Revue d'anthropologie des connaissances

20-1 | 2026
Techno-politiques des vulnérabilités et production des catastrophes et crises

Logiciel tel quel

Réflexions sur le livre Middle Tech de Paula Bialski

Florian Jaton

Édition électronique
URL : https://journals.openedition.org/rac/41171
DOI : 10.4000/15s7p
ISSN : 1760-5393

Traduction(s) :
Software actually - URL : https://journals.openedition.org/rac/41398 [en]

Éditeur
Société d'Anthropologie des Connaissances

Référence électronique
Florian Jaton, « Logiciel tel quel », *Revue d'anthropologie des connaissances* [En ligne], 20-1 | 2026, mis en ligne le 24 février 2026, consulté le 02 mars 2026. URL : http://journals.openedition.org/rac/41171 ; DOI : https://doi.org/10.4000/15s7p

Logiciel tel quel

Réflexions sur le livre Middle Tech de Paula Bialski

Florian Jaton

Introduction

1 C'est l'évidence même : nous sommes entouré·e·s de logiciels. Jour et nuit, nos cours d'action – qu'on le veuille ou non – ne cessent de se mêler à des dispositifs numériques dont l'agentivité procède de listes d'instructions exécutables par des ordinateurs. Ce qui était déjà vrai dans les années 1990 (Kling, 1996) l'est devenu de manière encore plus manifeste avec l'essor mondial de l'informatique mobile, dont les applications logicielles participent à rien de moins que nos trajectoires de vie.

2 D'où un mystère qui demeure insondable, du moins pour les practicien·ne·s des *Science & Technology Studies* (STS) qui, comme moi, s'intéressent au façonnage social des sciences et des techniques informatiques : pourquoi existe-t-il si peu de descriptions empiriques détaillées des processus pratiques par lesquels les logiciels en viennent à être construits ? Une fois mise de côté l'abondante littérature en gestion des logiciels, qui s'intéresse à la manière dont les choses devraient se dérouler plutôt qu'à la façon dont elles se déroulent effectivement (Suchman, 1995) ; celle en psychologie du logiciel, qui demeure cantonnée à des environnements artificiels où les variables sont soigneusement contrôlées, se privant ainsi de documenter ce qui se passe *in the wild* (Hutchins, 1995) ; et même – à mon grand désarroi – celle de l'anthropologie du logiciel, qui tend à se limiter aux habitudes culturelles des communautés de technicien·ne·s et de hackers sans véritablement décrire le travail concret de développement logiciel qui les passionne (Jaton, 2019) ; une fois mise de côté, en somme, toutes ces littératures intéressantes, mais qui ne s'intéressent *pas* au façonnage concret et *in situ* des logiciels, on constate que les descriptions empiriques se comptent sur les doigts d'une main (maximum deux)[1]. Et parmi ces descriptions rares et précieuses, pratiquement *aucune* n'a donné lieu à une monographie soignée, publiée par un éditeur académique respectable garantissant un processus d'évaluation sérieux. Si seulement davantage d'efforts en STS s'attachaient à décrire avec minutie les pratiques de celles et ceux qui

animent nos ordinateurs ! Nous en saurions bien plus sur nos attachements, et donc sur nous-mêmes (Hennion, 2017)[2].

3 J'ai tenté, ailleurs, de percer ce mystère, hélas sans grand succès[3]. Mais l'heure est aujourd'hui à la joie plutôt qu'à la plainte, car Paula Bialski – professeur de sociologie du numérique à l'université de Saint-Gall, en Suisse – a réussi à mener à bien l'une des toutes premières études ethnographiques de longue haleine sur le travail quotidien de développement logiciel. Grâce à une enquête en tout point unique et exigeante menée au sein d'une entreprise basée à Berlin, Bialski (2024) a accompli l'exploit de décrire le travail de développement logiciel tel qu'il se fait, sans fioritures cyber-modernistes. Et même si le texte n'est pas sans problèmes (que j'aborderai à la fin de ce commentaire), il convient de rappeler qu'une grande ethnographie n'est pas une ethnographie sans reproche : c'est une ethnographie sans peur (Jaton, 2023). Et c'est bien cette intrépidité – dont découle une série de propositions fortes que je m'apprête à résumer – qui fait de *Middle Tech* un ouvrage STS majeur ; une leçon d'empirisme, en somme, qu'il me semble crucial de signaler à la communauté, au cas où elle ne l'aurait pas vu passer[4].

Tech moyenne, tech réelle

4 Si le livre marque autant, c'est notamment parce qu'il opère un retour radical au réel. Loin d'être hypnotisé par la rhétorique des géants de la technologie, souvent perçus comme les leaders incontestés du développement logiciel, Bialski nous rappelle qu'ils ne sont en réalité que la partie la plus visible d'un écosystème beaucoup plus vaste et intéressant. En effet, la grande majorité des logiciels qui contribuent à nos infrastructures quotidiennes – des systèmes bancaires aux réseaux de transport, des administrations publiques aux hôpitaux – ne sont *pas* l'œuvre des géants de la Silicon Valley, comme le résume parfaitement Bialski dans cette formule concise : « Nous sommes si souvent confronté·e·s aux récits des grandes entreprises technologiques de la Silicon Valley que nous en oublions que la plupart de nos infrastructures numériques ne sont en réalité pas conçues par ces entreprises. » (p. 42, ma traduction).

5 Exit donc – pour le moment du moins – la big tech de Google, Microsoft, Amazon et compagnie : la vraie vie logicielle est ailleurs[5]. Peut-être dans les start-ups, ces fleurons de la nouvelle économie numérique, et leur capacité supposée à bouleverser les marchés établis ? Rien n'est moins sûr tant leur modèle d'affaires repose généralement sur leur rachat par de grandes entreprises, typiquement des géants de la tech. Leur motivation première consiste ainsi le plus souvent à lever des fonds afin de prouver la validité de leur concept (Mirowski, 2012 ; Rosental, 2021), orientation qui pèse lourdement sur le produit logiciel final, rarement optimisé, sécurisé ou pleinement fiable, et qui, par conséquent s'en trouve moins largement diffusé (du moins espérons-le). Là encore, Bialski résume très bien cette différence importante entre le développement de logiciels au sein de start-ups et au sein d'entreprises établies :

> Au cours de mon enquête de terrain, j'ai entendu d'innombrables récits de développeur·euse·s logiciel évoquant leurs débuts précipités, lorsque le développement logiciel consistait à « bricoler » des idées et à les mettre en œuvre rapidement, plutôt que de prendre le temps de maintenir un système cohérent. Ralentir et créer de la cohésion est souvent privilégié [en dehors des start-ups], non seulement pour rendre le code plus élégant, mais aussi pour maintenir la stabilité à long terme du système, réduire le risque de bugs potentiels et aider celles et ceux qui devront maintenir le code à l'avenir. (p. 58, ma traduction)

6 Si nous entendons documenter le développement logiciel dans sa forme ordinaire – et donc pervasive –, il convient de se garder d'un trop grand intérêt pour les grandes entreprises technologiques, comme pour les start-ups : leur poids médiatique/politique est disproportionné au regard de leur contribution effective à la production et à la maintenance des infrastructures logicielles. Où, dès lors, porter le regard ? L'un des principaux enseignements de l'ouvrage – à rebours d'un certain battage médiatique auquel les sciences sociales ne sont pas immunisées (Vinsel, 2021) – réside dans la nécessité de s'intéresser à ce que Bialski nomme la « moyenne tech » (*medium tech*) : des entreprises de taille intermédiaire (le plus souvent quelques milliers de salarié·e·s) qui, loin des projecteurs, conçoivent, maintiennent et adaptent discrètement les logiciels indispensables au tissu numérique du monde collectif (pp. 26-32).

