HARVARD Kennedy School
CARR CENTER
for Human Rights Policy

FALL 2022

CARR CENTER FOR HUMAN RIGHTS POLICY
HARVARD KENNEDY SCHOOL

Can We Move Fast Without Breaking Things?

Software Engineering Methods Matter to Human Rights Outcomes

Alexander Voss

Carr Center
Discussion Paper

Can We Move Fast Without Breaking Things?

Software Engineering Methods Matter to Human Rights Outcomes

Carr Center for Human Rights Policy
Harvard Kennedy School, Harvard University
October 24, 2022

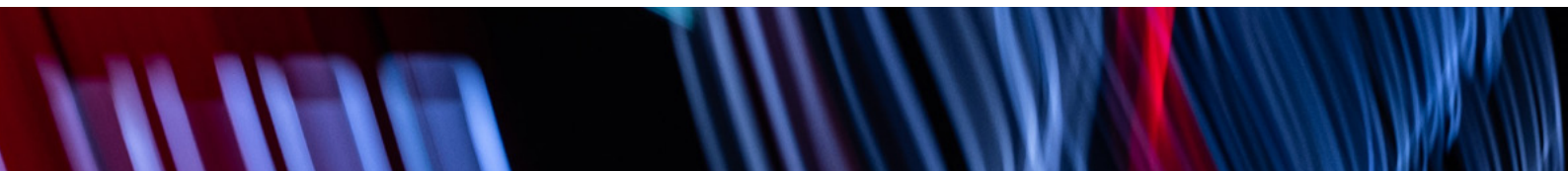
Alexander Voss
Technology and Human Rights Fellow
Carr Center for Human Rights Policy

ABSTRACT

As the products of the IT industry have become ever more prevalent in our everyday lives, evidence of undesirable consequences of their use has become ever more difficult to ignore. Consequently, several responses ranging from attempts to foster individual ethics and collective standards in the industry to legal and regulatory frameworks have been developed and are being widely discussed in the literature. This paper instead makes the argument that currently popular methods of software engineering are implicated as they hinder work that would be necessary to avoid negative outcomes. I argue that software engineering has regressed and that introducing rights as a core concept into the ways of working in the industry is essential for making software engineering more *rights-respecting*.

ACKNOWLEDGEMENTS

I would like to thank the Carr Center for accepting me into their Technology Fellows Program and its fellows for the numerous discussions that have been invaluable in shaping my views on the relationship between human rights and technologies as well as the context and methods of their production. Special thanks to Sushma Raman and Mathias Risse for their support and encouragement as well as Laryssa Da Silveira, Rachel Harris and Philip Hamilton for their help managing the proofing and publication process. Many thanks also to Brandie Nonnecke and Vivek Krishnamurthy for their constructive comments on the paper. I am indebted to Matt Ritter for his insightful comments around the question of how rights could ever "rise above the line" in the backlog. I also want to thank Bob Wyman for his comments on the right to freedom of opinion and expression as well as on the level of influence that other iterative process models had on software engineering before the advent of agile approaches.



Introduction

As the products of computer science and the IT industry have become ubiquitous in our lives, we have drawn great benefits from the availability of planet-wide networking, of cheap, mass-marketed devices, and, recently, of machine learning systems powered by large-scale distributed systems. By now, however, it has also become abundantly clear that the immense benefits we have reaped are accompanied by myriad negative consequences. The recent surge in machine learning applications in particular has produced a constant stream of examples of negative outcomes documented in the academic and trade literature as well as news headlines.

Most authors writing about these negative consequences agree that these are not attributable to "bugs" that are unfortunate but ultimately fixable. As I will discuss below, they reflect systemic problems. Authors disagree to some extent on the causes but many point to business models, the "bro culture" in the industry, a lack of diversity more generally, and the influence of wider inequities in society. What they largely agree on is that it is not "just a few bad apples" (though some particular apples get mentioned quite often) but that, instead, there are mechanisms at play that do not depend on individual actors misbehaving to cause the problems we are experiencing.

