.

Se projeter : les médiations à l'échelle du projet

23 Après avoir évoqué le cours de l'action dans les phases de programmation, et la manière dont les développeurs font appel à des renforts, cette section aborde la façon dont les concepteurs de logiciels mettent en place des stratégies à l'échelle du projet. Après avoir opéré une distinction entre projet et projection, on évoquera la manière dont les développeurs s'orientent dans l'activité au travers d'outils et de problèmes spécifiques.

Le développement logiciel, un travail de projection

24 L'organisation par projets a été identifié comme un mode préférentiel d'organisation du travail dans les univers artistiques (Menger, 2002) et plus largement au sein des économies capitalistes depuis les années 1970, le terme de « projet » étant le plus utilisé dans les manuels de management étudiés par Luc Boltanski et Eve Chiapello (Boltanski, Chiapello, 1999). Toutefois, les développeurs de la Silicon Valley ne considèrent pas le projet comme un certain type d'organisation du travail, parmi d'autres possibilités, mais comme un implicite de leur activité. Pour le comprendre, on peut se référer à ce qui constitue l'un des premiers classiques de la littérature sur le management dans le domaine de l'édition de logiciels.

25 Frederick Brooks (Brooks, 1996), ingénieur en charge du projet OS/360[7] publia en 1975 un livre présentant les leçons de sa difficile expérience en tant que chef d'équipe chez IBM (« *project manager* »). Dans *The Mythical Man-Month*, l'auteur présente la conception logicielle comme une suite presque anarchique d'édition de versions : à partir d'une version prototype ou beta, les ingénieurs informatiques développent des fonctions absentes des versions antérieures sur la base d'aller-retours (« *feedbacks* » et « itérations ») entre ingénieurs, puis avec des utilisateurs, sans que les propriétés de la version finale ne puisse être anticipées. À partir d'un problème, les membres de l'équipe travaillent à élaborer des solutions, sous la forme de schémas et de scénarios, qui fixent des points d'arrivée approximatifs, rarement, voire jamais atteints, avec pour objectif de produire une version stabilisée. F. Brooks souligne que ce mode opératoire conduit à la multiplication des pistes et des orientations, au point de menacer la continuité (ou « unité ») conceptuelle du logiciel. L'addition de travailleurs au sein de l'équipe ne peut jouer comme une ressource, puisque chaque nouveau travailleur est susceptible d'ouvrir de nouvelles directions de travail, de manière non-coordonnée, éloignant un peu plus le collectif de l'édition d'une version cohérente. Dès lors, « ajouter des personnes à un projet en retard accroît son retard » (*Brooks law*). La production en séries, les organisations hiérarchiques, une division du travail poussée se révèlent dans cette

perspective inappropriée.

26 Cette série de constats conduit à renverser le modèle d'organisation du travail associés aux grandes entreprises aux Etats-Unis. L'historien A. Chandler (1977) a mis en lumière la manière dont la centralisation des informations et des prises de décision par des *top* et *head managers*, situés au sommet de divisions distinctes, constituait un avantage comparatif pour les entreprises émergentes au 19e siècle dans le domaine des transports, de l'énergie et des communications. Les manageurs ont ainsi prévalu en tant qu'intermédiaires autonomisés entre des producteurs-offreurs et des clients-demandeurs. Ils incarnent et concentrent le pouvoir pour des raisons fonctionnelles, car ils permettent de faire circuler de manière cohérente, hiérarchique et verticale, l'information et les prises de décisions au sein des entreprises.

27 Or, F. Brooks propose un modèle inverse. Selon lui, pour laisser libre court au processus d'itération nécessaire à la production logiciel, le travail doit être coordonné de manière horizontale, pour favoriser l'accompagnement dans la continuité du développement logiciel. Il plaide pour cette raison en faveur de petites équipes réunies autour d'un architecte, qu'il compare à un « chirurgien en chef » placé, non plus en amont ou en surplomb comme dans les grandes entreprises analysées par A. Chandler, mais au cœur de l'action. Ce modèle, qui repose sur l'équilibre entre itérations sur différentes versions et intégrité du logiciel, met en lumière une dimension souvent apparue dans les entretiens : le caractère projectif du travail de développement. Les développeurs se projettent dans un travail sans en connaître l'issue, le temps nécessaire ou les propriétés finales du processus de production[8]. La conception logicielle procède ainsi nécessairement sur le mode du projet, mais un projet sans point d'horizon clairement identifiable. Si les développeurs peuvent s'appuyer sur des scénarios (Jaton, 2021 : 192), ces derniers ne permettent pas une coordination durable et stable à la différence des scripts dans le cinéma (Rot, 2014) ou des partitions dans le domaine de la musique (Becker, Faulkner, 2008). Pour cette raison, le quotidien des développeurs consiste à se projeter, plus qu'à suivre les étapes prévues dans le cadre d'un projet. Or s'ils ne peuvent s'appuyer sur des supports de continuité en aval, ils mobilisent une série d'outils en amont du processus de production.

S'orienter via les médiations

28 Les développeurs de la Silicon Valley répètent fréquemment leur attachement au référentiel de l'innovation. Pourtant, ils n'ont de cesse de recourir à des outils préexistants. Ils usent ainsi de langages de programmation, de cadres de développement (« *framework* », « infrastructure de développement », « environnement de développement » ou « socle d'applications »), des éditeurs de texte, des collections de routines compilées prêtes à être utilisées (bibliothèques), des systèmes de gestion de versions, de gestion de base de données, des systèmes d'exploitation et d'intégration, des systèmes de serveurs et de mémoire cache, ainsi que des systèmes de surveillance. Ils recourent également à des outils de mesures de performances ; des systèmes de réseau de diffusion de contenus, des gestionnaires de paquets destinés à automatiser les processus d'installation-désinstallation et de mise à jour des logiciels. Ils doivent en outre savoir utiliser des systèmes de suivi des bugs. Cette multiplicité d'entités explique les succès chez les développeurs des SDK (*Software development Kit*) regroupant une partie de ces outils à l'initiative des grandes entreprises (Pybus, Coté, 2021). Enfin, pour organiser et coordonner le travail, ils se réfèrent aux méthodes dites agiles (Boboc, Metzger, 2020).