7 Forte de cette perspective éclairée, notamment inspirée des *Middletown Studies* de Lynd et Lynd (1959 [1929]), Bialski est parvenue à obtenir un accès très enviable à une entreprise de moyenne tech, qu'elle nomme MiddleTech[6]. Implantée à Berlin, celle-ci occupe une position centrale dans le développement de logiciels de navigation et de routage, en particulier pour les constructeurs automobiles allemands. À l'issue d'une immersion ethnographique de six mois, prolongée par des allers-retours réguliers sur près de deux ans, Bialski a patiemment documenté le travail quotidien des différentes équipes techniques de MiddleTech : les développeur·euse·s *front-end* en charge des applications de navigation ; les développeur·eus·s *back-end* responsables du système d'exploitation, des bases de données et des algorithmes ; les *data scientists* conduisant des expérimentations sur de vastes ensembles de données ; ainsi que les équipes dites *DevOps* (*Development Operations*), chargées de l'infrastructure logicielle rendant possibles les tests et détections de bogues – autant d'acteurs essentiels à la vie de bureau infra-ordinaire (Perec, 1989), telle qu'elle se déploie dans la production des logiciels contemporains.

8 Aux premiers abords, l'accent mis par Bialski sur le « moyen », le « banal » et l'« ordinaire » (pp. 24-25) pourrait donner l'impression d'une sorte de nostalgie romantique pour un passé numérique soi-disant plus authentique. Mais il apparaît rapidement qu'il n'en est rien, notamment parce que, comme la plupart des autres entreprises de taille moyenne encore en activité aujourd'hui, MiddleTech a connu son lot de fusions, d'acquisitions et de changements de nom pour atteindre sa forme actuelle. Elle aurait également pu disparaître, comme c'est le cas de nombreuses entreprises moyennes (pp. 26-27). Mais elle est toujours bel et bien vivante et, comme beaucoup d'autres sur les cinq continents, ses quelques milliers d'employés continuent leur besogne quotidienne pour développer, et surtout maintenir, les logiciels qui soutiennent nos infrastructures numériques.

9 Ceci nous conduit à une question centrale, celle de la maintenance logicielle (*software maintenance*), qui émerge directement de l'examen salutaire que Bialski consacre à la moyenne tech. En mettant au jour les multiples formes, souvent discrètes, du travail d'entretien technologique, elle souligne de manière décisive que l'activité quotidienne des ingénieur·e·s et technicien·ne·s en entreprise relève bien davantage de la maintenance et de la mise à jour de logiciels vieillissants que de la quête d'innovation :

> La vie avec la technologie est généralement très éloignée des dernières avancées en matière d'invention et d'innovation, et consiste plutôt à maintenir le statu quo. [...] Nos logiciels et nos entreprises vieillissent. À mesure que nos logiciels vieillissent, nos projets logiciels deviennent de plus en plus complexes, évoluant vers des monstruosités multicouches [...] La plupart des logiciels que nous utilisons

> aujourd'hui sont le fruit d'années et d'années d'efforts de la part de développeur·euse·s qui ont réussi à bricoler un projet pour le faire fonctionner. (pp. 5-6)

10 Pas de romantisme donc, mais de la rigueur méthodologique qui place immédiatement l'ouvrage au cœur des enjeux les plus pressants d'aujourd'hui, à commencer par la nécessité de faire durer nos technologies logicielles vieillissantes. Et cette insistance, obligée, sur la maintenance logicielle est d'autant plus intéressante qu'elle s'apparente également à la prise en soin d'*actifs* dans le cadre d'*efforts de capitalisation* :

> Chez MiddleTech, cet actif numérique est un logiciel de cartographie et de navigation, appelé « moteur cartographique », qu'ils ont vendu et continuent de vendre à des entreprises tierces qui ont besoin de cartes dans leurs produits (tels que des voitures ou d'autres applications nécessitant une fonctionnalité cartographique). Grâce à cette intégration prolifique de son logiciel, l'entreprise est en mesure de tirer profit des revenus générés par son actif, créé il y a plusieurs années… [De manière plus générale], les entreprises de moyenne tech ne génèrent pas autant de revenus que leurs homologues de la big tech, et leurs possibilités de réinvestissement dans la recherche ou l'innovation sont assez limitées. Compte tenu de cette limitation, le travail logiciel chez MiddleTech était principalement consacré à la maintenance et à la réparation. (pp. 34-35, ma traduction)

11 Le regard de Bialski sur la moyenne tech qui irrigue les infrastructures numériques réunit ainsi les études sur la maintenance (Denis & Pontille, 2025) et celles sur l'actifisation[7] (Birch & Muniesa, 2020) autour d'un nouvel objet : le travail logiciel quotidien. Cette emphase sur des activités spécifiques, mais ordinaires met en lumière, et donc documente, l'un des centres névralgiques de la reconfiguration sociotechnique du capitalisme contemporain. Cela vous semble-t-il un peu exagéré ? Rappelez-vous alors le chaos provoqué par un minuscule bogue dans une mise à jour de sécurité de l'entreprise de moyenne tech CrowdStrike en juillet 2024. Une seule erreur de maintenance dans l'actif numérique le plus précieux de CrowdStrike – implémenté dans le noyau de Microsoft Windows – et le monde s'est arrêté pendant quelques heures (Satariano *et al.*, 2024). Infra-ordinaire, logiciel, moyenne tech, maintenance, actifisation : cinq termes d'un ensemble qui nous concerne toustes.

Les entretiens ne suffisent pas

12 Bialski évite ainsi adroitement le piège de l'attention excessive accordée aux big tech et aux start-ups, ce qui lui permet, en retour, de souligner le rôle central des opérations de maintenance logicielle, qui constituent autant d'efforts de capitalisation. Mais elle parvient également à éviter un deuxième piège tout aussi insidieux : celui de se fier uniquement à des entretiens pour comprendre le travail quotidien des développeur·euse·s logiciel.

13 Bien qu'il s'agisse d'un sujet classique des sciences sociales qualitatives – largement débattu depuis des décennies (voir par exemple Becker & Geer, 1957, 1958 ; Atkinson & Coffey, 2004) –, il mérite certainement d'être réexaminé ici, tant il constitue une condition essentielle de cette remarquable enquête. Car Bialski aurait très bien pu limiter son étude à une série d'entretiens. C'est d'ailleurs ce qu'elle a fait au départ, notamment avec des ingénieur·e·s de la Silicon Valley travaillant pour Meta, l'organisation Wikimedia ou en tant qu'entrepreneur·euse·s indépendant·e·s (pp. 2-3). Sans surprise, ces développeur·euse·s ont évoqué des nuits blanches passées à

perfectionner des logiciels censés être « révolutionnaires », dans une logique de disruption et de performance totale. Une rhétorique courante – omniprésente dans les discours managériaux, qui opèrent comme ressources discursives directement mobilisables par les personnes interrogées – qui donne l'impression que l'amélioration continue et la quête de la perfection sont les principaux moteurs du développement logiciel. Comme le note Bialski :

> Les *techies* de la Silicon Valley que j'ai rencontrés semblaient croire que la technologie se devait d'être formidable et que le travail dans ce domaine se devait être dur et fastidieux. [...] Ce qui m'a frappé, c'est le discours récurrent selon lequel les développeur·eus·s logiciel se dévouaient à travailler tard le soir pour perfectionner quelque chose « hors du commun ». Les logiciels n'étaient pas simplement assemblés pour fonctionner, tomber en panne de temps en temps et être maintenus ; ils étaient destinés à bouleverser, révolutionner les codes et à innover d'un seul coup. (pp. 2-3)

