

Middle Tech: Software Work and the Culture of Good Enough

by Paula Bialski (2024) Princeton and Oxford, Princeton University Press, 224 pp.

Mareike Smolka 

Wageningen University and RWTH Aachen University

Tobias Meierdierks

On Monday morning, Tobias and his co-workers participated in an onboarding session. The team had worked together for years at a German software development company but was recently bought out by an American corporation with regional offices in Europe. During an online meeting, human resource managers from the American headquarters introduced their new employees to the organization and its core values. A manager explained: “Working at our company means challenging yourself while facing each challenge with a positive attitude.” Tobias noticed that his German colleagues looked a bit startled. After some hesitancy, one of them interposed: “This may be an example of cultural differences. In Germany, employees are not expected to display a positive attitude if they do not feel like it on a hard work day. We do our work properly, but pretending to be cheerful is not part of the job.”

This is one of the many anecdotes we have chatted about over lunch during a homeworking day. Our conversations resemble the lunch talks that Paula Bialski describes in her ethnography *Middle Tech: Software Work and the Culture of Good Enough*. Similar to Bialski, Mareike is interested in uncovering the social dimensions of computing from a Science & Technology Studies (STS) perspective. Similar to the interlocutors who Bialski met during ethnographic fieldwork, especially her software developer friend Ori, Tobias opens the door to peek into the world of Medium Tech: middle-sized companies located outside of global tech hubs like the San Francisco Bay Area, Singapore, and Tel Aviv, whose products are well-established on the market and whose employees are moderately ambitious.

The anecdote above illustrates the juxtaposition between what Bialski calls “the culture of good enough” and the discourse of excellence. The discourse of excellence touts a work ethic of outperforming any competitors, engaging in long working hours with a positive attitude, and striving for extraordinary technology that changes the world. This discourse is reiterated in management handbooks, business schools, and the media. Bialski observes that software developers also aspire to such excellence, including those she met during two years of fieldwork at a Berlin-based company, dubbed MiddleTech, which develops mapping, routing, and navigation software for cars. However, her ethnography shows that a culture of good enough hides behind the hegemonic discourse of excellence.

In doing so, Bialski pursues two objectives. First, by making good enoughness visible, Bialski celebrates it. She considers it as a form of resistance against exploitative work environments which squeeze out as much labor power as possible from their employees, milking their (overly) ambitious work drive. Employees protect themselves from work pressure and demands by drawing boundaries around what it entails to do a “good enough” job. For Tobias and his German co-workers, this means doing their work “properly” but without making pretense of a positive attitude. Bialski acknowledges that such boundary-drawing is not available to those working under precarious conditions. In fact, it is enabled by the German labor law which provides greater protection of employees against unfair discharge than in the United States and even includes a *right to average performance and quality*. Second, Bialski exposes excellence as a myth. She distinguishes between what it means to be a “good enough” software engineer and what makes “good enough” software. Her ethnography shows that no matter how good a software developer performs, software can never be more than good enough because of its “material and human sociality” (p. 47).

In five chapters, Bialski describes different aspects of this sociality based on her experiences in MiddleTech. After presenting the company as a case of Medium Tech (Chapter 1), Bialski elaborates on the complex team work required for making software (Chapter 2), the problems that can emerge in team work (Chapter 3), and the role of management (Chapter 4) and temporalities (Chapter 5) in shaping software engineering culture. This culture is characterized by good enoughness partly because the best engineer cannot speed up the time-to-release of a product beyond the slowest worker in the team. Bialski elucidates this phenomenon by highlighting the coordination between multiple interdependent sub-teams who all work on the same product without knowing exactly what the others are doing. They need to ensure that what one sub-team builds fits with what the others are building, and they review each other’s code. In unpacking the review process, Bialski highlights how “good enough” code is defined in a social process of negotiating between the standards of a team, the time it would take to fix some lines of code, and other demands experienced by software developers.

Software developers can afford to release “good enough” code and take the risk that “stuff goes wrong”. According to Ori, “it’s cheaper to fail with our kind of work” (p. 69) than with building a car. Once a car is built and sold to a customer, it is not so simple to fix. Software, by contrast, can usually be updated quite easily via the internet. This has two effects: one relates to the malleability of software whose ongoing mutation implies that negotiations about whether it is “good enough” are repeatedly re-opened in a continuous social process; the other refers to the attitude of software developers to “fix things later”. A big part of software development is fixing bugs. This is the case because software consists of layers on layers of code, some of which – the so-called “legacy code” (p. 86) – is inherited from the original developers. As software grows older, it becomes increasingly complex, to the point where software developers cannot fully understand it, let alone predict how newly written code will affect the whole.

The relation between developers and managers is another aspect of software’s sociality. Managers and management tools, such as Scrum, aim to find the sweet spot between structure and flexibility. Whereas a structured work flow supports the coordination among developers and with their customers, there is also a need for flexibility to handle the complexity, fluidity, and unpredictability of software. In balancing structure and flexibility, negotiations

between management goals of efficiency and customer satisfaction, on the one hand, and software developers' care for software quality and reasonable workload, on the other hand, produce a culture of good enough.

While Tobias enjoys recognizing his own working life in Bialski's ethnographic stories, Mareike appreciates the ways in which these stories engage with literature from organization studies, workplace studies, computer science, and STS. To develop the concept of software's sociality, Bialski draws on classic STS literature, including Wiebe Bijker, Thomas Hughes, and Trevor Pinch's *Social Construction of Technological Systems* as well as John Law and Anne-Marie Mol's "Notes on Materiality and Sociality". When flipping through the pages, one also finds references to Karin Knorr-Cetina, Bruno Latour, Lucy Suchman, Sheila Jasanoff, and Sherry Turkle. Bialski refers to diverse STS authors to reflect on her rich fieldwork experiences, instead of diving into theoretical discussions on the differences and overlaps between their works. This way of thinking and writing with theory creates a light and immersive reading experience because it allows for ethnographic story-telling.