In this paper, I argue that the *methods by which we develop systems* matter to outcomes. They do not just determine if a project or product is successful, whether it serves the stated purpose, and whether it improves the "bottom line." Our choice of methods also affects human rights outcomes for stakeholders. I will argue below that software engineering has regressed as methods have become overly focused on the continuous delivery of new functionality at the expense of overarching and cross-cutting concerns. From this critique, I develop the notion of *rights-respecting software engineering* and outline what it would take to develop methods that make an explicit representation and consideration of rights possible in the development of software products and services.

Negative Outcomes

While machine learning systems are attracting a lot of attention at the moment, much more mundane systems can also give rise to a range of negative outcomes. Illustrative examples can be found in Eubanks' book on automating inequality,[1] which details the impact of workflow management technologies in welfare administration, Hartzog's book on design and privacy,[2] or in the work of Meyer and Wachter-Boettcher,[3] who point out that the humble online form can lead to negative outcomes for those whose data do not match the expectations of the form's designers.

One whole class of designs that are simple in nature but create negative outcomes for users on a massive scale due to their wide adoption[4] are "dark patterns," which are "deceptive, manipulative, and coercive practices to encourage certain patterns of use and discourage others."[5] Dark patterns serve corporate interests at the expense of the interests of users. Examples are user consent banners that make it much easier to consent to tracking than not to[6] and online shops displaying countdown timers suggesting that an offer is about to come to an end.[7]

[1] Virginia Eubanks, *Automating Inequality: How High-Tech Tools Profile, Police and Punish the Poor* (New York, NY: St. Martin's Press, 2018).

[2] Woodrow Hartzog, *Privacy's Blueprint: The Battle to Control the Design of New Technologies* (Cambridge, Massachusetts: Harvard University Press, 2018).

[3] Eric Meyer and Sara Wachter-Boettcher, Design for Real Life, 1st edition (A Book Apart, 2016); Sara Wachter-Boettcher, *Technically Wrong: Sexist Apps, Biased Algorithms, and Other Threats of Toxic Tech* (New York, NY: W.W. Norton & Company, Inc, 2018).

[4] Arunesh Mathur et al., "Dark Patterns at Scale: Findings from a Crawl of 11K Shopping Websites," Proc. *ACM Hum.-Comput. Interact.* 3, no. CSCW (November 2019), https://doi.org/10.1145/3359183; Hartzog, *Privacy's Blueprint: The Battle to Control the Design of New Technologies.* See also https://webtransparency.cs.princeton.edu/dark-patterns/.

[5] Colin M. Gray and Shruthi Sai Chivukula, "'That's Dastardly Ingenious': Ethical Argumentation Strategies on Reddit," *Proceedings of the ACM on Human-Computer Interaction* 5, no. CSCW1 (April 2021), https://doi.org/10.1145/3449144. p.69.

[6] Colin M. Gray et al., "Dark Patterns and the Legal Requirements of Consent Banners: An Interaction Criticism Perspective," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21 (New York, NY, USA: Association for Computing Machinery, 2021), https://doi.org/10.1145/3411764.3445779.

[7] Arvind Narayanan et al., "Dark Patterns: Past, Present, and Future: The Evolution of Tricky User Interfaces," *Queue* 18, no. 2 (April 2020): 67–92, https://doi.org/10.1145/3400899.3400901.

There also seems to be agreement that while problems with negative outcomes of the use of IT are not new[8] (witness Kling,[9] Weizenbaum,[10] and many others), the degree to which IT permeates our lives as well as the increased use of automated decision-making mean that the impact is growing.[11] Moreover, the impact is unevenly distributed, with those suffering from historical injustices and persistent inequities being the most affected.

