Revue d'anthropologie des connaissances

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

Software actually

Reflections on Paula Bialski's book Middle Tech

Florian Jaton

Electronic version
URL: https://journals.openedition.org/rac/41398
DOI: 10.4000/15s7r
ISSN: 1760-5393

This article is a translation of:
Logiciel tel quel - URL : https://journals.openedition.org/rac/41171 [fr]

Publisher
Société d'Anthropologie des Connaissances

Provided by Bibliothèque cantonale et universitaire Lausanne

Electronic reference
Florian Jaton, "Software actually", *Revue d'anthropologie des connaissances* [Online], 20-1 | 2026, Online since 09 February 2026, connection on 02 March 2026. URL: http://journals.openedition.org/rac/41398 ; DOI: https://doi.org/10.4000/15s7r

Software actually

Reflections on Paula Bialski's book Middle Tech

Florian Jaton

Introduction

1 Is there any flatter platitude than to say that we are surrounded by software? Day and night, our courses of action – whether we like it or not – repeatedly cross the path of digital devices whose agency derives from computer-readable lists of instructions. What was already true in the 1990s (Kling, 1996) has become obviously truer with the global advent of mobile computing, whose software applications contribute to our life trajectories, to say the least. It is not love, but *software actually* that is all around.

2 Hence a mystery that remains unfathomable, at least for Science & Technology Studies (STS) practitioners who, like me, are interested in the social shaping of computer science and technology: Why are there so few detailed empirical descriptions of the practical processes by which software comes to be constructed? Once we leave aside the massive literature in *software management* which focuses on how things should be, not on how they unfold (Suchman, 1995); that in *software psychology* which remains confined to artificial settings where variables are duly controlled, therefore preventing itself from documenting what happens in the wild (Hutchins, 1995); and even – to my great dismay – that in *software anthropology* which tends to limit itself to the cultural habits of techies and hackers communities, without really describing the concrete software development work they are passionate about (Jaton, 2019); once we leave aside – in sum – all these interesting massive literatures that yet are *not* interested in documenting the social shaping of software *in situ*, we realize that empirical descriptions can be counted on the fingers of one hand (or maybe two).[1] And of these rare and precious descriptions, virtually *none* has resulted in a careful authoritative monograph released by a conscientious academic publisher whose reviewing process can be trusted. If only more STS efforts were devoted to meticulously describing the practices of those who make computers compute in desired ways! We would know a lot more about our attachments, and therefore about ourselves (Hennion, 2017).[2]

3 I tried elsewhere to get to the bottom of this mystery, without much success alas.[3] But today is time to celebrate rather than complain, for as Paula Bialski – a professor of Digital Sociology at the University of St. Gallen, Switzerland – has succeeded in completing one of the very first long-standing ethnographic studies of everyday software development work. Through a quite unique and demanding inquiry into a navigation software company based in Berlin, Bialski (2024) has accomplished the daring feat of describing software development work as it is, without much cyber-modernist frill. And even if the text is not without its problems (which I will discuss at the end of this commentary), it is worth remembering that a great ethnography is not an ethnography without blame: it is an ethnography without fear (Jaton, 2023). And it is this much-needed fearlessness – from which derives a series of strong propositions that I'm now about to summarize – that makes *Middle Tech* a major STS work; a lesson in empiricism, in short, that I think it's crucial to point out to the community, just in case they missed it.[4]

Medium tech, actual tech

4 If the book is so striking, it's not least because it marks a radical return to reality. Far from being hypnotized by the rhetoric of the tech giants, often perceived as the undisputed leaders of software development, Bialski reminds us that they are in fact only the most visible part of a much larger, and more interesting, ecosystem. Indeed, the vast majority of the software packages that contribute to our living infrastructures – from banking systems to transport networks, from public administrations to hospitals – is *not* the work of the Silicon Valley behemoths, as Bialski neatly sums up in this concise formula: "We are so often confronted with stories from the Silicon Valley Big Tech that we forget that most of our digital infrastructure isn't actually made by these companies." (p. 42).

