RESET
Recherches en sciences sociales sur Internet

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

Writing code, making software

Gabriel Alcaras and Antoine Larribeau

Electronic version
URL: https://journals.openedition.org/reset/3944
DOI: 10.4000/reset.3944
ISSN: 2264-6221

This article is a translation of:
Codes. L'informatique comme elle s'écrit - URL : https://journals.openedition.org/reset/3914 [fr]

Publisher
Association Recherches en sciences sociales sur Internet

Electronic reference
Gabriel Alcaras and Antoine Larribeau, "Writing code, making software", *RESET* [Online], 11 | 2022, Online since 22 April 2022, connection on 11 October 2024. URL: http://journals.openedition.org/reset/3944 ; DOI: https://doi.org/10.4000/reset.3944

Writing code, making software

Gabriel Alcaras and Antoine Larribeau

1 How is the software we use every day created? Who builds the informational infrastructure of our contemporary societies? What is the digital world made of? These questions can and do receive many answers, sometimes idealized, sometimes disenchanted. Whatever the tone of the discourse, one thing is taken for granted: software is efficient machinery and produces effects on the social world. The growing debates on algorithms, especially concerning their power and opacity, perfectly illustrate this particular logic of interpretation of the computer world. Although studies devoted to the appropriation of techniques warn against a deterministic view of technologies and document "the possibilities of autonomy and emancipation for individuals and groups"[1] (Proulx, 2015), digital infrastructures retain their aura of fascinating devices, at once incredibly complex and perfectly ordered.

2 To question this illusion of order and efficiency, one possible strategy is to enter the heart of the system: "from the outside, one is struck by the marvelous arrangement of the elements, well-aligned with each other, harmoniously united; from the inside, one discovers the twisted elements, the creases, the blockages, the rough edges" (Dodier, 1995, p. 5). But what does "inside" mean when the infrastructures are so vast and so diverse? Should we look at buildings, circuit boards, servers, network cables, databases, programs, platforms, or a host of other objects — not to mention the people who create, maintain, and use them? Making a choice is not easy, especially in the face of the polysemy of the term digital, whose uses and definitions vary greatly according to context (Moatti, 2012; Drot-Delange and Bruillard, 2012; Baron, 2018).

3 This special issue of RESET aims to encourage the social sciences to explore a path that is seldom taken — software. By this term, we mean both the scientific discipline and the industrial engineering that participate in the production of our digital infrastructures and whose resource is code, i.e. text intended to be executed by a computer[2]. Studying software as it is written is therefore a particularly engaging way to understand the construction of software infrastructures from the inside.

1. Programming or coding?

4 Recently, the term "program" has been used to address software in all its globality and ambivalence (Méadel and Sire, 2017), from code to uses, from interfaces (Galloway, 2012) to APIs[3] (Ermoshina, 2017), from physical infrastructures to databases. This special issue furthers this line of research. It argues that starting the analysis from a more precise object, computer code, and a more circumscribed activity, the writing of code, can not only document software in the making but also contribute to a more global understanding of software infrastructures.

1.1. Hacker, geek, entrepreneur: beyond the mythical figures

5 As an activity, making software largely remains in the shadow of great mythical figures. Whether it be hackers (Hikkamen, 2001; Auray, 2013; Lallement, 2015), geeks (Kelty, 2008), or even genius entrepreneurs (Turner, 2006), these emic[4] representations often attract attention to the detriment of concrete software practices. The latter are then regularly obscured from academic work, in favor of the — sometimes conflicting — discourses that coders hold about their worlds. These discourses include, for example, the observation that code is the law of cyber-space (Lessig, 1999); the view that programming is an instrument of political (Auray and Ouardi, 2014) or economic emancipation (Stevens 2012; Vicente 2017); the promise of a reinvention of work through play and experimentation in the programming activity (Berrebi-Hoffmann et al, 2018; Flichy, 2017); the enthusiastic remark that "software is eating the world"[5] and that all aspects of our contemporary societies could be transformed, even improved, by putting them into code.

6 Accounting for these representations is undoubtedly a strength for sociological and anthropological research. Thanks to this work, we have gained a much more granular understanding of essential elements of the technical (Coleman, 2010), political (Coleman, 2013; Auray, 2007; Broca, 2013), and economic (Rosental, 2017; Vincente 2017) cultures of computing environments. These discourses help us to understand the interest in code, the relationships that individuals have with their activity, or the strategies of distinction and legitimization deployed by certain social groups. But discourses, however operative they may be, are not practices. If the former have been remarkably invested by the social sciences, we cannot say the same about the latter. When surveys and fieldwork are carried out, they often approach it through the prism of these discourses, whether to verify or invalidate them. Despite the answers provided by this body of research, the questions remain, on the whole, unchanged — and these questions are what we need to renew today.