14 Mais après seulement quelques jours sur le terrain, il apparaît clairement que la réalité ordinaire est très différente. Si le discours associé à l'excellence et à l'amélioration constante existe bel et bien et circule, il trouve en fait très peu d'expression pratique :

> Au cours de mon travail de terrain, j'ai découvert que, bien que ces indicateurs, méthodes et modèles d'excellence et d'amélioration soient présents dans la culture d'entreprise de MiddleTech, la réalité est tout autre. Sur le plan discursif, les environnements logiciels en entreprise peuvent être considérés comme des usines d'accélération technologique, où la technologie est constamment mise à jour afin d'améliorer et de tendre vers l'excellence. Pourtant, dans la réalité quotidienne, souvent banale, les développeur·euse·s logiciel s'inspirent davantage des principes et des pratiques du *good enough*. (p. 11, ma traduction)

15 Je traiterai plus loin de la notion centrale de *good enough* (que je conserve en anglais, faute de traduction satisfaisante), la grande observation empirique et proposition théorique du livre. Mais pour l'heure, il convient de souligner la leçon méthodologique – qui n'apparaît clairement que rétrospectivement – consistant à se méfier des discours conventionnels sur le développement logiciel qui, le plus souvent, reprennent, propagent (Boullier, 2023) et amplifient des idéalisations morales. On peut établir ici un parallèle avec les études sociales des sciences et leur confrontation passée avec la philosophie des sciences : tout comme l'épistémologie a longtemps proposé une vision théorique et normative des sciences très éloignée des pratiques réelles des chercheur·euse·s (Latour, 1987 : 2-17), la littérature en management logiciel propose le plus souvent une représentation abstraite et idéalisée du développement logiciel, davantage façonnée par des enjeux politiques et organisationnels que par la réalité concrète du travail quotidien des développeur·euse·s.

16 Pour autant, il est important de préciser que le livre de Bialski n'est pas un exercice critique visant à démontrer que les acteur·ice·s sont prisonnier·e·s d'une forme d'*illusio*. Le propos est en réalité beaucoup plus simple. Premièrement, lorsqu'iels répondent à un·e enquêteur·ice qu'iels connaissent peu (ou pas du tout), les acteur·ice·s fournissent souvent les réponses qu'iels pensent être attendues. Deuxièmement, ces acteur·ice·s accordent suffisamment d'importance à ces attentes supposées pour les propager davantage. Bialski résume bien cette idée au tout début de son livre :

> Les travailleur·euse·s rejettent les notions d'excellence dans la pratique, mais je tiens à souligner qu'un discours hégémonique sur l'excellence existe bel et bien *en théorie*. Les entreprises logiciel, comme beaucoup d'entreprises, propagent une idéologie d'excellence et d'amélioration, tant en ce qui concerne les logiciels

> qu'elles développent que le type de travail nécessaire à leur fabrication. (p. 7, ma traduction. Italiques dans l'original)

17 L'ouvrage parvient ainsi à lever plusieurs obstacles, ne serait-ce que pour préparer le terrain d'une enquête rigoureuse sur le travail concret du développement logiciel (une démarche pratiquement sans précédent, faut-il le rappeler). Mais qu'en est-il des résultats effectifs de cette aventure ethnographique inédite ?

Good enoughing, ou le salut du développement logiciel privilégié

18 La proposition principale du livre est que le travail de développement logiciel en entreprise – du moins chez MiddleTech (mais très probablement aussi dans bien d'autres entreprises de taille moyenne) – est façonné par une activité que Bialski nomme *good enoughing*[8]. Bien que l'autrice ne définisse pas précisément ce qu'elle entend par « activité » – un terme qui semble parfois interchangeable avec celui de « culture » –, nous pouvons en déduire qu'il s'agit d'un ensemble de pratiques orientées vers le même type d'accomplissement. Mais quel est cet accomplissement dans le cas du *good enoughing* ? Réponse approximative : la capacité à déterminer de manière adéquate où s'arrêter dans le processus de développement. En ce sens, les développeur·euse·s étudié·e·s par Bialski apparaissent comme de véritables aristotélicien·ne·s, sans nécessairement en être conscient·e·s : pour elleux aussi, *la série doit bien s'arrêter quelque part et ne pas être infinie*. Mais reste encore la question délicate de savoir *comment* fixer cette limite sans compromettre ni la qualité du logiciel en cours de fabrication, ni l'investissement des développeur·euse·s dans le processus. C'est là toute la subtilité du *good enoughing*, qui ne peut être assimilé à une forme de paresse, comme le souligne clairement Bialski dès la première page de son ouvrage : « L'idée que je défends tout au long de ce livre est que parvenir au *good-enoughness* est une entreprise incroyablement complexe et intéressante » (ma traduction).

19 Si le *good enoughing* n'est certainement pas une activité spécifique au développement logiciel, il en est aujourd'hui un élément central, notamment en raison de plusieurs changements qui ont récemment profondément affecté la matérialité logicielle. Le premier changement est ce que l'on appelle communément le « logiciel-service » (*software-as-a-service*), un modèle de distribution qui a coïncidé avec l'essor d'Internet dans les années 2000. Avant l'ère du logiciel-service, les logiciels étaient principalement distribués sous forme de produits physiques (*shrink-wrapped software*), nécessitant une installation et des mises à jour manuelles par l'utilisateur·rice. Avec l'essor d'Internet dans les années 2000, le modèle du logiciel-service a permis d'accéder aux logiciels en ligne, via des plateformes dédiées. Cela a permis aux développeur·reuse·s de contrôler en permanence leurs applications, leur permettant de « pousser » les mises à jour logicielles quasi en temps réel, en seulement quelques clics (p. 69).

20 Un autre changement récent est lié à l'essor du stockage dit en nuage. Au lieu d'être stocké localement sur des machines à l'interne, le code composé par les développeur·euse·s est désormais hébergé sur des serveurs distants, souvent via des fournisseurs de services en nuage tel qu'Amazon Web Services (AWS). Cette transition a apporté une plus grande sécurité, notamment en libérant les logiciels des contraintes des infrastructures physiques locales. Cependant, elle a également engendré une forme

d'hyper-instabilité, où le logiciel n'est plus une entité fixe, mais plutôt un ensemble dynamique de requêtes entre interfaces.

21 Ces changements ont conduit à la formation progressive d'une culture (ou habitude, p. 71) de la mise à jour au sein du développement logiciel en entreprise, qui incite à considérer le logiciel comme un projet continu, sans véritable version finale. Sur le plan pratique, cette habitude tend à normaliser les cycles de développement itératifs, où des mises à jour fréquentes remplacent les versions stables et définitives. Cela encourage à son tour une approche où le logiciel est constamment corrigé en fonction des commentaires des utilisateur·rice·s ou des responsables, ajoutant ainsi un niveau supplémentaire d'imprévisibilité. Et si le développement de logiciels a une longue histoire d'échecs et de déceptions (Brooks, 1975), ces changements récents semblent avoir accru la possibilité de problèmes inopportuns, ce qui suggère à son tour la nécessité de faire preuve de *prudence* :

> Le logiciel-service, le stockage en nuage et la culture de la mise à jour qui a résulté de ces changements ont ancré l'échec et la répétition dans la culture de travail des programmeur·euse·s. Ceci peut entraîner des dysfonctionnements dans un système logiciel, et remettre les choses en ordre lorsque cela se produit demande également beaucoup d'énergie, diverses formes de connaissances et du temps. (p. 71)
>
> Travailler sur des logiciels signifie que différentes formes hétérogènes de connaissances sont en concurrence constante les unes avec les autres, et que la base de code s'étend sans être toujours rafraichies, créant ainsi des systèmes complexes hérités et interdépendants qui sont difficiles à appréhender. Ces deux facteurs conduisent à une culture de travail particulière axée sur la « débrouillardise », le compromis et la confusion. (p. 69)