5 Exit then – for the moment at least – the big tech of Google, Microsoft, Amazon and co.: software life is elsewhere.[5] Maybe in start-ups, those flagships of the new digital economy, and their supposed ability to disrupt established markets? Nothing is less certain, as their business model typically revolves around being acquired by larger companies – ideally, big tech giants. Their primary motivation, then, is generally to raise funds to demonstrate compelling proofs of concept (Mirowski, 2012; Rosental, 2021), a feature that weighs heavily on the resulting software, which is rarely optimized, secure, or even reliable, and therefore ultimately less widely distributed (or so we hope). Again, Bialski sums up very well this crucial difference between start-up and corporate software development:

> During my fieldwork, I heard countless stories from software developers about their quick-and-dirty start-up times, when software development was about "hacking together" ideas and getting them out quickly, rather than slowing down to maintain a cohesive system. Slowing down and creating cohesion is often favored [outside of start-ups] – not simply to make code beautiful but rather to maintain the long-term stability of their system, to reduce the chance of potential bugs, and to help those who will maintain the code in the future. (p. 58)

6 If we want to document software development in its ordinary, and therefore pervasive, form, we need to be wary of big tech companies and start-ups: their media/political weight is largely disproportionate to their actual role in the production and maintenance of software infrastructures. So where to look at? One of the key lessons of

the book – cutting against the grain of hype that social sciences are not immune to (Vinsel, 2021) – is the need to inquire into what Bialski calls *medium tech*: mid-sized companies (typically of a few thousand employees) that, away from the spotlight, quietly design, maintain, and adapt the software essential to the digital fabric of the collective world (pp. 26-32).

7 With this well-informed perspective – further inspired by Lynd and Lynd's *Middletown Studies* (1959 [1929]) – Bialski managed to gain a much enviable access to a medium tech company, which she named MiddleTech.[6] Based in Berlin, this company plays a key role in the development of navigation and routing software, notably for German car manufacturers. During six months of total immersion, complemented by return trips over almost two years, Bialski meticulously documented the day-to-day work of the different technical teams of MiddleTech: front-end developers in charge of navigation applications; back-end developers who design the operating system, databases and algorithms; data scientists who conduct experiments on vast datasets; and the so-called DevOps teams (Development Operations) responsible for the software infrastructure facilitating testing, and bug detection; all of them interesting participants in the *infra-ordinary* (Perec, 1989) office life of contemporary software in the making.

8 At first, Bialski's emphasis on the 'medium,' the 'mundane,' and the 'average' (pp. 24-25) could give the impression to be tinged with a romantic nostalgia for some authentic vestige of the twentieth-century digital industry. But it quickly becomes clear that this is not the case, especially since, like most other medium-sized companies still operating today, MiddleTech has undergone its own series of mergers, acquisitions, and rebranding to reach its current form. It could also have disappeared, as many medium-sized companies do (pp. 26-27). But it is still very much alive and, like many others across the five continents, its thousands of employees continue their day-to-day work to develop, and above all maintain, the software embedded in our digital infrastructures.

9 This brings us to the crucial question of *software maintenance*, which derives from Bialski's salutary look at medium tech. Spotting the many subtle forms of technological care work, she crucially notes that the daily tasks of corporate computer scientists and technicians revolve more around maintaining and adapting aging software than pursuing innovation:

> [L]ife with technology is usually far removed from the cutting edges of invention and innovation and is instead devoted to keeping things the same. [...] [O]ur software and our software companies are aging. As our software ages, our software projects become more and more complex, evolving into multilayered beasts ... Much of the software we use today is built on years and years of effort by software developers who have managed to patch together a project to make it work. (pp. 5-6)

10 No romanticism, then, but methodological rigor, which immediately places the book at the heart of today's most pressing issues, starting with the need to make our aging software technologies last. And this necessary emphasis on software maintenance is all the more interesting in that it is also akin to caring for *assets* as part of *capitalization efforts*:

> At MiddleTech, this digital asset is mapping and navigation software, called the map engine, which they sold and continue to sell to third-party businesses in need of maps in their products (such as a car or another app that needs a map feature). Due to this prolific embeddedness of the company's software, the company is able to coast on its revenues from its asset, which was built years ago... [More generally]

> Medium Tech companies do not make as much revenue as their Big Tech counterparts, and their scope for reinvestment in research or innovation is quite limited. With this limitation, the software work at MiddleTech was dedicated mostly to maintenance and repair. (pp. 34-35)

11 Bialski's look at the medium tech that irrigates digital infrastructures brings together studies on maintenance (Denis & Pontille, 2025) and those on assetization (Birch & Muniesa, 2020) around a new object: day-to-day software work. This emphasis on specific but quite ordinary activities illuminates, and therefore documents, one of the nerve centers of the socio-technical reconfiguration of contemporary capitalism. Does that sound a bit exaggerated? Just recall, then, the chaos sparked by a tiny bug in a security update of the medium tech company CrowdStrike in July 2024. A single maintenance error in CrowdStrike's most valuable digital asset – implemented in Microsoft Windows' kernel – and the world came to a halt for a couple hours (Satariano *et al.*, 2024). Infra-ordinary, software, medium tech, maintenance, assetization: five terms of a set that concerns us all.