Noble speaks of technological redlining[12] when she describes the role of algorithms in racial profiling, drawing a comparison with the redlining practices that further segregated the US[13] by systematically discriminating against people based on their place of residence. Scholars like Alexander[14] and Roithmayr[15] point out how racial discrimination and inequalities are re-produced despite the advances of the era of reconstruction and the civil rights movement. While explicitly race-based policies were replaced by discriminatory practices that are not overtly racist, these practices also have a devastating effect on the populations they target. Jim Crow laws were replaced by a range of practices that Alexander calls "The New Jim Crow."[16] Following this theme, Benjamin describes the role that technology plays in reinforcing racial inequities the "New Jim Code."[17]

For better and for worse, technologies are often used as ways to implement policy agendas. In her book *Automating Inequality*, Eubanks describes how technologies are used in the administration of welfare and in other areas to police and punish the poor.[18]

Uses of computer technologies to administer welfare are not new, of course. Kling wrote in 1978 about how "automated information systems may serve several uses simultaneously."[19] Systems that are advertised to increase efficiencies through better inter-agency coordination, better use of resources, improved delivery of services soon also get used to "catch 'welfare cheaters' or [to help] police obtain current addresses of wanted persons."[20] Kling points out that the introduction of one system he studied had more to do with"administrative attractiveness" and the ability of administrators to access feder-

[8] Mar Hicks, "When Did the Fire Start?," in *Your Computer Is on Fire*, ed. Thomas S. Mullaney et al. (Cambridge, MA: The MIT Press, 2021).

[9] Rob Kling, ed., *Computerization and Controversy: Value Conflicts and Social Choices*, 2nd ed. (San Diego: Academic Press, 1996).

[10] Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation* (San Francisco: W. H. Freeman, 1976).

[11] One might argue that the growth is exponential given the exponential growth of many sectors of IT use and the fact that multiple factors are at play. However, any significant growth should be sufficient to cause us to consider the arguments laid out in this paper.

[12] Safiya Umoja Noble, *Algorithms of Oppression: How Search Engines Reinforce Racism* (New York: New York University Press, 2018).

[13] Richard Rothstein, *The Color of Law: A Forgotten History of How Our Government Segregated America*, First edition., Democracy and Urban Landscapes (New York; London: Liveright Publishing Corporation, a division of W.W. Norton & Company, 2017).

[14] Michelle Alexander, *The New Jim Crow: Mass Incarceration in the Age of Colorblindness*, Rev. ed. (New York, NY: [Jackson, TN]: New Press; Distributed by Perseus Distribution, 2012).

[15] Daria Roithmayr, *Reproducing Racism: How Everyday Choices Lock In White Advantage* (New York: NYU Press, 2014).

[16] Alexander, *The New Jim Crow: Mass Incarceration in the Age of Colorblindness.*

[17] Ruha Benjamin, *Race after Technology: Abolitionist Tools for the New Jim Code* (Newark: Polity Press, 2019).

[18] Eubanks, *Automating Inequality: How High-Tech Tools Profile, Police and Punish the Poor.* p.12.

[19] Rob Kling, "Automated Welfare Client-Tracking and Service Integration: The Political Economy of Computing," *Communications of the ACM* 21, no. 6 (1978): 484–93. p. 485

[20] Kling. p. 485.

al funds than it had to do with enabling the work in the welfare agencies themselves that might have helped to improve delivery of services to those in need.[21]

It seems fair to conclude that the unprecedented application of information technologies in ever more domains of life has unintended negative consequences brought about by incompetence, ineptitude, carelessness, or simple oversight, but also by reproduction of systemic inequities and injustices. For those most vulnerable to rights violations, these negative consequences add to the burden they carry and can substantially impact their lives. It is thanks to authors like Noble, Benjamin, and Eubanks (to name a few) that these consequences reach our collective consciousness.

With the increasing impact of automated decision-making and the increasing role of new media in society, matters are arguably coming to a head. The now regular pictures of the captains of the IT industry having to answer before Congress show that the sense of urgency has spread beyond a small circle of activists and academics. In the words of Mullaney, "the time for equivocation is over."[22]

Negative Outcomes, Experiences, and Rights

Negative outcomes range from the mundane, even trivial and merely annoying, to those with serious consequences for life, bodily and mental health, economic welfare, privacy, liberty, or a person's ability to fully participate in society and democratic processes. Some of these problems might be described as merely poor "user experience"[23] but others constitute clear human rights violations. A pedestrian run over by an automated car is not just having a bad day in our technicized society. Someone being denied health insurance because they could not file a document on time and did not have a caseworker to talk to because the welfare administration system was centralized[24] is also not just suffering from poor "user experience."