22 Cette confusion favorise une certaine prudence, voire un certain conservatisme, dans le développement logiciel en entreprise, également affecté par la présence croissante du code hérité (*legacy code*, p. 86), entendu comme le code issu de projets antérieurs ou écrit par d'autres développeur·euse·s, qu'il provienne d'une acquisition, d'une équipe précédente ou de phases antérieures de développement. Ce code hérité – que le stockage en nuage permet de conserver à moindre coût – est ambivalent, car il peut constituer une base rassurante, éprouvée et généralement fonctionnelle. Mais le code hérité peut également être source de confusion et de complexité, car il est parfois écrit dans des langages différents (parfois obsolètes) ou a été modifié à plusieurs reprises, ce qui rend sa structure difficile à comprendre. Certain·e·s développeur·euse·s considèrent même le code hérité comme un fardeau, une sorte de « monstre » fastidieux à maintenir et qui rend difficile l'ajout de nouvelles fonctionnalités (pp. 87-88).

23 En somme, un·e développeur·euse logiciel n'est jamais seul·e ; iel est entouré·e d'une myriade de collègues actuel·le·s et ancien·ne·s qui ont produit une myriade de lignes de code plus ou moins obscures et monstrueuses. Le problème pratique auquel sont confronté·e·s quotidiennement de nombreux·ses développeur·euse·s logiciel en entreprise de moyenne tech (et probablement ailleurs aussi) est donc le suivant : comment composer au mieux avec le code hérité, dont la prolifération s'est accélérée avec l'essor du développement logiciel en nuage ? Et c'est précisément là que le *good enoughing* entre en jeu, de manière quasi fonctionnelle, en tant que délicate activité d'évaluation face à de multiples contraintes.

24 Cette activité contrainte d'évaluation imprègne tout le travail de développement chez MiddleTech. Mais nulle part n'est-elle plus importante et plus visible que pendant les fameuses journées de finalisation des fonctionnalités (*feature-complete days*), où de

nombreuses fonctionnalités doivent être achevées et intégrées dans le code principal. Pour illustrer cela, Biaslki prend l'exemple d'une équipe qu'elle a suivie pendant plusieurs semaines et qui travaillait sur les fonctionnalités de routage pour les voitures électriques. Cette équipe, comme la plupart des centaines d'autres équipes de MiddleTech, s'inspire de la méthodologie Scrum (nous y reviendrons plus tard) et structure son travail en sprints, qui commencent par une réunion de planification définissant les tâches sous forme de tickets. Chaque ticket correspond à une étape de développement spécifique, comme dans ce cas l'intégration d'une bibliothèque de stations de recharge dans la base de données de routage. Une fois attribuée, la tâche est développée individuellement, puis soumise à une révision du code sur Gerrit (un outil de collaboration en ligne), où chaque contribution est évaluée et notée par les autres membres de l'équipe, selon des critères plus ou moins arbitraires (qui peuvent faire l'objet de débats, cf. p. 93). Et toute cette organisation est elle-même tournée vers les journées de finalisation des fonctionnalités, ces journées cruciales qui marquent la finalisation et l'intégration simultanée de multiples fonctionnalités issues des sprints dans le code principal. Mais ce sont aussi les journées où les imprévus techniques explosent : conflits de fusion, bogues critiques, fonctionnalités incomplètes ou instables sont légion :

> Les développeurs ne se concentrent pas uniquement sur la création de logiciels géniaux, mais essaient également d'éviter que tout ne parte en vrille ou, pour ainsi dire, que toute la maison ne parte en fumée. Être juste *good enough* pour survivre à un incendie gigantesque est déjà un exploit. Dans cette culture où l'on s'efforce de maintenir les choses en état, une équipe de développement comprend qu'elle ne peut pas fournir un logiciel parfait et se contente d'un code qui est *good enough*. (p. 96)

25 Mais le *good enoughing* ne concerne pas seulement les programmeur·euse·s qui tentent de maintenir en vie des logiciels composés d'éléments disparates qui cherchent tous à aller dans leur propre direction (générant ainsi des erreurs) ; il concerne également les managers et les outils de gestion qu'iels mobilisent. Cela est particulièrement évident, par exemple, avec la méthodologie Scrum, déjà mentionnée ci-dessus et examinée en détail au chapitre 4 du livre (pp. 99-131).

26 En tant que méthode dite agile, Scrum vise à organiser le travail des développeur·euse·s autour de cycles courts et itératifs appelés sprints. Si cette approche offre aux développeur·euse·s une réelle autonomie et flexibilité, elle constitue également un outil de contrôle managérial, permettant de mesurer la productivité des équipes et d'en rendre compte aux instances décisionnelles et stratégiques. Cependant, en fonction des contraintes du moment, la version de Scrum mise en place à MiddleTech semble laisser une certaine marge de négociation, intégrant ainsi de manière informelle certains aspects du *good enoughing*. Mais au final, et de façon intéressante, ces ajustements pragmatiques ont tout de même tendance à être reformulés dans le langage de l'excellence, de l'amélioration continue et de l'innovation, réinscrivant ainsi ces négociations dans la rhétorique managériale traditionnelle.

27 En ce sens, si les outils de gestion tels que Scrum perdurent, ce n'est peut-être pas simplement parce qu'ils rendent le développement logiciel plus efficace ou innovant. Les méthodologies dites agiles, telles que Scrum offrent également aux managers (qui sont souvent d'ancien·ne·s développeur·euse·s) un cadre leur permettant de rester en phase avec les principes cyber-modernistes idéalisés du génie logiciel, tout en laissant aux développeur·euse·s une certaine marge de manœuvre. En bref, la logique du *good*

enough semble être lâchement intégrée dans les récents modèles de gestion agiles, qui visent à équilibrer les exigences de contrôle managérial avec la flexibilité évaluative des développeurs, leur permettant ainsi de s'adapter à l'évolution des besoins sans remettre entièrement en cause les normes officielles, traditionnelles et « hors-sol » du génie logiciel. Comme le note Bialski :

> Ce discours sur l'excellence et la performance optimale imprègne l'environnement logiciel en entreprise, et des méthodes telles que Scrum en sont la manifestation. Scrum propose aux travailleur·euse·s un ensemble de schémas, d'histoires, de rituels et de routines durables qui les guident, imposant une transparence constante dans le but d'atteindre une performance optimale. Ces méthodologies servent donc plusieurs objectifs, qui sont en contradiction avec l'objectif initial de Scrum et d'autres méthodes similaires : elles contribuent à définir l'identité des développeur·euse·s logiciel (en tant que personnes dont le travail et les machines avec lesquelles iels travaillent ne peuvent être « domptées » par une méthode) et aident à définir les objets de leur attention (les logiciels passent avant tout, pas les clients, les utilisateurs ou les performances optimales). Dans la pratique, des méthodes telles que Scrum imposent aux développeur·euse·s des rituels et des routines quotidiennes auxquels iels ne se conforment que partiellement (de façon *good enough*) afin d'apaiser leur hiérarchie tout en reproduisant une culture d'imprévisibilité et de soin apporté à leurs logiciels. (p. 131).