Interviews are not enough

12 Bialski thus skillfully avoids the pitfall of media over-attention to big tech and start-ups, which enables her, in turn, to underline the centrality of software maintenance operations, which are also capitalization efforts. But she also manages to avoid a second, equally insidious pitfall: that of relying solely on interviews to understand the work of software developers.

13 Though a classic topic in qualitative social sciences – extensively debated for decades (e.g., Becker & Geer, 1957, 1958; Atkinson & Coffey, 2004) – it is certainly worth revisiting, as it constitutes a key condition of this remarkable inquiry. For Bialski could quite comfortably have based her investigation on a series of interviews. In fact, she began by doing so, notably with Silicon Valley techies working for Meta, the Wikimedia Organization, or as entrepreneurs (pp. 2-3). Unsurprisingly, these developers spoke of sleepless nights spent perfecting software that was supposed to be 'revolutionary,' in a logic of disruption and absolute performance. A widespread rhetoric – omnipresent in managerial discourses, which operate as discursive resources directly mobilizable by the interviewees – that gives the impression that continuous improvement and the quest for perfection are the main driving forces behind software development. As Bialski notes:

> The Silicon Valley techies I encountered seemed to believe that technology had to be great, and that work on technology had to be hard and sweaty. [...] What I found striking was the repetitive narrative that software developers were dedicated to working into the late hours perfecting something 'outta this world.' Software was not just patched together to run, occasionally break down, and be maintained; it was meant to run, disrupt, and innovate all in one go. (pp. 2-3)

14 But after just a few days of fieldwork, it becomes clear that the ordinary, concrete reality is very different. While the rhetoric associated with excellence and constant improvement does exist and circulate, it finds very little practical expression *in situ:*

> I discovered throughout my fieldwork that while these metrics, methods, and modes of excellence and improvement are present in the MiddleTech office culture, the reality is different. On a discursive level, corporate software environments can be understood as factories of so-called technological acceleration, where

> technology is constantly updated to improve and strive for excellence. Yet in the everyday, often mundane reality, software developers are more informed by good-enough principles and practices. (p. 11)

15 I will come back to the central notion of 'good enough', the book's great empirical observation and theoretical proposition, in the next section. But for now, it is worth highlighting the methodological lesson – obvious only in hindsight – of being wary of conventional discourses on software development, which more often than not take up, propagate (Boullier, 2023), and thus amplify moral idealizations. A parallel can be drawn here with the social studies of science and their past confrontation with the philosophy of science: just as epistemology has long offered a theoretical and normative vision of science far removed from researchers' actual practices (Latour, 1987: 2-17), software management literature tends to produce an abstract and idealized account of software development, shaped by political and organizational considerations rather than the practical reality of developers' day-to-day work.

16 However, it is important to clarify that Bialski's book is not a critical debunking exercise meant to prove that actors are trapped in a form of *illusio*. Rather, the point is much simpler. First, when responding to an interviewer they know little (or nothing) about, actors often provide the answers they believe are expected. Second, these actors care enough about these perceived expectations to further propagate them. Bialski captures this idea nicely at the very start of the book:

> [W]orkers reject notions of excellence in practice, but I'd like to highlight that a hegemonic excellence discourse does exist *in theory*. Corporate software companies, like many corporate environments, propagate an ideology of excellence and improvement, both in relation to the software product they are building and regarding the type of work that goes into building a software product. (p. 7. Italics in the original)

17 So the book successfully overcomes quite a few hurdles, if only to prepare the ground for a solid inquiry into software development actual work (something virtually unprecedented, as we should remember). But what of the actual results of this quite unique ethnographic undertaking?