1.2. Avoiding the algorithmic reduction

7 When software is not overshadowed by these mythical figures, it is regularly reduced to the problem of algorithms. The growing centrality of the concept of algorithm is easily understood. On the one hand, algorithms seem to find ever more concrete applications and more data to process, from the recommendation of cultural goods, such as music (Beuscart et al 2019), to the dissemination of information (Benkler, 2018), to justice (Christin, 2017), to predictive policing (Benbouzid, 2017), and so many others, not to

mention the social sciences themselves (Edelman et al 2020). On the other hand, algorithms are seemingly constantly gaining autonomy thanks to systems such as neural networks (Cardon et al. 2018). The fear is that these technologies, pictured as capable of learning, will absorb the inequalities inherent in data and reproduce systems of oppression (Noble, 2018). Algorithms are now part of a double *mise en scène*, which presents them both as lacking transparency and as omnipotent. The power of the "algorithmic drama" (Ziewitz, 2017) lies in this interplay between opacity and power, which reinforce each other. If we assume that algorithms have power, it is not easy to understand how they exercise it, which makes them all the more mysterious. Conversely, their opacity can be interpreted as an additional clue of their power. In short, if we do not know what algorithms are and what they do, everyone agrees on their status as objects of power.

8 However, sociologists and anthropologists who follow the trail of algorithms rarely come across such entities because, in the practice of making software, the word designates very different objects, with changing contours. For example, many developers understand "algorithm" as a reference to the classic sequences of instructions they learned at university or in engineering school. In this sense, algorithms manifest themselves when a developer is reviewing her algorithmic knowledge for a job interview or an engineer is searching a forum for a precise implementation of quicksort[6]. No matter how complex or impressive, algorithms are just one element in a vast toolbox — a far cry from the mysterious objects of power we mentioned earlier. Conversely, if we understand an algorithm to mean any sequence of instructions that can be executed by a computer, the meaning becomes so broad that it could encompass every aspect of software production, to the point that the object struggles to be operational for the social sciences. Nick Seaver rightly points out that in computer science, the word "algorithm" is undergoing a trajectory similar to that of the word "culture" in anthropology (Seaver, 2017), becoming so popular inside and outside its discipline that it is almost overused. Under these circumstances, sociological inquiry struggles to define and identify these infamous black boxes. And often, when it does find them, opening them proves to be extremely difficult, all for a confusing, sometimes disappointing result (Winner, 1993).

9 Algorithms are therefore not always the right entry point for understanding computer activity; an excessive focus on algorithms could even end up obscuring a large part of the practices, knowledge, and above all the meaning of computer activity. Our aim is not to contest the algorithm as a sociological object — several articles in this issue deal with this question by approaching it from a different angle — but rather to re-situate it among the many other objects of computing activity.

1.3. Studying software as it is written

10 Our proposal to consider software activity as a form of writing finds many echoes in various research traditions. Let us note, at the turn of the 1990s, the will to take the history of computing out of its internalist preoccupations to inscribe it in the contemporary questions of the history of techniques (Mahoney, 1988) and, more precisely, of information technologies (Kranakis, 1994; Aspray, 1994). At the same time, a new wind swept through several disciplines, from design to sociology, asking how computer technology assists (or can assist) cooperation in work situations: studies on

Computer Supported Cooperative Work (CSCW) have opened up several theoretical and practical debates, for example around performativity (Suchman, 1994; Winograd, 1994). In the early 2000s, it was the turn of Software Studies to bring together a group of diverse fields, from Media Studies to engineering, to apprehend software as cultural objects and practices. Later, Critical Code Studies encouraged a more specific focus on source code; they were the first to assiduously participate in the construction of code as a textual object, resulting from writing practice. In the French-speaking scientific literature, the sociology of work has also examined the organization of software production in the context of open source software (Demazière et al., 2007), while the sociology of writing has appropriated code by extending the intuition of Critical Code Studies to consider it as a text (Couture, 2012). If many others before us have opened the way of software writing, we think that this lead deserves more exploration and that it still holds many discoveries. But what exactly does it mean to study software as it is written?

11 To be interested in the writing of codes means, first of all, to place the activity in the foreground. The goal is to detach ourselves, if only for a moment, from the mythical figures and discourses that surround it in order to think of this object in the context of its ordinary, routine, vulnerable aspects. Writing software is not simply a matter of producing new code. Many operations involve, for example, "caring for things" with respect to scriptural infrastructures that must be maintained (Denis & Pontille, 2012; Denis & Pontille, 2020). Others simply involve understanding and managing existing code (Couture, 2012). As in any anthropological study of infrastructure, the investigation sheds light on these laborious, tedious, or invisible actions to better explore this interplay between the transparent and the opaque (Star, 1999). Understanding the writing of code in all its nuances reveals how these differences serve to support distinctions, legitimization strategies, for example when an engineer automates what he considers to be thankless "manual" work (Alcaras, 2020), and more broadly the construction of a professional ethos (Zarca, 2009).

12 Studying software as it is written also implies taking into account the relationship that coders have with this activity. This is part of a sociological approach that is concerned with technique (Desrosières, 2013; Dagiral and Martin, 2017) and that refuses to "leave aside the realities experienced by those who develop, distribute and promote the software and technologies concerned" (Vinck & al., 2018). Beyond this approach, relationship to software writing participates in the construction of identities in the professional sense (Perrenoud et al., 2018) as well as in the broader sense through, for example, gender performance (Faulkner, 2000) in a male-dominated world of work (Jorgenson, 2002; Collet, 2006). Finally, since work and relationship to activity inform the structure and values of professions (Abbott, 1988), focusing on the content and experience of software writing sheds light on the professional worlds of computing as a whole.