Si les développeurs ont une connaissance hétérogène et souvent limitée de l'histoire

de ces médiations, il est pour eux impératif d'établir et d'alimenter un répertoire afin de s'orienter efficacement au cours de leurs projets. La culture scolaire joue de ce point de vue un rôle important puisque ces outils sont développés, désignés et organisés dans la continuité des technologies scripturaires (Goody, 1979). Les développeurs utilisent des « *library* », « *chapters* », « *scripts* », « *repositories* », « traducteurs » et « interpréteurs » (pour traduire et établir les liens d'un programme; l'interpréteur exécutant le programme), des « éditeurs de liens » (ou « lieur », qui lie ensemble les différents modules et les bibliothèques de routines en un seul fichier exécutable), tandis que les notions de « lignes », de « syntaxe », d'« archives », de « code », de « prédicats » et de « paradigmes » (impératif, déclaratif -logique et fonctionnel-, orienté objet, sémantique, textuelle, etc.) rendent possible l'organisation des informations suivant une architecture modulaire (Rieder, 2019). Ce constat ne vise pas à nourrir la discussion serrée sur l'ontologie littéraire du code informatique (Rieder, 2006 ; Méles, 2016 ; Jaton, 2021). Cependant elle appelle deux observations sur la sociologie des développeurs.

30 D'une part, les développeurs ne s'inscrivent pas dans un univers *sui generis*, mais évoluent dans un environnement peuplé d'outils. On désignera ces outils par le terme de « médiations ». A. Hennion a souligné en quoi l'acte musical suppose des « moyens qui vont des dispositifs physiques et locaux, tels le cadre, le socle, l'éclairage, le lieu même d'exposition, aux médiations institutionnelles, comme le catalogue ou le musée, aux cadres généraux de l'appréciation collective, que sont le discours des critiques et l'inscription sélective de certains objets dans l'histoire de l'art, et jusqu'à l'existence même d'un domaine autonomisé appelé art : galeries, marchands, écoles de peinture, styles, grammaire, système des goûts, etc. » (Hennion, 2015). Il en va de même dans la conception logicielle, au travers du recours à des langages de programmation, des systèmes d'exploitation, et autres outils préexistants mobilisés au cours du processus de production.

31 D'autre part, la compréhension et la mobilisation de ces médiations par les développeurs sont rendues possibles par une formation préalable. Là où les « héritiers » décrits par P. Bourdieu et J.-C. Passeron valorisent des capitaux accumulés en dehors de l'école pour asseoir leur réussite sociale et professionnelle (Bourdieu, Passeron, 1964), les développeurs prennent à l'inverse appui sur une culture scolaire acquise antérieurement ou parallèlement. Bien qu'ils décrivent régulièrement des enseignements scolaires peu utiles, datés ou décorrélés des réalités d'une industrie particulièrement agile et dynamique, ils s'adossent sur des savoirs disciplinaires (grammaire, linguistique, statistique, etc.) et des compétences dont N. Auray a souligné ce qu'elles devaient à la pensée et au milieu scolaire (Auray, 2012).

Tableau 3. Les compétences non-techniques des développeurs

Compétences	Descriptions
Intrapersonnelles	Admettre être critiqué ; accepter de se corriger ; prendre sur soi ; art de pondérer la parole dans une communauté d'échange ; non recours à l'accusation personnelle ; à la violence ; à l'insulte ; savoir marquer un désaccord avec un supérieur
Interpersonnelles	Ouverture à l'altérité ; recherche de l'hétérogénéité des rencontres ; varier son univers relationnel ; être capable de trouver des ressources dans autrui ; construire des ponts ; ne pas intérioriser l'entre soi
Herméneutiques	Capacité à s'insérer dans un jeu de langage ; à déchiffrer correctement les sources documentaires; à analyser les jeux de pouvoir et les intentions cachées

Topologiques	Savoir explorer et adopter un raisonnement logique ; « Promenade » et « géométrie euclidienne » ; tension entre ces deux lignes ; talent en informatique ; savoir les dialectiser.

Source : Auray (2012)

32 L'organisation du travail par projection conduit ainsi les développeurs à s'appuyer sur des médiations et des compétences non techniques, disciplinaires et sociales, pour optimiser leur orientation. De ce point de vue, les problèmes représentent une dimension ambivalente, puisqu'ils constituent pour eux des impasses à éviter, mais aussi des repères à partir desquels s'orienter.

S'orienter par les problèmes

33 En effet, les développeurs au sein de la Silicon Valley perçoivent les problèmes comme des repères permettant de s'orienter.

> « Je voulais faire des applications pour portables, c'était le début des années 2010, mais je trouvais pas d'idée, ce n'était que des dérivatifs de 4chan, foursquare, etc. C'était difficile de trouver quelque chose de vraiment original. Je voulais m'investir sur quelque chose de nouveau, mais je passais mon temps à réinventer la roue… Je construisais quelque chose, je passais d'app en app… et puis finalement, je suis tombé sur quelque chose qui me posait problème, pour lequel je trouvais pas de solution ailleurs, et j'ai décidé d'essayer de le résoudre… » [traduit par l'auteur]. (Entretien avec un développeur-entrepreneur, homme, 34 ans, entreprise, série C)

34 Les problèmes constituent ainsi des impasses dans un cas, des points d'appui dans d'autres. En s'appuyant sur le lexique des mathématiques, on en distinguera deux types : d'une part, les « problèmes basiques » que les développeurs tentent de solutionner à travers des moyens existants ou connus de leurs pairs ; d'autre part, des « problèmes atypiques ou ouverts » qui appellent la formulation de solutions inédites. Dans la Silicon Valley, les développeurs valorisent particulièrement le second type de problèmes, n'ayant jusqu'alors pas été formulé et solutionner, notamment parce qu'ils représentent pour eux une garantie en termes d'apprentissages. Les développeurs envisagent en effet les problèmes à partir des rétributions possibles, des rétributions qui peuvent être sociales, matérielles, ou encore cognitives. Ils tendent d'ailleurs à anticiper les compétences qui seront acquises au cours du processus de résolution. Les développeurs, ne pouvant prévoir l'évolution du secteur, procèdent néanmoins à des anticipations probabilistes.