Good enoughing, or where the salvation of privileged corporate software development work lies

18 The main proposition of the book is that corporate software development work – at least within MiddleTech (and very probably in many other medium tech companies too) – is shaped by an activity that Bialski calls *good enoughing*.[7] While the author does not define precisely what she means by activity – a term that sometimes seems interchangeable with that of culture – we can infer that it is a set of practices oriented towards the same kind of accomplishment. But what is this accomplishment in the case of good enoughing? Rough answer: the ability to adequately determine where to stop in the development process. In this sense, the developers studied by Bialski appear to be true Aristotelians, without necessarily being aware of it: for them too, the *series must stop somewhere and not be infinite.* But there remains the delicate question of *how* to set this limit without compromising either the quality of the software under construction or the developer's own investment in the process. This is the subtlety of good enoughing, which cannot be equated with a form of laziness, as Bialski points out

clearly on the very first page of the book: “My point throughout this book is that achieving good enoughness is an incredibly complex and interesting endeavor.”

19 While good enoughing may not be an activity specific to software development work, it is central to it today, notably due to several changes that have recently deeply affected software materiality. The first change is what is commonly referred to as *software-as-a-service*, a distribution model that coincided with the rise of the Internet in the 2000s. Before the era of software-as-a-service, software was mostly distributed as a physical product (*shrink-wrapped software*), requiring manual installation and updates by the user. With the rise of the Internet in the 2000s, the software-as-a-service model enabled software to be accessed online, via dedicated platforms. This granted developers continuous control over their applications, enabling them to ‘push’ software updates in near real time, with just a few clicks (p. 69).

20 Another recent change is related to the rise of cloud storage. Instead of being stored locally on in-house machines, code composed by developers is now hosted on remote servers, often via cloud service providers. This transition has brought greater security, notably by freeing software from the constraints of local physical infrastructures. However, it has also engendered a form of hyper-instability, where software is no longer a fixed entity, but rather a dynamic set of requests between interfaces.

21 These changes have led to the gradual formation of an *update culture* (or habit, p. 71) within corporate software development, which encourages software to be seen as a continuous project, with no real final release. And at a practical level, this habit tends to normalize iterative development cycles, where frequent updates replace stable, definitive versions. This encourages, in turn, an approach where software is constantly corrected according to user or managerial feedback, adding an extra layer of unpredictability. And while software development has a long history of failure and disappointment (Brooks, 1975), these recent changes seem to have increased the possibility of untimely problems, which in turn suggests a need for *caution*:

> Software-as-a-service, cloud storage, and the update culture that have resulted from these changes write failure and iteration into the programmer’s work culture. This can cause stuff to go wrong in a software system, and putting stuff back together when it does go wrong also takes a lot of energy, various forms of knowledge, and time. (p. 71)
>
> Working with software means that different heterogeneous forms of knowledge are in constant competition with one another, and the code base often expands but is not always deleted, building convoluted, codependent legacy systems that are also challenging to figure out. These two factors lead to a particular work culture of ‘figuring out stuff,’ compromise, and confusion. (p. 69)

22 This confusion fosters a degree of caution – if not outright conservatism – in corporate software development, also affected by the growing presence of *legacy code* (p. 86), understood as code inherited from past projects or other developers, whether from an acquisition, a previous team, or earlier stages of development. This inherited code – which cloud storage makes it possible to keep at a lower cost – is ambivalent, because it can constitute a reassuring, tried-and-tested, and generally functional base. But legacy code can also be a source of confusion and complexity, as it is sometimes written in different (sometimes obsolete) languages or has been modified many times, making its structure difficult to understand. Some developers even see legacy code as a burden, a kind of “monster” that becomes tedious to maintain and makes it difficult to add new features (pp. 87-88).

23 In short, a corporate software developer is *never alone*; he or she is surrounded by a myriad of current and former colleagues who have produced a myriad of lines of more or less obscure and monstruous code. The practical problem facing many medium tech corporate software developers (and probably others too) on a daily basis is then: how best to compose with legacy code, whose proliferation has accelerated with the rise of cloud-based software development? And this is precisely where good enoughing comes into play, in a quasi-functional way, as a delicate reviewing activity trying to cope with multiple constraints.

24 This constrained reviewing activity permeates all development work at MiddleTech. But nowhere is it more important, and more visible, than during the infamous *feature-complete days*, when multiple features have to be finalized and integrated into the main code. To illustrate this, Biaslki uses the example of a team she followed for many weeks working on routing functionalities for electric cars. This team – like most of the hundreds of other teams within MiddleTech – is inspired by the Scrum methodology (more on this later) and structures its work in *sprints*, which begin with a planning meeting defining tasks in the form of *tickets*. Each ticket corresponds to a specific development step, such as – in this case – integrating a charging station library into the routing database. Once assigned, the task is developed individually, then submitted to a code review on Gerrit (a Web-based code-collaboration tool), where each contribution is evaluated and rated by the other team members, according to more or less arbitrary criteria (which can be subjects of debates, cf. p. 93). And all this organization is itself geared towards feature-complete days, those crucial days marking the finalization and simultaneous integration of multiple sprints-made-functionalities into the main code. But these are also the days when technical contingencies explode: merge conflicts, critical bugs, incomplete or unstable functionalities are legion. Faced with this avalanche of issues, integration teams enter a phase known as *firefighting*, where they actively scrutinize code mergers and improvise rapid solutions, most of times in the form of temporary *hacks* (quick, unstructured ways of writing code). As Bialski observed, in these intense but common situations:

> [Developers] are not focusing their attention only on creating awesome software but are trying to keep stuff from going completely wrong or trying not to let the whole house burn to the ground, so to speak. Being just good enough to survive in the face of a huge fire is an achievement. In this culture of keeping stuff together, a development team understands that they cannot deliver perfect software and becomes satisfied with code that is good enough. (p. 96)

25 But good enoughing does not just concern programmers trying to keep alive software made up of disparate elements that all seek to go in their own direction (thus generating errors); it also concerns managers and the management tools they mobilize. This is especially salient, for example, with the Scrum methodology, already mentioned above and considered in detail in chapter 4 of the book (pp. 99-131).

26 As a so-called agile method, Scrum aims to organize developers' work around short, iterative cycles (*sprints*). While this approach offers developers a genuine degree of autonomy and flexibility, it also constitutes a managerial control tool, enabling team productivity to be measured and reported to decision-making and strategic bodies. However, depending on the constraints of the moment, the version of Scrum in place at MiddleTech seems to leave some room for negotiation, thus informally integrating some aspects of good enoughing. Yet, in the end, these pragmatic adjustments tend to

be rephrased in the language of excellence, continuous improvement and innovation, thus reinscribing these negotiations in traditional managerial rhetoric.

27 In this sense, if managerial tools such as Scrum endure, it may not be simply because they make software development work more effective or innovative. Rather, so-called agile methodologies such as Scrum offer managers (who are often former developers) a framework that enables them to remain aligned with the idealized cyber-modernist principles of software engineering, while leaving developers a certain room for manoeuvre. In short, the logic of good enough appears to be loosely embedded in the latest 'agile' managerial frameworks, which aim to balance managerial control requirements with evaluative flexibility for developers, allowing them to compose with evolving needs without entirely challenging the official, traditional, and 'off-shore' norms of software engineering. As Bialski notes:

> [T]his narrative of excellence and top performance permeates the corporate software environment, and methods like Scrum are a manifestation of this narrative. Scrum presents the workers with sets of durable schemes, stories, rituals, and routines that guide them, enforcing constant transparency with the goal of reaching peak performance. These methodologies thus serve a number of purposes, which are at odds with the intended purpose of Scrum and other methods: They define the identity of software developers (as those whose work as well as the machines they work with cannot be "tamed" by a method) and help define the objects of their care (software comes first, not customers, users, or peak performance). In practice, methods like Scrum give developers rituals and daily routines that they thus only partially adhere to (in a good-enough way) in order to appease their management while at the same time reproducing a culture of unpredictability and care for their software. (p. 131).

28 But if good enoughing serves as a way for management to save face (because it somewhat adapts software development practice to its idealized conception), for developers to maintain a form of control over their work (because it allows them to define switch-on and switch-off moments), and ultimately as a means of composing with the prevalence of legacy code (itself derived from recent sociotechnical changes in software materialities), Bialski reminds us that, in the end, this is still a *privilege* afforded to certain workers and companies. Those who can afford to good enough typically enjoy safer working environments and greater job security – conditions that are far from trivial. Developers in outsourced coding farms, for example, often lack the flexibility to embrace good enoughing, as they must meet strict deadlines and endure job insecurity. Likewise, only companies with established, revenue-generating software assets to maintain – such as MiddleTech, Microsoft, Crowstrike – can afford to adopt a good-enough approach. These companies, having built their core assets in the past, now enjoy the financial stability to take on – and almost institutionalize – part of the concrete dimension of software development. In contrast, smaller companies struggling to gain traction cannot afford such a luxury. As Bialski sums it up well:

> Being a good-enough company like MiddleTech means also supporting an inequality in work speeds and demands, allowing some people to sit back and opt out of hyperproductivity while cruising on the unrecognized labor of other software developers and service workers. (p. 16-17)

29 The activity of good enoughing is thus fundamentally *ambivalent*: it is both essential to the proper appreciation of the software mode of existence, but also – and at the same time – contributes to the maintenance of the inequalities that ensure its perpetuation.