"Job losses, migration, ill health, bereavements, or simply the onset of old age are things most of us experience that can fundamentally change how we experience the outcomes of our interactions with IT systems."

The boundary between both categories is blurry and difficult to define as well as difficult to adjudicate even in concrete cases. Does, for example, the experience of transgender people of being more routinely selected for secondary screening in airport security areas constitute a mere poor experience? As Sasha Costanza-Chock points out, the answer may partly depend on whether there are other aspects of their lives that make them vulnerable or whether, conversely, they carry a privilege based on their citizenship and social status.[25]

It is not unusual that the importance of a right becomes clearly visible only when we consider situations in which we might be vulnerable or when we begin to understand the circumstances that make others vulnerable. Job losses, migration, ill health, bereavements, or simply the onset of old age are things most of us experience that can fundamentally change how we experience the outcomes of our interactions with IT systems.

The debate about transgender rights shows how changes in society may well lead to changes in what we see as a right of a citizen or as a human right. Transgender rights are not generally incorporated into human rights frameworks yet, but we can imagine a future in which this has changed.

[21] Kling. p. 488.

[22] Thomas S. Mullaney, *Your Computer Is on Fire*, ed. Thomas S. Mullaney et al., *Your Computer Is on Fire* (Cambridge, MA: The MIT Press, 2021). p. 7.

[23] e.g., Rex Hartson and Pardha S Pyla, *The UX Book: Process and Guidelines for Ensuring a Quality User Experience, UX Book, The* (San Diego: Elsevier Science & Technology, 2012).

[24] Eubanks, *Automating Inequality: How High-Tech Tools Profile, Police and Punish the Poor.*

[25] Sasha Costanza-Chock, *Design Justice: Community-Led Practices to Build the Worlds We Need, Information Policy* (Cambridge, MA: The MIT Press, 2020).

"A pedestrian run over by an automated car is not just having a bad day in our technicized society.

Someone being denied health insurance because they could not file a document on time and did not have a caseworker to talk to because the welfare administration system was centralized is also not just suffering from poor "user experience."

What is more, as Schulz and Raman[26] point out, not all rights are created equal. Even rights formally established in the UN Universal Declaration of Human Rights,[27] the UN's International Covenant on Economic, Social, and Cultural Rights,[28] or by national laws do not all have the same status. For example, they carry different weights when human rights courts have to adjudicate cases in which one person's right to A conflicts with another's right to B. They also have different levels of public support. At any point in time there is likely to be a gap between formally defined rights and the lived experience of rights in society. At times, formal rights may be established but poorly enforced. At other times, societies may be ahead in how they respect and promote rights, even before they are codified in laws and international treaties.

London protests against war in Ukraine, 2022

What also changes is who we take to be duty bearers. The traditional view is that it is states and their governments that have duties with regard to (human) rights. Increasingly, however, the role that other (trans-)national institutions and corporations play in affecting rights is recognized and, consequently, the view has widened to assign duties to actors beyond the nation state. These duties may be very different from those of states.[29] Human rights frameworks are translated into legislation at the national and supra-national level that corporations are subject to. In addition, the UN Guiding Principles on Business and Human Rights[30] provide a soft-law framework for regulating the duties of both states and businesses with regard to the impact that business activities can have on human rights.

In light of the breadth of rights impacts that non-state actors have, it seems fair to say they must not limit themselves to mere compliance with existing legislation, which is all too often playing catch-up with the new realities of our technicized world. The war unleashed by Vladimir Putin against the state of Ukraine and its people has, again, highlighted the complex intermingling of powers reserved by states and those held de-facto by private corporations and other non-state actors. The conflict has shown how vulnerable Western economies have become by their dependence on fossil fuel[31] as well as their investments in countries with worrying human rights records.[32]

The IT industry has yet to come up with convincing answers to the problem of amplification of mis- and dis-information through its platforms, especially answers that do not at the same time run the risk of negatively impacting other rights such as freedom of opinion and expression. Achieving a balance between the right to freedom of opinion and expression and other human rights is a significant challenge both for the industry and legislators as the 2021 report on "Disinformation and Freedom of Opinion and Expression" by the UN Special Rapporteur illustrates.[33]

Responses to IT's Negative Consequences

Clearly, something has to change and so it is worthwhile to consider the range of possible responses to the negative outcomes produced by the IT industry. We can categorize them according to where they locate the root of the problem, what they suggest should be done about it, and who should address it. One possible categorization is shown in Table 1.