28 Mais si l'activité de « *good enoughing* » permet à la hiérarchie de sauver la face (car cette activité permet de rendre vaguement commensurables les pratiques effectives de développement logiciel à sa conception idéalisée), aux développeur·euse·s de conserver une certaine forme de contrôle sur leur travail (car cette activité leur permet de définir des moments d'arrêt et de reprise), et aussi de composer avec la prévalence du code hérité (lui-même issu des récents changements sociotechniques dans les matérialités logicielles), Bialski nous rappelle qu'il s'agit également, et en fin de compte, d'un *privilège* accordé à certain·e·s travailleur·euse·s et certaines entreprises. Celles et ceux qui peuvent se permettre le *good enoughing* bénéficient généralement d'environnements de travail plus sûrs et d'une plus grande sécurité de l'emploi, des conditions qui sont loin d'être négligeables. Les développeur·euse·s travaillant dans des fermes de codage externalisées, par exemple, n'ont souvent pas la stabilité nécessaire pour pratiquer le *good enoughing*, car ils doivent respecter des délais stricts et endurer l'insécurité de l'emploi. De même, seules les entreprises disposant d'actifs logiciels établis et générateurs de revenus à maintenir, telles que MiddleTech, Microsoft ou Crowstrike, peuvent se permettre d'adopter une approche *good-enough*. Ces entreprises, qui ont construit leurs principaux actifs dans le passé, bénéficient désormais d'une stabilité financière leur permettant d'assumer, voire d'institutionnaliser, une partie de la dimension concrète du développement logiciel. En revanche, les petites entreprises qui peinent à se faire une place sur le marché ne peuvent se permettre un tel luxe. Comme le résume bien Bialski :

> Être une entreprise *good enough* comme MiddleTech signifie également encourager une inégalité dans les rythmes et exigences de travail, permettant à certaines personnes de se soustraire à l'hyperproductivité tout en profitant du travail méconnu d'autres développeur·euse·s logiciel et prestataires de services. (p. 16-17)

29 L'activité de *good enoughing* est donc fondamentalement *ambivalente* : elle est à la fois essentielle à la bonne appréciation du mode d'existence du logiciel, mais contribue également – et simultanément – au maintien des inégalités qui assurent sa perpétuation. Biaslki a réussi à rendre plus intelligible ce phénomène anthropologique

fascinant, qui dépasse certainement le cadre du développement logiciel[9] : un bel exploit !

Un paradoxe: mais où sont les pratiques situées de programmation ?

30 Le livre regorge d'observations intéressantes, comme ce que Bialski appelle le « mythe de la connaissance » (c'est-à-dire la tendance des développeur·euse·s à feindre une expertise dans des domaines techniques qui dépassent leurs connaissances réelles ; pp. 79-83) ou encore les méthodes créatives et souvent fantaisistes utilisées pour estimer la durée des différents projets (qui s'appuient souvent sur l'échelle des tailles de t-shirts ! pp. 83-85). Il convient toutefois également de mentionner ici ce qui constitue sans doute la principale lacune du texte : l'absence surprenante de descriptions détaillées de la programmation informatique en action. L'éléphant dans la pièce : alors que le code informatique est au cœur des discussions, des incertitudes, des joies et des défis rencontrés par les employé·e·s de MiddleTech, il n'est presque jamais fait mention des moments où ce code est inscrit manuellement.

31 Si je dis « presque », c'est parce que ce type de situation est brièvement abordé dans quelques pages au début du livre (pp. 51-54). La discussion tourne autour d'un état abstrait appelé « flux » (*flow*), dans lequel les programmeur·euse·s entreraient lorsqu'iels travaillent intensément sur leur code, ces complexes listes numérotées de symboles. Cet état de flux est décrit comme une synchronisation profondément intuitive et immersive entre le ou la programmeur·euse et sa machine, souvent associée à des signes extérieurs de retrait, telle que le port de gros écouteurs pour s'isoler des autres personnes dans un bureau en espace ouvert. Empruntant la terminologie de la romancière et ancienne programmeuse Ellen Ullman, fréquemment citée dans le livre, le flux évoque une sorte de proximité symbiotique entre l'humain et l'ordinateur[10].

32 Si ce court passage me semble paradoxal, c'est notamment parce qu'il est tiré d'entretiens, faisant ainsi écho au discours conventionnel sur le développement logiciel – discours qui est précisément la cible des critiques de Bialski, et qui sous-tend même la pertinence de son importante approche ethnographique. En d'autres termes, l'aura presque mystique – et assez abstraite – qui entoure les situations de programmation découle directement d'entretiens imprégnés de formules toutes faites, facilement mobilisables, car largement diffusées et socialement acceptées. Mais pourquoi ne pas aller au-delà de ces récits préétablis ? Car s'ils contiennent des éléments qui sont sûrement pertinents (et qui doivent être respectés), ils ignorent complètement l'environnement de travail réel des développeur·euse·s, avec sa panoplie de grands moniteurs affichant une multitude de fenêtres interactives formant un véritable établi numérique relié à de nombreux autres établis via différents espaces de discussion (forums, chats, etc.). Au-delà de l'omission initiale de Bialski, un oubli encore plus déroutant et omniprésent persiste : la tendance à considérer la programmation informatique de manière isolée, détachée de sa dimension manifestement hyper-graphique, – matérielle et – située (Goody, 1977 ; Latour, 1986).

33 Le fait que même l'étude ethnographique la plus approfondie et la plus sophistiquée sur le développement logiciel ait du mal à saisir la programmation informatique en action – s'appuyant plutôt sur des entretiens qui renforcent des stéréotypes cyber-modernistes

abstraits – me laisse perplexe. Peut-être qu'une telle entreprise est tout simplement impossible ? Et pourtant, comme l'ont récemment indiqué Pütz (2021), Fischer (2025) et moi-même (Jaton, 2021 : 135-195 ; 2025), il semble y avoir des pistes à explorer, même si elles exigent une bonne dose d'obstination. Mais une approche ethnographique plus fine, axée sur les gestes, les outils et les spatialités numériques des développeur·euse·s logiciel, aiderait sans nul doute à dépasser nos récits conventionnels, en offrant une compréhension plus riche et plus incarnée du travail de programmation informatique dont nous dépendons de plus en plus. Ce faisant, cette *micro-sociologie* potentielle de la programmation informatique (Jaton, 2022) pourrait même ouvrir la voie à des découvertes socio-ergonomiques, un peu à l'instar des travaux pionniers d'Edwin Hutchins sur la signalisation de la vitesse dans les cockpits des avions de ligne (1995).

Conclusion: l'ethnographie est un exercice d'auto-défense et d'équilibrisme (et donc de diplomatie)

34 Est-ce que ne pas être parvenu à aller au-delà – ou en deçà – des discours idéalisés sur les pratiques de programmation signifie que l'entreprise ethnographique de Bialski n'est pas une réussite totale ? Mais on ne voit pas que les ethnographes soient destiné·e·s à réussir totalement, pas plus que Louis Pasteur ou Marie Curie. Documenter minutieusement une partie substantielle de la vie infra-ordinaire du développement logiciel en entreprise constitue une contribution si importante à l'étude sociale des sciences et techniques informatiques qu'il serait plus qu'injustifié de blâmer Bialski pour cette lacune (même si je continue de penser qu'elle est importante). Ce serait d'autant plus une erreur que le livre nous rappelle avec force le rôle central de l'ethnographie dans la composition laborieuse d'un monde commun.

35 Car le travail de Bialski montre clairement que le développement logiciel n'est pas une question d'optimisation. Il s'agit avant tout de *problématisation*. C'est là en effet un processus orienté-problème, où les frictions inhérentes – souvent dues à la diversité des spécifications des nombreux composants d'un projet – jouent un rôle crucial dans la solidité *et* la fragilité des logiciels intégrés à nos infrastructures quotidiennes. Ces défis donnent lieu à des modes d'activité spécifiques, tels que le *good enoughing*, qui permet aux développeur·euse·s de faire collectivement face aux inévitables complexités sociales de leur profession.