Biaslki has succeeded in making this fascinating anthropological phenomenon – which may go beyond software development[8] – more intelligible: quite a feat!

A sticky paradox: where are the situated programming practices?

30 The book is sprinkled with many other interesting insights, such as what Bialski terms the 'myth of knowing' (i.e., the tendency for developers to feign expertise in technical areas beyond their actual knowledge; pp. 79-83) – or the creative and often whimsical methods used to estimate project timelines (which frequently rely on the T-shirt size scale! pp. 83-85). However, what constitutes the text's most notable shortcoming is probably worth mentioning here as well: the striking absence of detailed accounts of computer programming in action. The elephant in the room: While computer code is at the heart of the discussions, uncertainties, joys, and challenges experienced by MiddleTech employees, there is almost no mention of the moments during which this code is manually inscribed.

31 I say "almost" because this type of situation is briefly addressed on a few pages at the beginning of the book (pp. 51-54). The discussion revolves around an abstract state called 'flow,' which programmers are said to enter while deeply working on their code, those intricate numbered lists of symbols. This flow state is described as a deeply intuitive and immersive synchronization between the programmer and their machine, often associated with outward signs of retreat, such as wearing large headphones to isolate oneself from the other people in the open space office. Borrowing the terminology of novelist and former programmer Ellen Ullman – frequently cited in the book – flow evokes a kind of symbiotic *closeness* between a human and a computer.[9]

32 If this short passage strikes me as paradoxical, it is not least because it derives from interviews, therefore echoing the conventional discourse on software development – a discourse that is precisely the focus of Bialski's criticism, and even underpins the relevance of her immensely important ethnographic approach. In other words, the almost mystical – and quite abstract – aura surrounding programming situations stems directly from interviews imbued with ready-made formulas, easily mobilized because widely disseminated and socially accepted. Yet why not go beyond these pre-established narratives? For although they contain elements that are certainly relevant (and that should be respected), they ignore quite completely the actual work environment of developers, with its massive monitors displaying a multitude of interactive windows forming a genuine digital *workbench* linked to many others via different discussion spaces (e.g., forums, chats). Beyond Bialski's initial omission, an even more puzzling and pervasive oversight persists: the tendency to consider computer programming in isolation, detached from its inherently, and obvious, hyper-graphical, -material, and -situated dimension (Goody, 1977; Latour, 1986).

33 The fact that even the most thorough and sophisticated ethnographic study of software development work struggles to capture computer programming in action – relying instead on interviews that reinforce abstract cyber-modernist stereotypes – gives me pause. Perhaps such an endeavor is simply impossible? And yet, as recently indicated by Pütz (2021), Fischer (2025), and I (Jaton, 2021: 135-195; 2025), there do seem to be ways forward, even if they require a great dose of obstination. But a more fine-grained

ethnographic focus on the gestures, tools, and digital spatialities of software developers would undoubtedly help move beyond our conventional narratives, offering a richer, more embodied understanding of the computer programming work on which we increasingly rely. In doing so, this would-be *micro-sociology of computer programming* (Jaton, 2022) could even open the door to meaningful socio-ergonomic insights, much like Edwin Hutchins' foundational work on speed signaling in airliner cockpits (1995).

Conclusion: Ethnography is a martial and equilibrist (and thus diplomatic) art

34 Does the failure to account for computer programming situations beyond idealized discourses mean that Bialski's ethnographic endeavor is not a total success? But one does not see that ethnographers are destined to succeed totally, any more than Louis Pasteur or Marie Curie. Carefully documenting a substantial part of the infra-ordinary life of corporate software development is such a significant contribution to the social study of computer science and technology that it would be more than unfair to fault Biaslki for this shortcoming (though I still believe it is an important one). This would be all the more a mistake, given how effectively the book reminds us of the centrality of ethnography to the laborious composition of a common world.

35 For Bialski's work makes it clear that software development is not about optimization. It is, at its core, about *problematization*. It is a problem-oriented process, where the inherent frictions, often arising from the diverse specifications of a project's many components, play a crucial role in shaping the solidity *and* fragility of the software packages embedded in our living infrastructures. These challenges give rise to specific regimes of activity, such as good enoughing which enables developers to collectively stand the inevitable social complexities of their profession.