13 Taking the writing of codes as an object invites us to consider this activity in all its materiality, starting with the concrete conditions of work. If code is written everywhere, it is written differently according to these conditions of production. Indeed, the software worlds testify to a vast heterogeneity of statuses (from the freelance developer to the salaried engineer), of contexts (militant spaces, IT departments in non-tech industry, prominent software company in Silicon Valley), of scales (from small personal scripts to the vast projects of digital multinationals) and of

spaces (traditional office, co-working spaces, remote work). The analysis of the materiality of software writing continues with how one relates to the matter of code. While the algorithm allows us to think in terms of a conceptual device, code encourages us to look at the concrete act of writing programs, i.e. the production of a text that will be interpreted or compiled by machines, inserted into hardware and software infrastructures, read and amended by colleagues, copied and pasted by amateurs or hobbyists and so on. By firmly anchoring our analysis of software in the materiality of code, we can then account for the diversity, tensions, or conflicts that run through these practices, that is to say how codes are written.

14 The ambition of this special issue is not to proclaim that everything is code or that code is the only valid object; that would be tantamount to denouncing one reductionism and immediately replacing it with another. To separate completely the program from the code would be artificial; on the other hand, this distinction can help us to think about software from all angles. Thus, if programming refers to planning and computer architecture, then coding refers to the concrete and material task of making software. By taking a close look at codes, we are not only documenting an essential, everyday activity of computing: we can also observe and analyze how other objects, algorithms or otherwise, manifest themselves in material practices. This is why this special issue considers software writing both as an object of analysis and as a gateway into the "inside" of digital infrastructures.

2. Asking old questions to new objects

15 Although computing has been democratized, massified, and made commonplace over the last four decades, code still gives the impression of being a new object for sociology. This feeling of novelty is probably explained by the limited amount of existing empirical research compared to the immensity of the software field. As a result, code can sometimes appear to be a frightening object, requiring considerable efforts of theorizing to become a sociological object. In our opinion, software writing precisely has the advantage of being at the crossroads of many research traditions. In other words, this object allows researchers to use a wide range of theories, heuristic questions, and empirical approaches. In the spirit of the RESET journal, this special issue is an invitation to borrow and develop long-standing and classical sociological concerns to apply them to computer codes.

2.1. Starting with software know-how

16 Discourses on the mythical figures of the digital world as well as on algorithms focus, in their own way, on the power of software. The representation of the hacker emphasizes the emancipating role of this activity for individuals, while the algorithmic reduction is more commonly seen in the form of technological determinism. Despite their differences, these discourses first raise the question of power. Looking at the writing of codes puts aside — at least temporarily — the question of what software can do to investigate what coders know.

17 How does an engineer find the right line of code to solve a bug? Why does a coder choose one styling convention over another? How does a systems architect become constrained by decisions made over thirty years ago? How does a group of developers

build a collective representation of the code they are working on? What does one person mean when he or she says that they need to fix this infrastructure "manually"? In summary, we suggest returning to the first questions of cognitive anthropology and applying them to the people who write software. What do they need to know to do what they do? How do they manage to know what they know (Hutchins, 1994)? Software writing is thus revealed as a vast set of practices, bits of knowledge, and, above all, skills and *savoir-faire* (know-how).

18 Surprisingly, this shift from the political to the epistemic does not limit software to a mere intellectual exercise. On the contrary, studying software as it is written invites us to consider it a concrete activity. The code is seen as a text, i.e. as a material that has its own constraints. This material becomes an object of knowledge and action thanks to the many tools that compose a digital environment. As an inscription, the code is from the start closely inserted into a vast scriptural infrastructure (Denis, 2018) which is itself the result of a long history of construction and maintenance. Analyzing software writing means investigating the know-how involved in the materiality of text, which is based on knowledge and actions situated in a scriptural environment. Such skills are inherited from past technical constraints and grow around collective norms and representations.

19 By bringing to light what code workers know, we also see how much is beyond their knowledge. To write code is to constantly come up against gray areas and unexpected events, whether it be a minor bug that appears without any apparent cause. The lack of knowledge can also be more radical and yet pose no problem, for example when engineers use an algorithm without understanding how it works. The sociology of software writing can then draw the contours of computer knowledge in action, explain why certain gray areas go unnoticed while others become territories to be explored, and underline the strategies implemented both to circumvent the obscurity or to shed light on it.

2.2. How acts of writing question the power of code

20 Questioning software writing skills gradually clarifies what coders can do and, ultimately, what code can actually do. A detailed analysis of these practices avoids the trap of "code fetishism" denounced by Wendy Chun, a pioneer in code studies (Chun, 2008). This pitfall consists, she says, of confusing source code with its machine execution. This confusion is explained by the military and gendered history of software: "in the military, there is supposed to be no difference between a command given and a command completed" (p. 304). This critical approach toward the power of code leads her to question Lawrence Lessig's famous expression "code is law". According to Chun, this statement expresses a fantasy of what the law should be, based on an embellished image of code execution. Instead, she invites us to wonder how software can emerge from code and how code can become executable and performative.