36 En conséquence, les rêves cyber-modernistes d'optimisation du développement logiciel et de réduction drastique de ses erreurs – par exemple en s'appuyant largement sur du code confiant généré automatiquement par de grands modèles de langage entraînés sur des forums de programmeur·euse·s tels que Stack Overflow – sont voués à l'échec. En effet, ce sont précisément les nombreuses erreurs et inexactitudes qui alimentent le développement logiciel et organisent sa production. Le travail méticuleux de Bialski nous invite donc à repenser le développement logiciel, non pas au prisme des fonctionnalités et des performances, mais bien plutôt au prisme des erreurs, des bogues et des pratiques sociales conçues pour composer avec eux, telles que le *good enoughing*.

37 Ces observations clés, qui ne sont pas sans précédent, sont présentées ici avec une remarquable rigueur, ce qui les rend particulièrement pertinentes aujourd'hui, à l'heure où les programmeur·euse·s figurent fréquemment sur les listes des professions

susceptibles d'être remplacées par des programmes d'intelligence artificielle. Ces déclarations extravagantes doivent évidemment être considérées à la lumière d'une économie de l'attention friande de sensationnalisme, mais elles peuvent néanmoins avoir des effets réels, notamment en raison de l'opportunité cynique qu'elles offrent de déguiser des mesures d'austérité classiques en innovations numériques de pointe (Merchant, 2025). Dans ce contexte, les descriptions réalistes de Bialski peuvent servir de contre-feu, offrant aux travailleur·euse·s un moyen tangible de mettre en évidence – et de défendre – les aspects intrinsèquement non optimisables de leur artisanat.

38 Pour l'instant, cependant, le potentiel d'autodéfense des propositions de Bialski semble entrer en collision avec l'attachement durable de nombreux·ses développeur·euse·s à la vision moderniste dominante de leur travail, centrée sur l'excellence, l'amélioration continue et la résolution efficace des problèmes. Les dernières pages du livre illustrent cette tension, en relatant les réactions mitigées de plusieurs développeur·euse·s de MiddleTech lorsque, sur une terrasse berlinoise, Bialski révèle que son compte rendu d'enquête se concentrera sur l'activité du *good enoughing* (pp. 157-160). Et c'est précisément là qu'apparaît un deuxième défi très délicat des études ethnographiques des sciences et des techniques, au-delà de la conduite méticuleuse de l'enquête elle-même : le raffinement des propositions en collaboration avec celles et ceux qui ont rendu l'enquête possible.

39 Car les propositions de Bialski constituent en partie un acte de dévoilement : les pratiques réelles de développement logiciel divergent considérablement des discours « officiels » (pour ne pas dire « légitimes ») qui prétendent les définir. Mais s'arrêter à cette révélation (modérée) serait irresponsable, car cela placerait l'ethnographe dans le rôle d'une sociologue-reine (Rancière, 2004 : 165-202). Il est donc essentiel d'approfondir la question et d'évaluer si le compte rendu alternatif proposé est acceptable pour les praticien·ne·s. De façon intéressante, le livre suggère que ce n'est pas tout à fait le cas (p. 160). En effet, certaines valeurs constitutives du discours « officiel » cyber-moderniste – bien que potentiellement dangereuses, car risquant d'alimenter les forces qui s'opposent au développement logiciel ou qui risquent même d'accélérer son démantèlement – continuent de faire sens. Cela soulève une question *compositionniste* (Latour, 2010) fondamentale, qui appelle à une redéfinition institutionnelle : serait-il à la fois politiquement approprié et méthodologiquement valide d'adjoindre certains idéaux cyber-modernistes aux descriptions empiriques, non pas à titre de concession, mais comme moyen stratégique et moral de réorienter le discours général sur les logiciels vers les réalités concrètes du travail de développement, afin de mieux le préserver ?

40 Il s'agit là clairement d'un exercice d'équilibrisme qui dépasse le cadre du livre de Bialski. Mais son enquête a le mérite immense d'avoir réussi à jeter les bases d'un travail *diplomatique* (Janicka, 2023) sur les valeurs les plus à même de préserver les pratiques du développement logiciel. Et par là même, son enquête a le mérite plus immense encore de nous rappeler qu'une grande ethnographie n'est pas celle qui prononce, mais bien celle qui requiert.

BIBLIOGRAPHIE

Alauzen, M. (2025) Paula Bialski, Middle Tech. Software Work and the Culture of Good Enough, *Revue d'anthropologie des connaissances*, *19*(1). https://doi.org/10.4000/13eg8

Amrute, S. (2016). *Encoding Race, Encoding Class: Indian IT Workers in Berlin*. Durham : Duke University Press.

Atkinson, P., & Coffey, A. (2004). Revisiting the relationship between participant observation and interviewing. Dans J. Gubrium, & J. Holstein (Eds). *Postmodern Interviewing* (pp. 109-122). Thousand Oaks : SAGE Publications Ltd. https://doi.org/10.4135/9781412985437.n6

Becker, H. S. & Geer, B. (1957). Participant Observation and Interviewing: A Comparison. *Human Organization*, *16*(3), 28-32. https://doi.org/10.17730/humo.16.3.k687822132323013

Becker, H. S., & Geer, B. (1958). "Participant Observation and Interviewing": A Rejoinder, *Human Organization*, 17(2), 39-40. https://doi.org/10.17730/humo.17.2.mm7q44u54l347521

Bialski, P. (2024). *Middle Tech: Software Work and the Culture of Good Enough*. Princeton : Princeton University Press.

Birch, K., & Muniesa, F. (2020). *Assetization: Turning Things into Assets in Technoscientific Capitalism*. Cambridge : MIT Press.

Boullier, D. (2023). *Propagations : Un nouveau paradigme pour les sciences sociales*. Paris : Armand Colin.

Brooks, F. (1975). *The Mythical Man-Month: Essays on Software Engineering*. Reading, Mass : Addison-Wesley Professional.

Button, G., & Sharrock, W. (1995). The mundane work of writing and reading computer programs. In P. T. Have, & G. Psathas (Eds). *Situated Order: Studies in the Social Organization of Talk and Embodied Activities* (pp. 231-258). Washington : University Press of America.

Denis, J., & Pontille, D. (2025). *The Care of Things: Ethics and Politics of Maintenance*. Cambridge : Polity Press.

Fischer, J. (2025). Hands for AI: Programming in Machine Learning as labor, work, and action. Dans R. V. Burri, H. Göbel & I. Reimers (dir.), *Grounding Digitalization: Technologies, Materialities, and Spaces* (p. 237-254). transcript Verlag. https://www.degruyterbrill.com/document/doi/10.1515/9783839457887-014/html

Flor, N. V., & Hutchins, E. L. (1991). Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance. In J. Koenemann-Belliveau, T. Moher, & S. P. Robertson (Eds). *Empirical Studies of Programmers: Fourth Workshop* (p. 36-62). Norwood : Ablex Publishing Corp.

Goody, J. (1977). *The Domestication of the Savage Mind*. Cambridge University Press.

Hennion, A. (2017). Attachments, you say?... How a concept collectively emerges in one research group, *Journal of Cultural Economy*, *10*(1), 112-121. https://doi.org/10.1080/17530350.2016.1260629

Hutchins, E. (1995). *Cognition in the Wild*. Cambridge : MIT Press.