35 Ainsi donc, pour réduire les coûts temporels, ils s'appuient sur des outils, des référents, des protocoles, que l'on a regroupé sous le terme de médiations, sur lesquelles ils s'appuient. Cette section a également permis d'identifier l'ambivalence des problèmes, à la fois impasse et repère guidant leur orientation. Ce dernier enjeu renvoie à la manière dont les développeurs gèrent leurs orientations professionnelles.

Se réorienter : opérer des tournants au fil de la carrière

36 Dans cette section, on s'intéressera à la manière dont les développeurs tentent d'orienter leur parcours sur le long terme en initiant des tournants. Dans les entretiens, les enquêtés mettent en avant l'importance d'événements qui ont « réorienté leur

route » (Elder, 1998) : l'achat d'un ordinateur au sein de la famille, l'initiation au code par un tiers (ami, frère, oncle, cousin, voisin, etc.), une rencontre à l'université, un premier travail décroché dans la Silicon Valley, la réussite de tel ou tel projet, celle d'une entreprise où ils ont travaillé, etc. E. C. Hughes soulignait dès les années 1950 l'importance sociale des ressorts temporels de la vie professionnelle (Hughes, 1971). Les « *social life-cycles* » ouvrent et referment des « *cycle of work* », marqués par le passage par les écoles (*high school*, *prep school*, *universities*, etc.), dont les rites ne préservent pas totalement les individus de l'arbitraire de l'histoire (guerre, crise, creux démographique, etc.).

37 Or, les développeurs présentent ces croisées des chemins de manière rétrospective, et donc sont susceptibles de relever d'une forme d'illusion biographique. Les travaux d'A. Abbott (2001) permettent en partie de lever cette ambiguïté : « un *turning point,* et non une banale ondulation, suppose qu'un temps suffisamment long se soit écoulé dans la nouvelle orientation au point qu'il soit devenu clair que la direction a véritablement changé » (Abbott, 2001). Toutefois, cette conception suppose qu'une mesure du temps, une quantification des variations de parcours, voire d'éventuels points de comparaison soient possibles, dans la perspective d'une analyse historique et temporelle. Or, l'enquête a fait apparaître un autre type de tournant. En effet, les développeurs étudiés cherchent à produire, maîtriser et anticiper, de manière volontariste, des tournants. Il ne s'agit pas de tournants dont le point d'origine se situe dans le passé, évoqués de manière rétrospective, mais de tournants projetés, souhaités, anticipés, dont le point de départ se situe dans un futur proche. Pour cette raison, on qualifiera ces tournants d'intentionnels. Cette section s'intéresse à trois types de tournants intentionnels : la recherche de certifications, la création de médiations, et la fondation d'entreprises.

Obtenir des certifications

38 Comme déjà évoqué, les développeurs adoptent souvent un discours critique à l'égard des formations scolaires, vantant à l'inverse les chemins de traverse qui leur ont permis de s'initier à la programmation. Les enquêtés, et plus généralement les développeurs de la Silicon Valley, s'avèrent pourtant fortement diplômés et spécialisés. Dans le corpus, 95% ont un niveau master, 92% a réalisé sa formation dans le domaine de l'informatique. Leur formation initiale se trouve néanmoins souvent éloignée des centres de formation prestigieux et historiques que sont le MIT, Stanford, Caltech ou Carnegie-Mellon (12,5% sont diplômés d'un IUT français situé en dehors de la région parisienne). Une fois implantés dans la Silicon Valley, les développeurs du corpus, et plus généralement au sein de la région, suivent des formations certifiées, parfois onéreuses (près de 3 000 dollars pour un « *full certificate* » à l'université de Stanford, près de 45 000 dollars pour une année de formation). Au sein du corpus d'enquêtés, 45% ont suivi une formation certifiée aux Etats-Unis alors même qu'ils avaient déjà bénéficié d'une formation dans un autre pays.

39 Pour obtenir ces certifications, les développeurs de la région ne se tournent pas exclusivement vers des universités : des grandes entreprises (28% affirmaient en 2021 sur leur profil LinkedIn avoir obtenus une certification délivrée par Microsoft, Amazon ou Google), écoles du code de la Silicon Valley (plus d'une douzaine), des incubateurs ou des accélérateurs sont également susceptibles de leur offrir des certifications, à même selon eux de faire évoluer positivement leur trajectoire. Les motivations invoquées par les enquêtés sont diverses : obtention d'un visa, constitution d'un réseau, positionnement sur un segment prometteur du marché du travail. L'absence de standard international et d'étalon entre les différents systèmes nationaux de formations

expliquent en partie les démarches consistant à convertir la valeur d'un titre scolaire obtenu à l'étranger sur le marché du travail de la Silicon Valley.