21 While the algorithmic object allows us to think in terms of dispositive (in a Foucaldian sense), it also favors an intellectual vision of programming, separated from the resources by which the code is implemented. Programming is not necessarily grounded in digital technology, nor does it always involve computers (Aspray, 1990). Before the appearance of the first alphanumeric languages in the 1950s, algorithms were directly

programmed using material objects, such as the well-known punched cards of the Jacquard machine or tabulating machines (Gardey, 2008, pp. 263-267). Indeed, Alan Turing (1937) uses the term computer to designate a "calculator", i.e. not a machine, but a human being who calculates (Mélès, 2015). Similarly, the latest developments in machine learning make it possible to execute instructions that have not been coded explicitly. Understood as systematic methods for solving a problem, algorithms even exist outside the digital realm, for example in the solution of puzzles such as the Rubik's Cube. To borrow from the anthropology of writing, code and algorithms refer to different intellectual technologies and material cultures (Goody, 1979) — although they sometimes overlap, they are not equivalent.

22 However, thinking of code execution as an act of writing (Fraenkel & Pontille, 2003; Fraenkel, 2007) emphasizes the whole set of material, technical and social conditions that allow code to become effective. Execution is a long road; it does not always succeed. Beyond the scriptural restraints that affect the code as a text (syntax, style, integration), we also think of the social phenomena that make it possible (for instance, to agree on which version of the code is the official "source"). Finally, following software writing shows the duality between code and its execution. A line of code that seems perfectly clear, that runs without glitches, can yet produce a completely unexpected outcome: a situation that some coders use to their advantage in "sneaky" code challenges[7]. Engineers can also play with this confusion between code and execution to their advantage, such as when they perform an artificial intelligence that is not present in the code (see Ionescu, 2022 in this issue).

23 At the same time, we can also consider how the act of writing, once it has taken effect, becomes authoritative in software worlds. This authority explains why engineers prefer to use software rather than simple guidelines to enforce technical standards. It also explains why certain codes can become crucial for groups with diverging interests, thus creating conflict to control these strategic resources. Thinking of computer science as a form of writing thus questions the technical and social construction of a scriptural power, while simultaneously taking into consideration the reality of effects that writing produces (Denis, 2018).

2.3. Discourses of codification and code practices

24 The activity of writing code is therefore not immune to the process of "intellectualist rationalization that we owe to science and scientific technique" (Weber, 1959). Indeed, some discourses insist on the "mastery", "efficiency" and "predictability" of the practice. Rather than reinforcing this illusion of a fully normed and rationalized field, like the "truth discourses" of Web 2.0 promoters (Bouquillion and Matthews, 2010), we prefer to "show how social actors negotiate the meaning of each of these words before imposing them on others as primary truths" (Callon, 2013). In response to these discourses, many researchers show that the uses of a technology can escape its producers (Jouët, 2000; Proulx, 2015), for example when users appropriate, divert or re-invent scripts (Akrich 1987; 2010). The papers in this issue take inspiration from these works and extend them by investigating the uses of the people who produce software. In other words, coding is not only creating; it also implies using tools and techniques.

25 This perspective leads us to distinguish between the code practices and the discourses about code, which are often discourses of codification. In a paper entitled "Habitus,

code and codification" (1986), Pierre Bourdieu focuses on the social processes of norm formalization, particularly in the legal and linguistic fields. According to him, "to codify is both to put into form and to put into shape [*mettre en forme et mettre des formes*]": the rules meant to formalize practice become an area of struggle and control themselves. Similar phenomena run through software writing. We observe both formal codification processes (such as the setting up and negotiation of web standards within the W3C) and a play on informal practices, such as copying and pasting from the StackOverflow Q&A platform. Mastery of code is therefore accompanied by mastery of codes, be they explicit or implicit.

26 The respect of norms ("to obey rules"), their subversion ("to play with the rules"), or their abrogation ("to break the rules") will not have the same symbolic value depending on the context: rationalization and optimization of production processes in engineering, values promoted by the hacker ethic, aesthetic appreciation of the code as an art form (Knuth, 1974), proximity with the "disciplinary matrix" of computer science (Millet, 2003). By studying the relationship between activity and discourse, between code and codification, between "making " and "talking about making [*dire sur le faire*]" (Lahire, 1998), software writings are placed in the continuum of literary, scholarly and ordinary writings.

27 To do this, the discourse of coders that gives coherence and meaning to practices must be considered accordingly. This is why studying software as it is written requires an empirical observation of writing to shed light on a set of informal practices, micro-practices, and invisible practices. The sociology of software writing is therefore faced with two challenges. The first challenge is to overcome a double methodological difficulty: what does it mean to "observe" in the context of a digital field? How can we understand what is a't stake in these practices without systematically relying on codification discourses, without forgetting the importance of such discourses? The second challenge concerns the question of scientific writing. How can we describe a teeming activity without losing the thread of analysis? How can we account for messiness without being messy ourselves? How do we shape these descriptions without adopting the same codification discourse?

28 These questions run, of course, through all social sciences. If the task seems especially difficult in the case of software, it is undoubtedly because we still have only a small number of works documenting the specific pitfalls of these fields (Mahoney, 2008). For this reason, we are particularly pleased to bring together, in this special issue of RESET, seven pieces of research, each proposing an original way of approaching software writing, each responding in its own way to the questions mentioned above.