36 As a result, cyber-modernist dreams of optimizing software development and drastically reducing its errors – for example, by extensively relying on confident code generated automatically by large language models trained on programmers' forums such as Stack Overflow – are doomed to failure. Indeed, it is precisely the many errors and inaccuracies that fuel software development and organize its production. Bialski's meticulous work thus invites us to rethink software development, not in terms of functionalities and features, but rather through the lens of errors, bugs, and the social practices designed to compose with them, such as good enoughing.

37 These key insights – while not without precedents – are presented here with remarkable rigor and clarity, making them especially relevant today, as programmers frequently appear on lists of professions allegedly at risk of being replaced by artificial intelligence programs. These far-fetched statements are obviously to be seen in the light of an attention economy fond of sensationalism, but they can nonetheless have real effects, not least due to the cynical opportunity they offer to disguise classic austerity measures as cutting-edge digital innovations (Merchant, 2025). In this context, Bialski's realistic descriptions can serve as *counterfires*, offering workers a tangible means of highlighting – and advocating for – the inherently non-optimizable aspects of their craft.

38 For now, however, the self-defensive potential of Bialski's propositions seems to collide with many developers' enduring attachment to the dominant modernist vision of their

work centered on excellence, continuous improvement, and efficient problem-solving. The book's final pages illustrate this tension, recounting some MiddleTech developers' mixed reactions when, on a Berlin terrace, Bialski reveals that her investigation account will focus on good enoughing (pp. 157-160). But this is precisely where a second, immensely delicate challenge of ethnographic studies in science and technology emerges, beyond the meticulous conduct of the inquiry itself: the further refinement of propositions in collaboration with those who made the research possible.

39 For Bialski's propositions are, in part, an act of unveiling: actual software development practices diverge significantly from the 'official' discourses that claim to define them. Yet stopping at this (soft) revelation would be irresponsible, as it would cast the ethnographer in the role of a sociologist-queen (Rancière, 2004: 165-202). It is therefore crucial to push the inquiry further, assessing whether the proposed alternative discourse is acceptable to practitioners. Interestingly, the book suggests that it is not entirely (p. 160). Indeed, certain values embedded in the 'official' cyber-modernist discourse – though potentially dangerous, as they risk fueling the very forces that oppose software development or even hasten its dismantling – continue to hold meaning. This raises a fundamental *compositionist* question (Latour, 2010), one that calls for institutional redefinition: Would it be both politically appropriate and methodologically sound to weave certain cyber-modernist ideals into empirical descriptions, not as a concession, but as a means of redirecting the discourse on software toward the concrete realities of software development work, to better preserve it?

40 Clearly, this is a tightrope act that goes beyond Bialski's book. But her work does have the immense merit of having successfully laid the foundations for *diplomatic* efforts (Janicka, 2023) over the values most likely to preserve the practice of software development. And, by the same token, her work has the even greater merit of reminding us that a great ethnography is not just one that poses, but also one that demands.

BIBLIOGRAPHY

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. In 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”. In 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: 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. Among these rare works, one of the pioneers was undoubtedly that of Flor and Hutchins (1991) on the situated and distributed activity of software maintenance. Alas, this study – and the interesting formalism it sets up to investigate actual programming practices, that I recently tried to take up (Jaton, 2021; 2022) – has not been systematically pursued, its two authors having subsequently branched off into other research topics. One should also mention the ethnomethodological contribution of Button and Sharrock (1995) on the work of writing and reading computer code. This investigation, which remains difficult to access, laid the foundations for a sociology of actual coding activities, notably by proposing to slow down the analysis and focus on micro-events as experienced in the programmers' workspace. This approach and part of its propositions have recently been taken up by Pütz (2021) in his patient and important description of self-repair in computer science work. It is also worth mentioning a paper by Mackenzie and Monk (2004) that ethnographically studies so-called Extreme Programming (XP), a software design approach characterized by its emphasis on streamlining and pair-working, in contrast to traditional hierarchical forms of software engineering. But here again, while this analytical approach led to some noteworthy propositions – notably that XP contributes to reincarnating individual programming practices as collaborative and organizational processes – it seems to have been the result of a conjunctural collaboration and not the starting point of a systematic line of research. In terms of monographs, it is worth mentioning the remarkable journalistic work of Rosenberg (2008), who patiently described the organization of software work in a trendy Silicon Valley start-up in the 2000s. There is also the much-refined ethnographic work of Amrute (2016) who, by following Indian tech workers in Berlin, examines the intricate ways in which race and class are encoded into IT work practices. Finally, it is worth mentioning the remarkable ethnographic work of Malaby (2011), who managed to follow part of the development of the virtual world Second Life, while also documenting – and, *in fine*, focusing on – the cultural practices of its users.