Janicka, I. (2023). Reinventing the Diplomat: Isabelle Stengers, B. Latour & B. Morizot, *Theory, Culture & Society*, *40*(3), 23-40. https://doi.org/10.1177/02632764221146717

Jaton, F. (2019). « Pardonnez cette platitude » : de l'intérêt des ethnographies de laboratoire pour l'étude des processus algorithmiques. *Zilsel*, *5*(1), 315-339. https://www.cairn.info/revue-zilsel-2019-1-page-315.htm

Jaton, F. (2021). *The Constitution of Algorithms: Ground-Truthing, Programming, Formulating*. Cambridge : MIT Press.

Jaton, F. (2022). Éléments pour une sociologie de l'activité de programmation. *RESET. Recherches en sciences sociales sur Internet*, (11). https://doi.org/10.4000/reset.3829

Jaton, F. (2023). Julie Patarin-Jossec, La fabrique de l'astronaute. Ethnographie terrestre de la station spatiale internationale. *Revue d'anthropologie des connaissances*, *17*(1). https://doi.org/10.4000/rac.29594

Jaton, F. (2025). Examining Algorithms in the Light of their Ground Truth Datasets: Results, Objections, and Avenues of Reflection. *Digital Society*, *4*(2), 61. https://doi.org/10.1007/s44206-025-00197-4

Kling, R. (1996). *Computerization and Controversy: Value Conflicts and Social Choices*, San Diego, Academic Press.

Latour, B. (1986). Visualization and Cognition: Thinking with Eyes and Hands, *Knowledge and Society*, *6*(1), 1-40.

Latour, B. (1987). *Science in Action*. Harvard : Harvard University Press.

Latour, B. (2010). An Attempt at a 'Compositionist Manifesto', *New Literary History*, *41*(3), 471-490. https://www.jstor.org/stable/40983881

Lynd, R. S., & Lynd, H. M. (1959) [1929]. *Middletown: A Study in Modern American Culture*. San Diego : Harcourt Brace Javanovich.

MacKenzie, A., & Monk, S. (2004). From Cards to Code: How Extreme Programming Re-Embodies Programming as a Collective Practice, *Computer Supported Cooperative Work*, *13*(1), 91-117. https://doi.org/10.1023/B:COSU.0000014873.27735.10

Malaby, T. (2011). *Making Virtual Worlds: Linden Lab and Second Life*. New York : Cornell University Press.

Merchant, B. (2025). AI Killed My Job: Tech workers. *Blood in the machine*. https://www.bloodinthemachine.com/p/how-ai-is-killing-jobs-in-the-tech-f39

Mirowski, P. (2012). The Modern Commercialization of Science is a Passel of Ponzi Schemes, *Social Epistemology*, *26*(3-4), 285-310. https://doi.org/10.1080/02691728.2012.697210

Perec, G. (1989). *L'Infra-ordinaire*. Paris : Seuil.

Pütz, O. (2021). Managing exactness and vagueness in computer science work: Programming and self-repair in meetings, *Social Studies of Science*, *51*(6), 938-961. https://doi.org/10.1177/03063127211010972

Rancière, J. (2004). *The Philosopher and His Poor*. Durham : Duke University Press.

Rosenberg, S. (2008). *Dreaming in Code: Two Dozen Programmers, Three Years, 4,732 Bugs, and One Quest for Transcendent Software*. New York : Three Rivers Press.

Rosental, C. (2021). *The Demonstration Society*. Cambridge : MIT Press.

Satariano, A., Mozur, P., Conger, K., *et al.* (2024). Chaos and Confusion: Tech Outage Causes Disruptions Worldwide. *The New York Times*, 19 July. Available at: https://www.nytimes.com/2024/07/19/business/microsoft-outage-cause-azure-crowdstrike.html (accessed 19 February 2025).

Suchman, L. (1995). Making Work Visible. *Communications of the ACM*, *38*(9), 56-64. http://doi.acm.org/10.1145/223248.223263

Ullman, E. (2012a). *Close to the Machine: Technophilia and Its Discontents*. New York : Picador.

Ullman, E. (2012b). *The Bug: A Novel*. New York : Picador.

Vinck, D. (2003). *Everyday Engineering: An Ethnography of Design and Innovation*. Cambridge : MIT Press.

Vinsel, L. (2021). You're doing it wrong: Notes on criticism and technology hype. *Medium*. https://sts-news.medium.com/youre-doing-it-wrong-notes-on-criticism-and-technology-hype-18b08b4307e5 (accessed 1 May 2023).

NOTES

1. Parmi ces rares travaux, l'un des pionniers fut sans aucun doute celui de Nick Flor et Edwin Hutchins (1991) sur l'activité située et distribuée de la maintenance logicielle. Hélas, cette étude – et le formalisme intéressant qu'elle met en place pour étudier les pratiques réelles de programmation, que j'ai récemment tenté de reprendre (Jaton, 2021 ; 2022) – n'a pas été poursuivie de manière systématique, ses deux auteurs s'étant ensuite orientés vers d'autres sujets de recherche. Il convient également de mentionner la contribution ethnométhodologique de Button et Sharrock (1995) sur le travail d'écriture et de lecture de code informatique. Cette recherche, qui reste difficile d'accès, a posé les bases d'une sociologie des activités concrètes de codage, notamment en proposant de ralentir l'analyse et de se concentrer sur les micro-événements tels qu'ils sont vécus dans l'espace de travail des programmeur·euse·s. Cette approche et une partie de ses propositions ont récemment été reprises par Olé Pütz (2021) dans sa description patiente et importante de l'auto-réparation dans le travail informatique. Il convient également de mentionner un article d'Adrian Mackenzie et Simon Monk (2004) qui étudie de manière ethnographique la méthode dite « *Extreme Programming* » (XP), une approche de conception logicielle caractérisée par l'accent mis sur la rationalisation et le travail en binôme, contrairement aux formes hiérarchiques traditionnelles de l'ingénierie logicielle. Mais là encore, si cette approche analytique a donné lieu à des propositions intéressantes – notamment celle selon laquelle l'XP contribue à réincarner les pratiques de programmation individuelles en processus collaboratifs et organisationnels –, elle semble avoir été le résultat d'une collaboration conjoncturelle et non le point de départ d'une ligne de recherche systématique. En termes de monographies, il convient de mentionner le remarquable travail journalistique de Scott Rosenberg (2008), qui a patiemment décrit l'organisation du travail logiciel dans une start-up en vogue de la Silicon Valley dans les années 2000. On peut également citer le travail ethnographique fin et approfondi de Sareeta Amrute (2016) qui, à travers le suivi de travailleur·euse·s indien·ne·s du secteur technologique à Berlin, explore les modalités complexes par lesquelles l'origine ethnique et la classe sociale s'inscrivent et se donnent à voir dans le travail informatique. Enfin, il convient de mentionner le remarquable travail ethnographique de Thomas Malaby (2011), qui a réussi à suivre une partie du développement du monde virtuel Second Life, tout en documentant – et, en fin de compte, en se concentrant sur – les pratiques culturelles de ses utilisateur·rice·s.