> « On a une définition très différente de l'ingénieur en Europe et aux US. Aux Etats-Unis, il y a des départements de *Computer Science*, qui sortent des mecs qui vont se spécialiser dans l'informatique, des mecs qui vont savoir faire de l'algo et du *deep learning*, c'est ce qui se rapprochent le plus de ce qu'est un ingénieur à la française. Mais les américains ont peu d'écoles avec un cursus uniformisé, comme en France, où tu as la CPI, avec certaines règles, cours, fondamentaux. En plus, aux Etats-Unis, tu as des types, de plus en plus, qui font trois mois de formation en ligne et qui se vendent comme *software engineer*. Ça explose tellement vite ici que tout le monde peut le devenir ; sans règle, avec plein de façons de l'être. » (*Software Manager*, homme, 41 ans, startup, série C)

40 Les développeurs justifient en effet le paradoxe entre leur critique sévère de l'offre de formation scolaire et académique avec le fait d'y recourir abondamment par leur volonté de mieux se positionner sur le marché du travail. Des enquêtés parlent ainsi de « réorganiser son CV » ou encore d'obtenir un « tampon » afin de rendre leur profil plus attractif.

> « J'ai appris à coder tôt, mais j'ai des trous dans ma formation, donc je voulais combler ça, et c'est pour ça que je voulais être codeur et pas *product manager*. Je voulais aller en profondeur. Donc j'ai lu beaucoup de bouquins, je me suis spécialisé en *backend*, parce que c'est soi-disant mieux vu ici, et que les projets sont plus intéressants, genre pas faire de l'iPhone mais de l'infrastructure des données. J'ai fait beaucoup de Mooc, une dizaine en dix ans. Et là je candidate pour une formation à quatre mille dollars à Stanford. Aujourd'hui, je me demande si je vais pas continuer mes études. J'ai envie de réorganiser mon CV. Parce qu'ici y'a deux façons de te faire connaître quand t'es *software engineer* : la boite dans laquelle tu travailles, et moi c'est pas Uber, ça bosse, pas mal, ça marche, on est leader mondial, mais c'est une niche. Donc y a pas le tampon. Et ensuite le diplôme que tu peux présenter. Stanford, Berkeley, etc. » (Ex-*Software developer Senior I* dans une startup, série C, devenu entrepreneur)

41 En l'absence d'un cadre réglementaire, d'un contrôle à l'entrée ou de syndicats donnant une assise aux professions (Abbott, 1988), ce type de certifications opère comme un principe d'orientation sur le marché du travail.

Créer des médiations

42 Une seconde façon d'initier un tournant dans leur carrière pour les développeurs consiste à produire des outils qui deviendront une référence dans leur domaine. L'histoire de l'informatique fournit quelques exemples de concepteurs devenus des références, notamment mis à l'honneur au musée de l'informatique de Mountain View ou dans le département de Computer Science à Stanford (accueilli sur le campus au sein du bâtiment Gates, financé par le fondateur de Microsoft), à travers des objets, des portraits et des légendes l'importance de leurs contributions. Ces mises en scène de la mémoire leur confèrent une reconnaissance certes durable mais qui s'avère suivant les termes d'en enquêté relativement rares.

> « En informatique, ça dépend des branches… Il y a des choses en commun, puis des branches, par communautés, dans la communauté Linux, Bill Gates n'a sûrement pas sa place et Steve jobs, qui n'a rien inventé, est un pignouf. Les gens qui le vénèrent sont plus coté marketing ou entrepreneuriat. Sinon dans le commun, il y a Turing, Knuth… c'est le top. Les pointures, c'est pas des milliers de personnes non plus, je dirais que ça représente entre 30 et 50 personnes… Et peut-

> être une dizaine qui sont des dieux. Des mecs qui ont fait l'informatique moderne. Il y a un côté un peu divin, mystérieux, parce que tu comprends pas d'où ça sort, donc il y a une admiration béate. Mais après, c'est une admiration limitée au domaine de l'informatique. Si tu vois Knuth bouffer au Mcdo ou faire la promo d'une marque, on s'en bat les couilles. » (Entretien avec un développeur indépendant, homme, 32 ans)

43 S'il est rare pour les développeurs d'atteindre une telle position, il est toutefois courant qu'ils cherchent des effets de reconnaissance par le biais technologique. À la manière de M. Callon, on pourrait adopter une vision stratégique et considérer que cette manière de procéder vise à « se placer entre (*inter-esse*), s'interposer » (Callon, 1986) pour consolider sa position sociale.

44 Cette reconnaissance suppose toutefois une grande qualité et maîtrise technique. La virtuosité est en effet souvent présentée par les développeurs et dans la littérature dédiée comme un principe de différenciation. Les développeurs envisagent les réalisations comme un élément de comparaison et d'objectivation de leur valeur. F. Brooks soulignait déjà dans les années 1970 qu'une application était trois plus facile à créer qu'un compilateur, lui-même trois plus facile à développer qu'un système d'exploitation, tandis que la productivité des meilleurs ingénieurs informatiques serait cinq à dix fois supérieure à celle des pires d'entre eux (Brooks, 1996). Le type de production, la rapidité à la développer, sa qualité constituent des indicateurs qui permettent aux développeurs d'identifier des hiérarchies, de se repérer et de s'orienter par rapport à elles.

45 Or, au sein de la Silicon Valley, beaucoup de développeurs produisent des applications, des services de plateformes, des API, pour alimenter un environnement technologique construit à partir (« *built on the top of* ») de couches ou de médiations plus anciennes ou plus profondes. Dans le groupe d'enquêtés, seul deux individus avaient participé à la production d'un système d'exploitation (Androïd pour le premier) et à un langage de programmation (Swift pour le second), en tant que membre d'une équipe comprenant des dizaines d'opérateurs, réunis autour de *project managers*, eux-mêmes guidés par des « architectes » : en l'occurrence Andreï Rubin pour Androïd, et Chris Lattner qui assura la continuité du projet Swift, intégré par Apple.