3. Investigating software writing

29 This issue opens with an exploration of the world of Silicon Valley developers, which situates software in its material writing conditions. Olivier Alexandre explores the daily life and specificities of professional careers marked by a high degree of uncertainty. Through observations, interviews, and online activity monitoring, the author presents the many strategies used by these workers to address the various expressions of these uncertainties, whether in design and maintenance activities, in their project orientation, or in their professional trajectories. The career of a computer developer is depicted as a series of orientations within a variety of mediations and tools, far from

the representations of a controlled and predictable job. By revealing the individual "catches" likely to guide the action and manage uncertainty (such as the rapid obsolescence of certain skills) this paper provides more subtle insights about the processes involved within a labor market with recurring injunctions to "evolution, agility, and fluidity".

3.1. Knowledge and ignorance in software production

30 After this analysis of software engineering careers, two ethnographic studies immerse us in the details of software writing, especially in its epistemic and practical dimensions.

31 First, Tudor Ionescu devotes a rich ethnographic study to a common and yet frowned-upon practice, hardcoding. This practice consists in specifying a mechanism or a piece of data as explicitly as possible in the code itself, without considering a more global solution to a particular problem. Through several "vignettes", the author follows the activities of a development team that is about to demo an intelligent robot on an assembly line. But the engineers have neither the time nor the means to develop this technology. So they resort to hardcoding and dictate in an extremely precise and — literally — millimetric way the motions that the machine must make, rather than creating an intelligence that would decide on the motions by itself. While the code reflects the extent of what the machine cannot do, its execution results in a convincing performance by the robot, which gives the impression that this artificial intelligence is within reach. In addition to informing many of the typical problems of industrial software writing (technical debt, integration hell), this investigation offers a fascinating examination of the duality between code and execution as well as the question of performativity. The author also shows how software writing is directly influenced by the economy of promise, project-based processes, and the mundane constraints of production.

32 The next article provides a perspective on a very ordinary sequence of software writing: an episode of debugging. In a meticulous investigation, Florian Jaton chooses to analyze step by step a scene that lasts only a few minutes to better understand the — sometimes dizzying — depth of each writing phase. In an Image Processing laboratory and alongside a researcher who encounters a bug, the author shows how the coder temporarily adopts the attitude of an investigator and tries to understand where the problem comes from. For a few moments, her computer becomes a kind of laboratory where she conducts a series of small experiments to isolate the line responsible for the bug. The author closely observes and describes these various experiments, which involve the use of documentation, online forums, and technical tools such as a debugger. Throughout this detailed and original description, the activity gradually gains consistency. The coder mobilizes different skills to inform the state of "distant entities" that are poorly known, such as the Interpreter, through a series of inscriptions (code, documentation, output of the Interpreter) that must be properly aligned. These small moments of knowledge production, quite similar to the scientific approach, shed light on these trivial and ephemeral moments of code writing.

3.2. Codifying code: from style to standards

33 Code practices, such as hardcoding or debugging, are often caught up in codification processes. These interactions between code and codification are the subject of the following two articles in this special issue. The first one looks at standardization in the web arena ; the second one focuses on the codification of style in the software industry.

34 By focusing on the standardization arenas of the W3C and the IETF, Julien Rossi describes the discussions, the controversies, and the consensus forms that develop there. In these places where communication takes place at the junction of face-to-face meetings and digital coordination tools, the author documents the methodological challenges of multiplying the types of data. Interviews, observations, analyses of meeting records, and mailing lists provide relevant leads, provided that their respective limitations are identified and the background of these sources is resituated. In these arenas, where the "injunction to reach consensus" is omnipresent, the actors give priority to the regime of technical justification to express their disagreements and criticisms. This norm is strengthened and regulated by the types of sanctions and (re)framings that take place in the debates. The intersection of the data then reveals the critical and political postures of the participants and reveals the discursive strategies that are deployed in these negotiation spaces.

35 Pierre Depaz addresses another sphere of the codification of software writing: the regulation and negotiation of style. The author analyses the tension between the subjective dimension of style (such as a personal preference for the use of a semicolon at the end of a line of code) and the emergence of collective norms. The latter aim to maintain consistency of style across a project, even when several people are involved. These norms are often codified in style guides, which are the focus of this article. By comparing three common style guides that are used for the Javascript language, the author seeks to understand how the content of these guides is negotiated, which arguments are mobilized, and which registers of justification ultimately prevail. If the argumentative repertory is common to the three guides, each one presents its specificities, in particular when the guide is accompanied by software that identifies, or even automatically corrects, the passages that deviate from its standards. Among other tensions between code and codification, the article explores the strength of the argument of technical authority.

3.3. The autonomy of software writing in question

36 This issue ends with two contributions that study how software writing fits into a set of industrial practices. They examine the relative autonomy of software writing, first in the face of insecurity processes while developing an encryption protocol, and then in the face of a set of guiding practices in the context of the development of recommendation algorithms.