2. Ce manque relatif d'études sociales peut s'étendre à l'ensemble des pratiques d'ingénierie industrielle, beaucoup moins étudiées par les praticien·ne·s des STS que les disciplines scientifiques universitaires, notamment en raison des difficultés d'accès (Vinck, 2003).
3. Dans Jaton (2021 : 93-134), j'émets l'hypothèse quelque peu téméraire, mais néanmoins étayée, que cette lacune ethnographique est en partie due à la prééminence d'une forme de computationnalisme au sein des sciences sociales, elle-même due à des processus historiques contingents, liés en particulier à la large diffusion de la notion de « cerveau électronique » grâce notamment aux travaux de John von Neumann et de ses disciples après la Seconde Guerre mondiale.
4. Pour un aperçu plus concis des différentes parties de l'ouvrage, les lecteur·rice·s peuvent se reporter au très bon compte rendu publié dans cette revue l'année dernière (Alauzen, 2025).
5. Il serait toutefois intéressant de documenter l'aspect moyenne tech des big tech, par exemple en examinant le travail des équipes chargées de la maintenance des produits d'entreprise hérités (*legacy corporate products*), comme les différentes versions de Microsoft Office. Il pourrait y avoir beaucoup plus de moyenne tech dans la big tech qu'on ne l'imagine.
6. Hormis quelques indices laissant entendre que des rencontres fortuites ont joué un rôle déterminant (pp. 43-46), le livre reste assez discret sur ce qui a permis à Bialski d'accéder au cœur des pratiques du développement logiciel. C'est regrettable, car ce genre d'astuces professionnels aurait sans doute pu aider les futur·e·s ethnographes cherchant à ouvrir des terrains d'enquête dans le monde du logiciel.
7. Le terme d'actifisation renvoie au terme anglais *assetization*, entendu comme le processus par lequel des choses (par exemple des commodités) sont transformées en actifs, générateurs de rentes.
8. Bien que Bialski ait tendance à privilégier la forme adverbiale *good enough*, notamment pour rester cohérente avec le sous-titre du livre, les formes gérondive et nominale *good enoughing* et *good enoughness* sont en tout point équivalentes, comme elle le souligne dans la note 1 de la page 1. Étant donné que le terme fait principalement référence à des pratiques, je préfère utiliser la forme gérondive – très anglo-saxonne – *good enoughing* dans ce commentaire.
9. Cette dynamique ne devrait en effet pas se limiter au seul développement logiciel, et des processus comparables pourraient bien affecter d'autres contextes d'entreprise dans lesquels les solutions provisoires, l'improvisation itérative ou le « faire avec » finissent par devenir la norme.
10. Dans ses passages consacrés aux pratiques de programmation, Bialski fait souvent référence à la nouvelle *Close to the Machine* de la romancière et ancienne programmeuse Ellen Ullman (2012a). À mon avis, il aurait été bien plus approprié – et moins trompeur – de se référer plutôt au grand roman d'Ullman intitulé *The Bug*, qui raconte l'histoire d'Ethan Levin, un programmeur de la classe moyenne travaillant pour une entreprise de moyenne tech dans les années 1980, aux prises avec un bug récalcitrant d'interface utilisateur (2012b) : sans doute l'une des descriptions les plus convaincantes – fictive mais réaliste – des situations quotidiennes de programmation.

RÉSUMÉS

Grâce à une immersion ethnographique au sein d'une entreprise basée à Berlin, Paula Bialski a accompli l'exploit de décrire le développement logiciel tel qu'il se fait. Quels enseignements cette leçon radicale d'empirisme offre-t-elle à l'étude sociale des technologies informatiques ? Premièrement, qu'il est crucial de ne pas céder trop rapidement aux effets de manche médiatique

entourant les big tech et les start-ups, qui ne constituent que la frange la plus visible et la plus bruyante d'un écosystème bien plus riche et diversifié. Deuxièmement, qu'il est essentiel de se défier des discours conventionnels sur le développement logiciel – qui tendent souvent à répéter, propager et amplifier des idéalisations morales – et de se concentrer plutôt sur la description patiente de pratiques concrètes. Troisièmement, que le développement logiciel en entreprise est façonné par le « *good enoughing* » ; une activité qui vise à définir le point d'arrêt capable de garantir la qualité du logiciel en train d'être construit, mais aussi le temps, l'énergie et l'investissement personnel des développeur·euse·s. Enfin, qu'il reste à trouver des moyens de mieux documenter les situations de programmation informatique, qui pour l'heure, continuent de résister au regard ethnographique.

Gracias a una inmersión etnográfica en una empresa con sede en Berlín, Paula Bialski ha logrado describir el desarrollo de software tal y como se lleva a cabo. ¿Qué enseñanzas ofrece esta radical lección de empirismo al estudio social de las tecnologías informáticas? En primer lugar, que es fundamental no ceder demasiado rápido a los efectos mediáticos que rodean a las grandes tecnológicas y las start-ups, que no son más que la parte más visible y ruidosa de un ecosistema mucho más rico y diverso. En segundo lugar, que es esencial desconfiar de los discursos convencionales sobre el desarrollo de software —que a menudo tienden a repetir, propagar y amplificar idealizaciones morales— y centrarse más bien en la descripción paciente de prácticas concretas. En tercer lugar, que el desarrollo de software en las empresas está marcado por el «good enoughing», una actividad que tiene como objetivo definir el punto de parada capaz de garantizar la calidad del software que se está construyendo, pero también el tiempo, la energía y la inversión personal de los desarrolladores. Por último, que aún quedan por encontrar formas de documentar mejor las situaciones de programación informática, que por el momento siguen resistiéndose a la mirada etnográfica.

Dank einer ethnografischen Immersion in ein Berliner Unternehmen ist es Paula Bialski gelungen, die Softwareentwicklung so zu beschreiben, wie sie tatsächlich stattfindet. Welche Erkenntnisse liefert diese radikale Lektion in Empirie für die soziale Untersuchung von Informationstechnologien? Erstens, dass es entscheidend ist, nicht zu schnell den medialen Effekten rund um Big Tech und Start-ups zu erliegen, die nur den sichtbarsten und lautesten Teil eines viel reichhaltigeren und vielfältigeren Ökosystems ausmachen. Zweitens, dass es wichtig ist, sich gegenüber konventionellen Diskursen über Softwareentwicklung – die oft dazu neigen, moralische Idealisierungen zu wiederholen, zu verbreiten und zu verstärken – kritisch zu verhalten und sich stattdessen auf die geduldige Beschreibung konkreter Praktiken zu konzentrieren. Drittens, dass die Softwareentwicklung in Unternehmen durch „Good Enoughing" geprägt ist; eine Aktivität, die darauf abzielt, den Punkt zu definieren, an dem die Qualität der zu entwickelnden Software gewährleistet ist, aber auch die Zeit, Energie und persönliche Investition der Entwickler*innen. Schließlich müssen noch Wege gefunden werden, um Situationen der Computerprogrammierung besser zu dokumentieren, die sich derzeit noch einer ethnografischen Betrachtung entziehen.

INDEX

Mots-clés : ethnographie, moyenne tech, développement logiciel, good enough, programmation informatique
Palabras claves : etnografía, tecnología media, desarrollo de software, good enough, programación informática
Schlüsselwörter : ethnographie, mittlere technologie, softwareentwicklung, good enough, programmierung

AUTEUR

FLORIAN JATON

Faculté des sciences sociales et politiques, Université de Lausanne & Collège des Humanités, EPFL.
Florian Jaton est enseignant-chercheur à l'Université de Lausanne (Faculté des sciences sociales et politiques) et à l'EPFL (Collège des Humanités). Il a précédemment travaillé à la Donald Bren School of Information and Computer Science à l'Université de Californie, Irvine, au Centre de sociologie de l'innovation à Mines Paris, Université PSL et au Geneva Graduate Institute of International and Development Studies. Ses intérêts de recherche sont la sociologie des algorithmes, la philosophie des mathématiques et l'histoire de l'informatique. Il est l'auteur de The Constitution of Algorithms : Ground-Truthing, Programming, Formulating (MIT Press, 2021 ; en accès libre). ORCID : https://orcid.org/0000-0002-5001-9098
Adresses : Faculté des sciences sociales et politiques, Institut des sciences sociales, 1015 Lausanne, Suisse ; Collège des Humanités, EPFL, 1015 Lausanne, Suisse.
Courriels : florian.jaton@unil.ch ; florian.jaton@epfl.ch