Tableau 4. Types de produits dont les enquêtés ont été en charge de la production entre 2015 et 2021

Api	62,6
Platform services	47,5
Apps	46,1
Cloud services	25
Security services	22,5
SDK	5
Langage	2,5

Source : entretiens LinkedIn, Medium

46 Si la création d'un outil capable de changer les manières de faire d'une large communauté d'usagers s'avère peu fréquent, beaucoup se montrent à l'affût d'opportunités (White, 1970) ouvertes par l'arrivée d'une technologie ou d'un service structurant.

> « Au départ, la vision c'était de travailler sur l'idée de génération. J'avais vu les différentes générations, depuis l'époque où j'étais chez Oracle : j'ai travaillé sur des *mainframe computers*, du web, des app, du mobile… Quand les apps sont apparues, c'était clair que ça serait l'équivalent du navigateur pour le web. Le canal par lequel les choses seront rendues accessibles. Les apps allaient devenir le vaisseau, le conteneur [ndlr : *container*] pour donner accès de nouvelles fonctionnalités de service aux gens. Et quand j'ai vu ça, je me suis demandé : comment ou quelle part du logiciel entreprise va aller vers le mobile ? Ma conclusion, c'était que le CRM allait être le meilleur endroit pour innover. Parce que c'est là où tu peux valoriser des compétences *Business to Consumer*. Beaucoup de clients viennent du monde de l'entreprise, pour demander de l'information, de l'aide, des contacts… Si tu demandes comment les entreprises vont toutes massivement demain fournir des fonctionnalités aux utilisateurs, c'est à travers les apps. Pour l'instant, pour accéder à ces entreprises, ça reste le téléphone et les courriels, des technologies qui ne sont pas de la même génération que l'app. Le mode de communication qui correspond vraiment à l'app, c'est la messagerie. SMS, WhatsApp, peu importe, c'est le nouveau mode de communication et on est parti de cette idée en disant : *ok les clients veulent communiquer avec les entreprises*, *nous on va construire un service qui le permet* » [traduit par l'auteur]. (Ex-*software developer* chez Oracle, devenu entrepreneur (serie C))

47 Ce type de recherche pose pour les développeurs la question du contrôle du temps investi et de la valeur générée dès lors qu'ils ambitionnent une percée technologique. La création d'entreprise apparaît de ce point de vue comme une solution privilégiée par une majorité de développeurs au sein de la Silicon Valley.

Fonder son entreprise

48 Célébré dans les années 1980 pour la fidélité de ses employés (Saxenian, 1994), la Silicon Valley donne depuis la fin des années 2000 peu d'exemples de loyauté de plusieurs années d'un salarié à une même entreprise. Comme le déclare un employé chez eBay « rester 15 ans dans la même entreprise, ça ne se fait pas » (Chrysos, 2015, p. 64). Les membres de l'échantillon ont ainsi connu entre 2 et 6 entreprises entre le premier trimestre 2015 et le premier trimestre 2021.

Tableau 5. Nombre d'entreprises pour lesquelles les membres de l'échantillon ont travaillé (1er trimestre 2015-1er trimestre 2021)

Nb	Part des enquêtés (en %)
1	10
2	42,5
3	25
4	17,5
5	2,5
6	2,5

Source : LinkedIn

49 Le passage d'une entreprise à une autre donne l'occasion de se confronter à de nouvelles problématiques techniques en même temps qu'il permet une élévation statutaire et/ou salariale. Un entrepreneur va jusqu'à considérer lors d'une discussion informelle que « ici, tu leur files 100 euros de plus, ils traversent la rue ». Cette

versatilité coïncide avec une réorientation d'importance quand les développeurs fondent leur propre entreprise, changeant alors, aussi bien d'organisation que d'activité. Dans le corpus d'enquêtés, une majorité des développeurs a sauté le pas qui sépare le statut de développeur de celui d'entrepreneur : 10% étaient des travailleurs indépendants, 35% étaient des salariés, tandis que 55% des développeurs étaient devenus des chefs d'entreprise en 2021. Ce passage à l'acte entrepreneurial s'avère paradoxal si l'on considère le traitement dont les développeurs sont gratifiés au sein des entreprises de la Silicon Valley.

50 Cette proportion est en effet difficilement explicable si l'on garde en tête les faibles chances de mener une entreprise au succès, soit suivant les critères des investisseurs, une revente ou une entrée en bourse, après de 5 à 10 ans de croissance. D'autant moins que les développeurs disposent d'importants avantages quand ils sont employés par de grandes entreprises de la région, qui offrent de hauts salaires, des assurances maladies, des services de restauration et de transports gratuits, des congés maternité et paternité de plusieurs mois, des systèmes de garde des enfants, des abonnements à des salles de sport, etc, le tout pour une quantité de travail bien inférieure aux exigences de la vie entrepreneuriale. Pour les salariés, quitter l'organisation pour fonder leur propre entreprise suppose de renoncer pour une période indéterminée à ce niveau de confort. La création d'entreprise ne peut donc pas être qualifiée sous ce régime de décision rationnelle au sein d'un groupe professionnel particulièrement attaché à l'objectivité et la rationalité.

51 Si beaucoup mentionnent la culture entrepreneuriale de la Silicon Valley pour justifier ce changement de cap, ils soulignent également la volonté de conserver le contrôle de la valeur possiblement générée. En effet, la création d'entreprise s'avère pour les développeurs non pas le plus sûr moyen d'enrichissement mais celui qui représente le plus grand potentiel. Un développeur présente l'équation qui se présente aux jeunes professionnels de la région entre la condition salariale et celle de fondateur d'entreprise :