37 In a survey of developers working on a cryptographic protocol, Sylvain Besençon explores the issues surrounding the security of computer code. By focusing on both the critical and routine aspects of maintenance and care, he makes explicit the constant efforts to make it less vulnerable. Far from the representation of a security universe regularly disrupted by flaws, he insists on the collective and cyclical dimension of securing and insecuring the code. Each new entry risks making the code more fragile,

even when the writing is intended to solve a security problem. By informing little-documented practices with a variety of material (public conferences, interviews, observations, online discussions, code-sharing platforms), the author invites us to shift the sensational gaze on security issues to the examination of routine practices of clarification, *ad hoc* amendments, and daily maintenance. The (in)securitization is thus analyzed as a process of continuity rather than rupture, inscribed in collaborative dynamics at the crossroads of different spheres of actors (organizations, companies, academia).

38 The last article shows how a number of tools, especially algorithmic ones, are used to write code. Camille Roth and Jérémie Poiroux examine numerous interviews with people responsible for developing recommendation algorithms. The authors show that this writing relies on a large number of infrastructures and algorithmic tools to repair, adjust or evolve the code of their own algorithms. They are particularly interested in the widespread, but so far little documented, practice of A/B testing. A/B testing consists of proposing different versions of the same functionality to quantify the effect of each variation on the uses of a given platform. Exploring these practices highlights that, contrary to the image of a completely rationalized and planned software engineering, many developers tinker. They do not try to open the algorithmic black boxes they deploy; most of them simply multiply the tests to retain the variations that seemingly give the best result. The authors discuss the theoretical and methodological consequences of this result since, in the end, this form of software writing seems to be less affected by algorithmic problems than by issues of quantification and benchmarking.

4. Conclusion. Software as practice and belief

39 In their famous paper *L'informatique comme pratique et comme croyance* (Software as practice and belief), Michel Gollac and Francis Kramarz (2000) note "the impossibility of an autonomous 'software field'" meaning any practice related to the use of a computer, from office work to writing code. Without neglecting research that considers programs and their digital uses in their entirety, this issue shows the interest in adopting a more precise definition of computing, captured by code practices. While the object of software writing seems particularly promising, it is only one starting point among others to explore the "inside" of digital infrastructures.

40 Asking new questions and constructing new objects: this issue is above all an invitation to explore software from all angles, with renewed curiosity and enthusiasm. Beyond the theoretical questions and their insights into computing activity, we hope above all that the contributions in this issue will provide concrete examples of original investigations and accounts — and that they will encourage further exploration of the practices and beliefs of software worlds.

BIBLIOGRAPHY

ABBOTT A., 1998, *The System of Professions. An Essay on the Division of Expert Labor*, University of Chicago Press, 452 p.

AKRICH M., 1987, « Comment décrire les objets techniques ? », *Techniques & Culture. Revue semestrielle d'anthropologie des techniques*, 9.

AKRICH M., 2010, « Retour sur « Comment décrire les objets techniques ? » », *Techniques & Culture. Revue semestrielle d'anthropologie des techniques*, 54-55, p. 202-204.

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

ASPRAY W., (dir.) 1990, *Computing before computers*, Iowa State University Press, 266 p.

ASPRAY W., 1994, « The history of computing within the history of information technology », *History and Technology*, *11*, 1, p. 7-19.

AURAY N., 2007, "Le modèle souverainiste des communautés en ligne : impératif participatif et désacralisation du vote", Hermès, n°47, p.137-145.

AURAY N., OUARDI S., 2014, « Numérique et émancipation », *Mouvements*, 3, p. 13-27.

AURAY N., VÉTEL B., 2013, « L'exploration comme modalité d'ouverture attentionnelle », *Réseaux*, 6, p. 153-186.

BARON G.-L., 2018, « Informatique et numérique comme objets d'enseignement scolaire en France: entre concepts, techniques, outils et culture »,.

BENBOUZID B., 2017, « Des crimes et des séismes », *Reseaux*, *206*, 6, p. 95-123.

BENKLER Y., FARIS R., ROBERTS H., 2018, *Network Propaganda: Manipulation, Disinformation, and Radicalization in American Politics*, New York, Oxford University Press, 472 p.

BERREBI-HOFFMANN I., BUREAU M.-C., LALLEMENT M., 2018, *Makers-Enquête sur les laboratoires du changement social*, Média Diffusion.

BEUSCART J.-S., COAVOUX S., MAILLARD S., 2019, « Les algorithmes de recommandation musicale et l'autonomie de l'auditeur », *Reseaux*, *213*, 1, p. 17-47.

BOUQUILLION P., MATTHEWS J.T., 2010, *Le Web collaboratif: mutations des industries de la culture et de la communication*, Presses Universitaires de Grenoble.

BOURDIEU P., 1986, « Habitus, code et codification », *Actes de la Recherche en Sciences Sociales*, *64*, 1, p. 40-44.

BROCA S., s. d., *Utopie du logiciel libre*, Passager clandestin (Le).

CALLON M., 2013, « Pour une sociologie des controverses technologiques », dans AKRICH M., LATOUR B. (dirs.), *Sociologie de la traduction : Textes fondateurs*, Paris, Presses des Mines (Sciences sociales), p. 135-157.

CARDON D., COINTET J.-P., MAZIÈRES A., 2018, « La revanche des neurones », *Reseaux*, *211*, 5, p. 173-220.