Surprisingly, however, Bialski does not engage with STS literature on how scientists, medical practitioners, and other professionals try to accomplish something good in practice, thereby advancing their values, tastes, and preferences. Informed by overlapping discourses on valuation studies (Heuts and Mol 2013) and empirical ethics of care (Pols 2015), this literature examines the everyday labor and care practices of producing different sorts of goods or values in heterogeneous networks of actors and non-actors. Such a practice-oriented, relational perspective resonates with Bialski's "constellations of good enoughness" (p. 161) which capture the relations that shape the meaning of good enough "for what" and "for whom" (*ibid.*) in a particular context and situation. Examples of these constellations are the relations between a software developer and their company, customer, professional community, tools, and private life. Within these constellations, different values conflict with one another, which induces software developers to make compromises between aesthetic code, production speed, application safety, a good life, and other values.

Research on what is good somewhere can inform critical reflection on what is good elsewhere (Pols 2015). A critical view can also foreground the "dark sides" of good enough, i.e., how it is shaped by power and politics, advancing the interests of some while putting others at a disadvantage. This view seems to fade into the background in Bialski's book. Bialski emphasizes what many of her book's readers may consider "bright sides" of good enough by framing it as a resistance against capitalism: placing care for the self and for others before profit, and prioritizing repair and maintenance over innovation in a degrowth-oriented mindset. She indicates that this emphasis helped her respond to software developers who felt offended by her book (p. 159), associating good enoughness with mediocrity and a sense of being left behind by colleagues who move to Big Tech companies. We also recognize good enoughness as a "freeing gesture" (p. 174). Inspired by a culture of good enough in software work, Mareike asked herself what it meant to be a good enough academic and felt relief while listening to a podcast episode with Bialski that was hosted by STS scholar Lee Vinsel (Peoples & Things 2024). In the episode, Bialski mentions that balancing her roles as academic, musician, and parent sometimes forces her to leave things good enough.

Nevertheless, we would have appreciated a deeper engagement with how power differentials can shape the situated construction of good enoughness. Bialski touches on an example

in analyzing the temporalities of software development. At times, managers make promises to a customer without ensuring that these promises, especially regarding timelines, are feasible for software developers. This can give rise to a situation in which managers decide that the software has to be “good enough” to be released at a particular moment. Under pressure from above, software developers agree with this decision but only after lowering quality standards regarding user experience and accessibility, aesthetic and efficiency of code, and undetected bugs or problems. By prioritizing speed over quality, such a renegotiation of good enoughness at least temporarily stabilizes capitalist dynamics rather than resisting them. Moreover, experience shows that it can have detrimental consequences for specific user groups, such as in the British Post Office Scandal (p. 143). Hundreds of postal workers were falsely accused and pursued for financial shortfalls which were actually caused by faults in an accounting software system. In this case, profit-oriented managers deemed the software to be “good enough” although IT specialists had expressed concerns. We add another example focusing on dynamics of inclusion and exclusion. If a culture of good enough is defined by white male software developers working in the Western world, this can exclude those who program otherwise and elsewhere. Such exclusion can exacerbate discrimination based on derogative stereotypes, for example of the “Bad Indian Coder” (Khazan 2013). Hence, it matters who has the power to define “good enough” technology and which values inform their judgement.

Attending to such “dark sides” of good enough is particularly important as artificial intelligence (AI) enters software development. We fear that GitHub Copilot and other generative AI companions for software developers “inject steroids” into a culture of good enough. They risk lowering the bar of what is seen as good enough, accelerating a race to the bottom. Trained on all kinds of publicly available source code, generative AI produces code that appears good enough at first sight but often turns out to be full of errors and inefficiencies when having a closer look – similar to a text written by ChatGPT composed of well-formulated sentences but filled with commonplace statements that, taken together, generate little meaning. In moments of high work pressure, the temptation to judge an AI-generated code as “good enough” is high. As generative AI seems to be here to stay, we have been excited to find out that Bialski’s current research focuses on Copilot (Peoples & Things 2024). We are confident that her study on Copilot will be a more than good enough complement to her attentive, insightful, and well-written ethnography of MiddleTech.

Funding

The preparation of this article was funded by the Veni Talent Program 2023 – Social Sciences (file number VI.Veni.231S.055) from the Dutch Research Council (NWO).

References

- Heuts, Frank, and Annemarie Mol (2013) *What Is a Good Tomato? A Case of Valuing in Practice*, in “Valuation Studies”, 1(2), pp. 125-146.

- Khazan, Olga (2013) *Behind the “Bad Indian Coder”*. The Atlantic. Available at: <https://www.theatlantic.com/international/archive/2013/10/behind-the-bad-indian-coder/280636/> (retrieved August 1, 2025).
- Peoples & Things (2024) *Paula Bialski on Middle Tech: Software Work and the Culture of Good Enough* (Podcast Season 4, Episode 14). Available at: <https://www.youtube.com/watch?v=3YxWz4sJnno> (retrieved August 1, 2025).
- Pols, Jeanette (2015) *Towards and empirical ethics in care: Relations with technologies in health care*, in “Medicine, Health Care and Philosophy”, 18, pp. 81-90.