> « Bosser pour une grosse boite, genre Google, où j'ai fait les entretiens, ou une bonne startup mature ? Mon sentiment, c'est que je suis trop jeune. Je pense que je retente une startup pour la troisième fois et ensuite j'irai travailler dans une grosse boite. Quand tu viens d'être diplômé, le truc c'est pas d'aller direct chez Google. Parce que ok t'auras un bon salaire, t'auras 200K, des actions et plein d'avantages. Mais t'auras la même presque chose toute ta vie. J'en connais un, il s'est fait embaucher pour bosser dans une startup, un européen, il a mis 4 mois pour faire son visa, entre temps, la boite a levé 20 millions ; c'est devenu une grosse boite, ça l'intéresse plus. Parce que il y a pas de potentiel… Le développement est limité, tu peux pas devenir riche. Alors que si tu rentres tôt dans une boite, que tu en récupères quelques pour cent, tu peux vraiment devenir riche et avoir un impact. Encore plus si tu crées ta boite. Une fois que la levée de fond est faite, tu vas te retrouver avec un poste à responsabilité pour encadrer les nouveaux employés et un bon paquet d'argent. Donc beaucoup de jeunes diplômés veulent bosser ou lancer des petites startups ; tu perds en salaire, mais tu peux gagner gros, plusieurs millions, voire encore plus versus les 200 000 par an de Google. Après quand t'es entrepreneur, tu as aussi une *deadline* personnelle. Je peux pas rester entrepreneur pendant dix ans, gagner 1500 dollars, en *early stage*, à travailler 15h par jour… Le but c'est de développer, lever, recruter, moins être aux commandes et profiter un peu. » (CTO, homme, 34 ans, startup, série C)

52 L'écart potentiel de rétribution explique que certains préfèrent délaisser un statut objectivement avantageux au profit d'une position particulièrement incertaine. L'entrepreneuriat laisse en effet aux développeurs entrevoir la possibilité de conserver le « bénéfice du fondateur » (Hilferding, 1970), soit la plus-value tirée de la revente d'actions. En effet, à mesure que l'entreprise croît, la valeur des actions dont il a le

contrôle s'accroît, et peuvent être céder au prix que les investisseurs, puis les acquéreurs de titres sont prêts à payer sur le très dynamique marché de l'investissement technologique[9]. La création d'entreprise requiert dans cette perspective de conserver un alignement entre travail et valeur, temps et capital, tournant technologique et tournant professionnel.

53 Les développeurs, à travers l'obtention de certifications, la création de médiations et la création d'entreprise, cherchent ainsi à mieux maîtriser une trajectoire incertaine sur le long terme. La notion de tournant intentionnel se justifie en ceci qu'il s'agit pour eux d'initier une séquence de longue durée permettant une valorisation optimale de leur travail.

Conclusion

54 L'enquête a permis de mettre en avant trois résultats. Premièrement, elle tend à souligner l'importance du recours par les développeurs à des renforts destinés à affronter les manqués et imprévus qui surviennent de manière continue dans le travail de conception de logiciel. Deuxièmement, elle conduit, à partir d'une distinction entre projet et projection, à mettre en perspective le rapport qu'entretiennent les développeurs aux problèmes, à la fois impasse et points d'appui, et aux outils, en tant que médiation et moyen d'accéder à la reconnaissance professionnelle. Troisièmement, elle invite à considérer les dimensions temporelles d'une activité qui a souvent été abordée à partir des espaces et des scènes de programmation. Pour conjurer l'incertitude au sein d'une industrie caractérisée par la complexité technique et la volatilité des valeurs, les développeurs se tournent vers des points d'appui de court, moyen et de longs termes. L'ensemble de ces stratégies visent à composer avec l'irréductible incertitude de leur activité, l'absence ou l'inefficacité de modes de contrôle professionnels externes. À la différence des risques, connus et probabilisables, l'incertitude ne peut être régulée par des modes de contrôles externes, ou encore des mécanismes assurantiels. Cette impossibilité permet de mieux comprendre les manières de faire des développeurs au sein de la Silicon Valley. Bien que la valeur attachée au logiciel dans la Silicon Valley tienne à sa capacité à stabiliser efficacement et durablement des entités multiples dans l'espace, les développeurs sont exposés à un risque constant d'obsolescence dans le temps de leurs compétences, de leur production et du parcours jusqu'alors réalisé. La recherche de points d'appui vise à contrebalancer cette incertitude. Elle explique en dernier ressort l'ambivalence des développeurs à l'entreprise. Alors qu'ils se montrent attachés à l'entreprise qui les emploie, ils perdent rarement de vue le marché du travail, changeant d'organisation à un rythme régulier, ou créant leur propre entreprise de manière à optimiser leur trajectoire au sein d'un environnement dont la vitesse d'évolution rend les carrières extrêmement inégales et aléatoires.

Bibliographie

Bibliographie

ABBOTT Andrew D. (1988), The System of Professions. An essay on the Division of Expert Labor. Chicago (IL): Chicago University Press.
DOI : 10.7208/chicago/9780226189666.001.0001

ABBOTT Andrew D. (2001). Time Matters. On Theory and Method. Chicago (IL): Chicago University Press.
DOI : 10.3917/tt.019.0183

ABBOTT Andrew D. (2003). « Ecologies liées : à propos du système des professions » in MENGER Pierre-Michel (dir.), Les professions et leurs sociologies, Paris, Editions de la Maison des Sciences de l'Homme, pp. 29-50.

ALCARAS Gabriel (2020), « Des biens industriels publics. Genèse de l'insertion des logiciels libres dans la Silicon Valley », Sociologie du travail, Vol. 62, 3.

ALEXANDRE Olivier (2021). « Contrôler les hackers en toute liberté. Stratégies d'appropriation de valeur dans la Silicon Valley », Quaderni, 103, pp. 39-52.
DOI : 10.4000/quaderni.2063

ALEXANDRE Olivier & COAVOUX Samuel (2021). « Les influenceurs de la Silicon Valley. Entreprendre, promouvoir et guider la révolution numérique », Sociologie, 2.

AURAY Nicolas (2012). L'Alerte ou l'enquête, Paris, Presses des Mines.
DOI : 10.4000/books.pressesmines.3657

BECKER Howard S. (1988). Les Mondes de l'art, Paris, Flammarion.