CHRISTIN A., 2017, « Algorithms in practice: Comparing web journalism and criminal justice », *Big Data & Society*, *4*, 2, p. 2053951717718855.

CHUN W.H.K., 2008, « On "Sourcery," or Code as Fetish », *Configurations*, *16*, 3, p. 299-324.

COLEMAN E.G., 2010, « The Hacker Conference: A Ritual Condensation and Celebration of a Lifeworld », *Anthropological Quarterly*, *83*, p. 42-76.

COLLET I., 2006, *L'informatique a-t-elle un sexe?*, Editions L'Harmattan.

COUTURE S., 2012, « L'écriture collective du code source informatique », *Revue d'anthropologie des connaissances*, *6*, 1.

DAGIRAL É., MARTIN O., 2017, « Liens sociaux numériques », *Sociologie*, N° 1, vol. 8.

DEMAZIÈRE D., HORN F., ZUNE M., 2007, « Des relations de travail sans règles ? L'énigme de la production des logiciels libres », *Societes contemporaines*, *66*, 2, p. 101-125.

DENIS J., PONTILLE D., 2012, « Travailleurs de l'écrit, matières de l'information », *Revue d'anthropologie des connaissances*, *6*, 1, p. 1-20.

DENIS J., 2018, *Le travail invisible des données : Éléments pour une sociologie des infrastructures scripturales*, Paris, Presses des Mines (Sciences sociales), 206 p.

DENIS J., PONTILLE D., 2020, « Maintenance et attention à la fragilité », *SociologieS*.

DESROSIÈRES A., 2013, *Pour une sociologie historique de la quantification: L'Argument statistique I*, Presses des Mines via OpenEdition, 331 p.

DODIER N., 1995, *Les hommes et les machines: la conscience collective dans les sociétés technicisées*, FeniXX.

DROT-DELANGE B., BRUILLARD E., 2012, « Éducation aux TIC, cultures informatique et du numérique: quelques repères historiques », *Études de Communication. Langages, information, médiations*, 38, p. 69-80.

EDELMANN A., WOLFF T., MONTAGNE D., BAIL C.A., 2020, « Computational Social Science and Sociology », *Annual Review of Sociology*, *46*, 1, p. 61-81.

ERMOSHINA K., 2017, « Le code peut-il réparer les routes? », *Reseaux*, 6, p. 155-189.

FAULKNER W., 2000, « Dualisms, Hierarchies and Gender in Engineering », *Social Studies of Science*, *30*, 5, p. 759-792.

FLICHY P., 2017, *Les nouvelles frontières du travail à l'ère numérique*, Média Diffusion.

FRAENKEL B., 2007, « Actes d'écriture : quand écrire c'est faire », *Langage et société*, *121-122*.

FRAENKEL B., PONTILLE D., 2003, « L'écrit juridique à l'épreuve de la signature électronique, approche pragmatique », *Langage et societe*, *104*, 2, p. 83-122.

GALLOWAY A.R., 2012, *The interface effect*, Polity.

GARDEY D., 2008, *Écrire, calculer, classer: Comment une révolution de papier a transformé les sociétés contemporaines (1800-1940)*, Paris: La Découverte.

GOLLAC M., KRAMARZ F., 2000, « L'informatique comme pratique et comme croyance », *Actes de la Recherche en Sciences Sociales*, *134*, 1, p. 4-21.

GOODY J., 1975, *La raison graphique*, Editions de Minuit, Paris

HIMANEN P., LEBLANC C., 2001, *L'éthique hacker et l'esprit de l'ère de l'information*, Exils Paris.

HUTCHINS E., 1994, *Cognition in the wild*. Cambridge, MIT Press.

JORGENSON J., 2002, « Engineering Selves: Negotiating Gender and Identity in Technical Work », *Management Communication Quarterly*, *15*, 3, p. 350-380.

JOUËT J., 2000, « Retour critique sur la sociologie des usages », *Réseaux. Communication - Technologie - Société*, *18*, 100, p. 487-521.

KRANAKIS E., 1994, Introduction, *History and Technology,* 11, pp. 1-5

KELTY C.M., 2008, *Two bits: The cultural significance of free software*, Duke University Press.

KNUTH D.E., 1974, « Computer programming as an art », *Communications of the ACM*, *17*, 12, p. 667-673.

LAHIRE B., 1998, « Logiques pratiques : le « faire » et le « dire sur le faire » », *Recherche & formation*, *27*, 1, p. 15-28.

LALLEMENT M., 2015, *L'Âge du Faire. Hacking, travail, anarchie: Hacking, travail, anarchie*, Média Diffusion.

LESSIG L., 1999, *Code: And other laws of cyberspace*, Basic Books.

MAHONEY M.S., 1988, « The History of Computing in the History of Technology », *Annals of the History of Computing*, *10*, 2, p. 113-125.

MAHONEY M.S., 2008, « What Makes the History of Software Hard », *IEEE Annals of the History of Computing*, *30*, 3, p. 8-18.

MÉADEL C., SIRE G., 2017, « Les sciences sociales orientées programmes », *Reseaux*, 6, p. 9-34.

MÉLÈS B., 2015, « L'informatique sans ordinateur », Site *CNRS : Images des mathématiques*.