2. This relative lack of social studies may be extended to industrial engineering practices as a whole, far less investigated by STSers than academic scientific disciplines, not least due to challenges of access (Vinck, 2003).

3. In Jaton (2021: 93-134), I make the somewhat wild but nonetheless empirically-supported hypothesis that this ethnographic blind spot is due in part to the pre-eminence of a form of computationalism within the social sciences, itself due to contingent historical processes, linked in particular to the wide distribution of the notion of ‘electronic brain’ via the work of John von Neumann and his disciples in the post-World War II era.

4. For a more concise overview of the book’s various parts, readers may wish to refer to the refined review published in this journal last year (Alauzen, 2025).

5. But it would be interesting to document the medium tech aspect of big tech, for instance by examining the work of teams responsible for maintaining legacy corporate products, like the various versions of Microsoft Office. There could be far more medium tech within big tech than one might expect.

6. Aside from a few hints suggesting that chance encounters were crucial (pp. 43-46), the book remains somewhat reserved about what enabled Bialski’s remarkable access to the core of software development in practice. This is a pity, because this kind of tricks of the trade could have helped future ethnographers in their attempts to open software-related field sites.

7. While Bialski tends to favor the adverbial form ‘good enough’ – not least to remain consistent with the book’s subtitle – the gerund and nominal forms ‘good enoughing’ and ‘good enoughness’ are equivalent, as she points out in note 1 on page 1. As the term refers mainly to practices, I prefer to use the gerund form ‘good enoughing’ in this commentary.

8. This dynamic is indeed unlikely to be confined to software development alone, and comparable processes may well affect other corporate settings in which provisional solutions, iterative improvisation, or 'making do' become normalized.
9. In her passages on programming practices, Bialski often refers to the short story *Close to the Machine* by novelist and former programmer Ellen Ullman (2012a). In my opinion, it would have been far more appropriate – and less misleading – to instead closely refer to Ullman's grand novel entitled *The Bug*, which follows the story of Ethan Levin, a middle-class programmer working for a Medium Tech company during the 1980s, grappling with a recalcitrant User Interface bug (2012b): arguably one of the most compelling depictions – fictional yet entirely realistic – of everyday, mundane programming situations.

ABSTRACTS

Through an in-depth ethnographic inquiry into a navigation software company based in Berlin, Paula Bialski has accomplished the daring feat of describing corporate software development in action. What can we learn from this radical lesson of empiricism for the further social study of computer science and technology? First, that it is crucial not to succumb too quickly to the sirens of big tech and start-up hype, both of which represent only the most visible and noisy part of a much richer and more diverse ecosystem. Second, that it is essential to be wary of conventional discourses on software development – which often repeat, propagate, and therefore amplify moral idealizations – and to focus instead on patiently describing concrete practices. Third, that corporate software development is shaped by 'good enoughing,' understood as the practice of deliberately defining an appropriate stopping point that maintains the software's quality while also respecting the developer's time, energy, and personal investment in the work. Finally, that it remains to find ways of documenting computer programming situations, which continue to resist the ethnographic gaze.

INDEX

Keywords: ethnography, medium tech, software development, good enough, computer programming

AUTHOR

FLORIAN JATON

Faculty of Social and Political Sciences, University of Lausanne & College of Humanities, EPFL
Florian Jaton is a researcher and lecturer at the University of Lausanne (Faculty of Social and Political Sciences) and at EPFL (College of Humanities). He previously worked at the Donald Bren School of Information and Computer Science at the University of California, Irvine, at the Centre de sociologie de l'innovation at Mines Paris, PSL University, and at the Geneva Graduate Institute of International and Development Studies. His research interests are the sociology of algorithms, the philosophy of mathematics, and the history of computing. He is the author of the book The

Constitution of Algorithms: Ground-Truthing, Programming, Formulating (MIT Press, 2021; Open Access). ORCID: https://orcid.org/0000-0002-5001-9098
Addresses: Faculty of Social and Political Sciences, Institute of Social Sciences, University of Lausanne, 1015 Lausanne, Switzerland; College of Humanities, EPFL, 1015 Lausanne, Switzerland.
Emails: florian.jaton@unil.ch; florian.jaton@epfl.ch