BECKER Howard S. (1999). Propos sur l'art, Paris, L'Harmattan.

BECKER Howard S. & FAULKNER Robert (2011). Qu'est-ce qu'on joue maintenant? Le répertoire de jazz en action, Paris, La Découverte.

BERREBI-HOFFMANN Isabelle, BUREAU Marie-Christine & LALLEMENT Michel (2018). Makers. Enquête sur les laboratoires du changement social, Paris, Seuil.

BOBOC Anca & METZGER Jean-Luc (2020). « Les méthodes agiles et leurs contradictions », SociologieS [En ligne]. http://journals.openedition.org/sociologies/12471
DOI : 10.4000/sociologies.12471

BOLTANSKI Luc & CHIAPELLO Eve (1999). Le Nouvel esprit du capitalisme, Paris, Gallimard.

BOUCHER Henri (1984). Architecture de l'ordinateur. T.3. Logiciel, Toulouse, Cepadues Editions.

BOURDIEU Pierre & PASSERON Jean-Claude (1964). Les Héritiers. Les étudiants et la culture, Paris, Minuit.

BOURDONCLE Raymond (1993). « La professionnalisation des enseignants : les limites d'un mythe», Revue française de pédagogie, 105, pp. 83-114.

BROCA Sébastien (2013). Utopie du logiciel libre. Du bricolage informatique à la réinvention sociale, Neuvy-en-Champagne, Le Passager clandestin.

BROOKS Frederick (1996). The Mythical Man-Month. Essays on Software Engineering. Boston (MU), Addison-Wesley [1975].

BUTTON Graham & SHARROCK Wes W. (1995). « The Mundane Work of Writing and Reading Computer Programs » in HAVE Paul T., PATHAS George (dir.), Situated Order: Studies in the Social Organization of Talk and Embodied Activities, Washington DC, University Press of America, pp. 231–258.

CALLON Michel (1986). « Éléments pour une sociologie de la traduction. La domestication des coquilles Saint-Jacques dans la Baie de Saint-Brieuc », L'Année sociologique, 6, pp. 170-208.

CHANDLER Alfred (1977). The Visible Hand. The Managerial Revolution in American Business. Cambridge (MA), Belknap Press.
DOI : 10.1016/0007-6813(78)90014-9

CHRYSOS Paris (2015). Les Développeurs, Limoges, FYP.

COLEMAN Gabriella (2013). Coding Freedom: The Ethics and Aesthetics of Hacking, Princeton Princeton (NJ), University Press.
DOI : 10.1515/9781400845293

DAGIRAL Eric & PEERBAYE A. (2012). « Les mains dans les bases de données. Connaître et faire connaître le travail invisible », Revue d'anthropologie des connaissances, 6 (1), pp. 191-216.
DOI : 10.3917/rac.015.0229

DENIS Jérôme (2018). Le Travail invisible des données, Paris, Presses des Mines.
DOI : 10.4000/books.pressesmines.3934

ELDER Glen H. (1985). « Perspectives of the Life Course » in ELDER Glen (dir.), Life Course Dynamics. Ithaca: Cornell University Press, pp. 23-49.

FERRARY Michel (2001). « Pour une théorie de l'échange dans les réseaux sociaux. Un essai sur le

don dans les réseaux industriels de la Silicon Valley », Cahiers Internationaux de Sociologie, 111, pp. 261-290.
DOI : 10.3917/cis.111.0261

FLICHY Patrice (2001). L'Imaginaire Internet, Paris, La Découverte.

GOODY Jack (1979). La Raison graphique. a domestication de la pensée sauvage, Paris, Minuit.

GRELON André (1997). « Ecoles de commerce et formation d'ingénieurs jusqu'en 1914 », Entreprises et Histoire, 14-15, pp. 29-45.
DOI : 10.3917/eh.014.0029

HILFERDING R. (1970). Le Capital financier, Paris, Minuit [1910].

HUGHES Everett C. (1950). « Cycles, Turning point and Career », in Hughes Everett C. (1971). The Sociological Eye. Chicago (IL), Aldine, pp. 124-131.

HUGHES Everett C. (1996), « Le drame social du travail », Actes de la recherche en sciences sociales, 115, pp. 94-99.

JATON Florian (2021). The Constitution of Algorithm, Ground-Truthing, Programming, Formulating. Cambridge (MU): MIT Press.

KELTY Christopher (2008). Two Bits: The Cultural Significance of Free Software. Duke University Press, Durham.
DOI : 10.2307/j.ctv1198vx9

LÉCUYER Christophe (2007). Making Silicon Valley. Innovation and the Growth of High Tech (1930-1970). Cambridge (MU), MIT Press.

LÉCUYER Christophe (2020), «Driving Semiconductor Innovation: Moore's Law at Fairchild and Intel », Enterprise & Society.

LEVY Steven (1984). Hackers. Heroes of the Computer Revolution. Garden City: Doubleday.

MAHONEY Michael S. (2011). Histories of Computing. Cambridge (MA): Harvard University Press.
DOI : 10.1179/030801805X25927

MANOVICH Lev (2001). The Language of New Media. Cambridge (MU): MIT Press.
DOI : 10.22230/cjc.2002v27n1a1280

MARTINEZ Antonio G. (2016). Chaos Monkeys: Obscene Fortune and Random Failure in Silicon Valley. New York City: Harper.

MACKENZIE Douglas (2006). An Engine, Not a Camera: How Financial Models Shape Markets. Cambridge (MA), MIT Press.

MÉLÈS Baptiste (2016). « Approche philologique des langages de programmation », Technique et science informatiques, Vol. 35 (2), pp. 237-254.
DOI : 10.3166/tsi.35.237-254

MENGER Pierre-Michel (2002), Portrait de l'artiste en travailleur. Métamorphoses du capitalisme, Paris, Seuil.