MILLET M., 2021, *Les Étudiants et le travail universitaire : Étude sociologique*, Lyon, Presses universitaires de Lyon (Hors collection), 256 p.

MOATTI A., 2012, « Le numérique, adjectif substantivé », *Le débat*, 3, p. 133-137.

NOBLE S.U., 2018, *Algorithms of Oppression: How Search Engines Reinforce Racism*, New York University Press.

PERRENOUD M., SAINSAULIEU I., 2018, « Pour ne pas en finir avec l'identité au travail », *SociologieS*.

PROULX S., 2015, « La sociologie des usages, et après? », *Revue française des sciences de l'information et de la communication*, 6.

ROSENTAL C., 2017, « Les conditions sociales des échanges dans la Silicon Valley », *Zilsel*, *1*, 1, p. 55-81.

SEAVER N., 2017, « Algorithms as culture: Some tactics for the ethnography of algorithmic systems », *Big Data & Society*, *4*, 2, p. 2053951717738104.

STAR S.L., 1999, « The Ethnography of Infrastructure », *American Behavioral Scientist*, *43*, 3, p. 377-391.

STEVENS H., 2012, « Autonomie récusée, autonomie fabriquée. Informaticiens à l'épreuve de l'Entreprise de Soi », *Genèses*, *87*, 2, p. 90-112.

SUCHMAN L., 1994, « Do categories have politics? The language/action perspective reconsidered », *Computer Supported Cooperative Work (CSCW),* 2, p. 177-190.

TURING A.M., 1937, « On Computable Numbers, with an Application to the Entscheidungsproblem », *Proceedings of the London Mathematical Society*, *s2-42*, 1, p. 230-265.

TURNER F., 2006, *From counterculture to cyberculture*, University of Chicago Press.

VICENTE M., 2017, « Apprentissage du code informatique et entrepreneuriat: de la création d'entreprise à l'esprit d'entreprendre », *Formation emploi. Revue française de sciences sociales*, 140, p. 87-106.

VINCK D., CAMUS A., JATON F., OBERHAUSER P.-N., 2018, « Localités distribuées, globalités localisées: actions, actants et me·diations au service de l'ethnographie du nume·rique », *Symposium*, *22*, 1, p. 41-60.

WEBER M., 1959, *Le savant et le politique*, Paris, Plon.

WINNER L., 1993, « Upon Opening the Black Box and Finding It Empty: Social Constructivism and the Philosophy of Technology », *Science, Technology, & Human Values*, *18*, 3, p. 362-378.

WINOGRAD T., 1993, « Categories, disciplines, and social coordination », *Computer Supported Cooperative Work (CSCW)*, *2*, 3, p. 191-197.

ZARCA B., 2009, « L'ethos professionnel des mathématiciens, The professional ethos of mathematicians. », *Revue française de sociologie*, *50*, 2, p. 351-384.

ZIEWITZ M., 2016, « Governing Algorithms: Myth, Mess, and Methods », *Science, Technology, & Human Values*, *41*, 1, p. 3-16.

NOTES

1. Translations from French are our own.

2. Our conceptualization partly comes from the French term "informatique", which has the advantage of uniting fields and activities that are often separated in the English language, namely computer science (the scientific discipline) and software engineering (industrial engineering). Since the word "informatics" already has a specific meaning, we prefer translating using the more generic term of "software".

3. An API, an acronym for *Application Programming Interface*, commonly refers to a system that enables communications and interactions beween several programs.

4. We borrow the term "emic" from the anthropological tradition (Sardan, 1998). It refers to the discourses and representations that make sense within the social world under study, in this case the software worlds.

5. See statement by investor Mark Andreesen, "Why Software Is Eating The World," Wall Street Journal, August 20, 2011: https://www.wsj.com/articles/SB10001424053111903480904576512250915629460 (accessed January 15, 2022).

6. *Quicksort* belongs to the family of sorting algorithms, which aim to order the elements of a data structure, such as a list or an array. Known for its speed in most situations, this algorithm uses a pivot system: it selects an element around which the other elements are permuted, so that those below it are on its left and those above it are on its right. The operation is repeated recursively within each partition.

7. For example, http://underhanded-c.org/ : « The Underhanded C Contest is an annual contest to write innocent-looking C code implementing malicious behavior. ».

ABSTRACTS

Digital infrastructures often appear as fascinating devices, both incredibly complex and perfectly ordered. While these infrastructures are the subject of much discourse, we know little about their production and maintenance practices, especially given their scope and diversity. How is the software we use every day created? Who builds the IT infrastructures of our contemporary societies? What is the digital world made of? This text provides a theoretical backdrop to introduce the contributions of the special issue. In order to go beyond the mythical figures (hacker, geek, entrepreneur) and to escape the algorithmic reduction, studying software as it is written offers multiple outlets for empirical investigation. This object is also at a fruitful intersection between several research traditions. It opens up promising avenues of inquiry into software know-how, performativity and the process of codification.

INDEX

Mots-clés: Code, infrastructure, digital, writing, practices

AUTHORS

GABRIEL ALCARAS

Centre Maurice Halbwachs, i3

ANTOINE LARRIBEAU

EXPERICE (Université Sorbonne Paris Nord)