MURILLO Luis Felipe (2020). « Hackerspace network: Prefiguring technopolitical futures? », American Anthropologist, Vol. 122, 2, pp. 207-221.
DOI : 10.1111/aman.13318

O'MARA Margareth (2019). The Code. Silicon Valley and the Remaking of America, New York, Penguin Press.

PYBUS Jennifer, COTÉ Mark (2021). « Did you give permission? Datafication in the mobile ecosystem », Information Communication and Society.
DOI : 10.1080/1369118X.2021.1877771

RIEDER Bernhard (2006). « Méchanologies et délégation. Pour un design orienté-société dans l'ère du web 2.0 », Manuscrit de thèse, Université Paris 8.

RIEDER Bernhard (2019). Engines of Order. A Mechanology of Algorithmic Techniques. Amsterdam, Amsterdam University Press.
DOI : 10.5117/9789462986190

ROT Gwenaële (2014). « Noter pour ajuster. Le travail de la scripte sur un plateau de tournage », Sociologie du travail, Vol. 56, 1, pp. 16-39.
DOI : 10.4000/sdt.4749

ROUEFF Olivier (2013). Jazz. Les échelles du plaisir. Intermédiaires et culture lettrée en France au XXe siècle, Paris, La Dispute.

SAXENIAN AnnaLee (1994). Regional Advantage: Culture and Competition in Silicon Valley and Route 128, Cambridge (MA), Harvard University Press.

SAXENIAN AnnaLee (1999). Silicon Valley's new immigrant entrepreneurs. Public Policy Institute of California.

SILES Ignacio (2013). « Inventing Twitter: An Iterative Approach to New Media Development », International Journal of Communication, 7, pp. 2105–2127.

TURKLE Sherkey (1984). The Second Self: Computers and the Human Spirit, New York City, HarperCollins Publishers.
DOI : 10.7551/mitpress/6115.001.0001

TURNER Fred (2006). From Counterculture to Cyberculture. Stewart Brand, the Whole Earth Network, and the Rise of Digital Utopianism. Chicago (IL), Chicago University Press.

WHITE Harrison C. (1970). Chains of Opportunity. System Models of Mobility in Organizations. Cambridge (MA), Harvard University Press.
DOI : 10.4159/harvard.9780674437203

WHITE Harrison C. (2008). Identity and Control: How Social Formations Emerge. Princeton: Princeton University Press.
DOI : 10.1515/9781400845903

Notes

1 « Le terme de *software* existe depuis 1953. *Logiciel* est le terme choisi en 1969 par des membres de la Délégation à l'informatique chargée du Plan Calcul pour désigner les parties modifiables d'un ordinateur en opposition au « *hardware* » et au « matériel » informatique. À son plus haut niveau de définition, le logiciel désigne l'ensemble des instructions qui doivent être suivies et exécutées pour faire fonctionner un système (programmes, scripts, applications, etc.). Ils sont le plus souvent réparties en trois familles : logiciels de programmation, système et application. » (Boucher, 1984 : 1)

2 Raymond Bourdoncle (1993) souligne les différences de constitution des professions entre les pays et en leur sein. En France, la voie dominante est celle d'une professionnalisation opérée via une lutte politique pour le contrôle des places et des statuts, où les corps d'État jouent un rôle central (modèle des corps d'Etat). Un autre modèle consiste en la formation d'une communauté de pairs qui définit leurs propres règles (modèle des confréries). Dans les pays anglo-saxons, les groupes professionnels se structurent dans la perspective d'une régulation du marché (le modèle des professions libérales).

3 En 1965, Gordon E. Moore, cofondateur d'Intel, affirme dans un article paru dans le magazine *Electronics* que la complexité des semi-conducteurs double tous les ans à coût constant. Cette « première loi de Moore » projette une augmentation exponentielle de la puissance des équipements informatiques (voir Lécuyer, 2020).

4 Ces notions sont à entendre en un sens probabiliste. En effet, les développeurs de la Silicon Valley représentent une population relativement privilégiée du point de vue des risques de la vie : ils sont en majorité jeune (35 ans de moyenne d'âge pour nos enquêtés), favorisés sur le plan socioéconomique (sur-représentation de personnes originaires de grandes villes, issues des professions intellectuelles supérieures), internationalisée et marquée par une forte domination masculine (80 à 90% d'hommes dans les entreprises de la Silicon Valley). Beaucoup d'entre eux ont conscience de bénéficier de conditions de vie et d'opportunités bien supérieures à leurs parents et leur milieu d'origine.

5 Traduit par l'auteur.

6 Voir Paris Chrysos et Olivier Alexandre, « Les valeurs de l'empathie. Création et captation de valeur chez les développeurs via les forums de Facebook et Google » (à paraître).

7 Le projet faillit mettre en péril l'équilibre commercial d'IBM. F. Brooks fut récompensé par le prix Turing pour sa réalisation.

8 F. Jaton a montré au sujet de la production des algorithmes par une équipe de jeunes chercheurs au sein d'un laboratoire d'informatique suisse que les objectifs d'origines dérivent vers la réalisation d'autres objectifs au cours des étapes de constitution de base de données de référence, de programmation et de formulation mathématique en dépit du travail de contrôle, d'évaluation et de scénarisation (Jaton, 2021 : 169).

9 Cf. François Xavier-Dudouet et Antoine Vion, « L'introduction de Google en bourse », séminaire *Capitalisme numérique et idéologies*, CNRS/Centre Internet et Société, avril 2021.

Pour citer cet article

Référence électronique
Olivier Alexandre, « Le code va changer », *RESET* [En ligne], 11 | 2022, mis en ligne le 09 avril 2022, consulté le 07 janvier 2024. URL : http://journals.openedition.org/reset/3498 ; DOI : https://doi.org/10.4000/reset.3498

Auteur

Olivier Alexandre
Chargé de recherche CNRS, membre du Centre Internet et Société

Droits d'auteur