[26] William F. Schulz and Sushma Raman, *The Coming Good Society: Why New Realities Demand New Rights* (Cambridge, MA: Harvard University Press, 2020).

[27] *Universal Declaration of Human Rights* (New York: United Nations, 1997).

[28] "International Covenant on Economic, Social and Cultural Rights" (United Nations, 1966), https://www.ohchr.org/en/professionalinterest/pages/cescr.aspx.

[29] Samantha Besson, "The Bearers of Human Rights' Duties and Responsibilities for Human Rights: A Quiet (r)Evolution?," Social Philosophy & Policy 32, no. 1 (2015): 244–68.

[30] "Guiding Principles on Business and Human Rights" (United Nations, April 2011), http://www.ohchr.org/Documents/Publications/GuidingPrinciplesBusinessHR_EN.pdf.

[31] Leif Wenar, *Blood Oil: Tyrants, Violence, and the Rules That Run the World* (Oxford; New York: Oxford University Press, 2016).

[32] Adam Satariano, "Russia Intensifies Censorship Campaign, Pressuring Tech Giants," *The New York Times*, February 26, 2022, https://www.nytimes.com/2022/02/26/technology/russia-censorship-tech.html.

[33] Irene Khan, "Disinformation and Freedom of Opinion and Expression" (United Nations Human Rights Council, April 2021), https://www.ohchr.org/en/documents/thematic-reports/ahrc4725-disinformation-and-freedom-opinion-and-expression-report.

Table 1

Types of Responses to Negative Impacts of (Information) Technologies

	CATEGORY	ACTOR	WHAT CHANGES
1	Human rights treaties	Supra-national organizations	International human rights frameworks
2	Legislation	State	Rules under which industry operates
3	Industry regulation	State	Rules under which industry operates
4	External activism	Civil Society	Market conditions; external pressure
5	Self-regulation	Companies	Rules under which industry governs itself
6	Diversity	Companies	Composition of the workforce
7	Professional licensing and standards	Professional associations	Entry to the profession controlled; standards of practice established
8	Ethics	Professional associations, Individual workers	Guidance for individuals; internal resistance to state of the industry; consideration of design decisions
9	Internal Activism	Individual workers	Internal resistance to state of the industry
10	Cultural Change	Education system; industry; professional associations	Attitudes and the intellectual wherewithal to deal with questions of the role of technology in society
11	Methods Change	Companies and their workers	Ways of working in the industry

It is the last category that I wish to argue has been neglected so far. Changes in the methods used in software engineering are essential if we wish responses 1-10 to be effective in the long term and across the board.[34] After all, it does not matter how much pressure is applied to the IT industry internally or externally, in the end changes are needed not just in policies and business practices but also in the way that technologies are designed and configured. However, as I will argue below, there is currently a disconnect between the predominant methods used to develop software and over-arching questions of the impact these systems and services have on people and on wider society.

There is currently much debate about the importance of ethics in the tech sector and of a culture change in the industry. While ethics education and the establishment of ethical guidelines are no doubt important, they can only be a partial answer to the issues we face as they leave a gap between ethical considerations and their translation into practice. As Green observes,[35] there is "a lack of mechanisms to enact or enforce the espoused [ethical] principles." Likewise, a reconsideration of the culture of computing and changes to make the industry less toxic and more inclusive will not in themselves change much if we do not have the wherewithal to translate insights about the social impact and considerations of a broader set of values into actual technical change. Connolly[36] calls for soul searching in computing departments:

> *"[…] because computing as a discipline is becoming progressively more entangled within the human and social lifeworld, computing as an academic discipline must move away from engineering-inspired curricular models and integrate the analytic lenses supplied by social science theories and methodologies."*

There is much to be gained by following this call and integrating social science methods in computing education and practice. However, there is an open question as to just what role social research methods could play in computing and what methods we should choose for what purposes. Methods, after all, do have politics and have a social life[37] in that they emerge from the social world and at the same time shape how we understand it. Our choices of how to engage with the vast array of research traditions in the social sciences will have an impact on the outcomes we achieve. What is more, the history of fields at the intersection[38] between computing and the social sciences has shown that there is often no straightforward way to translate insights from the application of social science methods to concrete "implications for design" and that, in fact, looking for such straightforward implications as inputs to the business of computing may be misguided.[39] Again, as in the case of tech ethics, there is a gap between ways to study and reason about the (potential) impact of technologies as well as our attempts to control negative impacts on the one side and the business of developing products, systems, and services on the other.

In the following, I show how recent developments in software engineering have widened this gap by constraining methodological choices to a set of methods for increasing the cadence of development and the delivery of 'features' at the cost of longer-term interests. This represents a regression compared to what had already been achieved in software engineering and in fields we might collect under the umbrella of "human-centric computing." The specific question I will then pursue is how we might narrow the gap again and develop what we might call *methods for rights-respecting software engineering*.

"Traditional" Software Engineering

Until the early 2000s, software engineering approaches tended to follow a phased approach in which each phase of work is a preparation for the next. The idea was to manage the risk of wasting work further downstream or jeopardizing the project outright by investing up-front effort into analysis and design activities, producing documentation that would guide the implementation of the system. Another aspect that made these approaches popular was that they provided a common language for measuring progress—albeit at a very coarse level—and provided a chance to exercise control over projects by assessing their readiness to move to the next phase at predefined milestones. The emphasis on planning lead to the description of these models as "plan-based."

[34] With the possible exception of (1) when the use of some technologies is banned and the need for better methods becomes superfluous. Facial recognition in the public domain may be such a case.

[35] Ben Green, "The Contestation of Tech Ethics: A Sociotechnical Approach to Technology Ethics in Practice," *Journal of Social Computing* 2, no. 3 (2021): 209–25, https://doi.org/10.23919/JSC.2021.0018.

[36] Randy Connolly, "Why Computing Belongs within the Social Sciences," *Communications of the ACM* 63, no. 8 (July 2020): 54–59, https://doi.org/10.1145/3383444. p.55.

[37] Mike Savage, "The 'Social Life of Methods': A Critical Introduction," Theory, Culture & Society 30, no. 4 (2013): 3–21, https://doi.org/10.1177/0263276413486160; E. Ruppert, J. Law, and M. Savage, "Reassembling Social Science Methods: The Challenge of Digital Devices," *Theory, Culture & Society* 30, no. 4 (July 1, 2013): 22–46, https://doi.org/10.1177/0263276413484941.

[38] Such as Human-Computer Interaction (HCI) or Computer Supported Cooperative Work (CSCW).

[39] Paul Dourish, "Implications for Design," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06 (New York, NY, USA: ACM, 2006), 541–50, https://doi.org/10.1145/1124772.1124855.

One of the earliest versions of plan-based development appears in a paper by Royce,[40] first presented in 1970, in which he also points out that a purely linear approach is unlikely to work since software that interacts with humans cannot be fully analyzed in advance, unlike software that merely performs a set of calculations. We might add to this that if development takes any serious amount of time, chances are that some requirements change in that timeframe. As a consequence, Royce advocated for the inclusion of *feedback* from work "downstream" into earlier phases.

It is unfortunate that Royce's paper became known as the source of the "waterfall" model, an apt description of the linear model that appears at the beginning of the paper only to be critiqued. It is even more unfortunate that the "waterfall model" became popular and a reference point for talk about software process models. This led to many projects that failed because of the kinds of problems that Royce foresaw in 1970. As I will discuss below, it also led to a rather summary dismissal of a phased approach to software development subsequently that I will argue has equally negative consequences—not for productivity, but for the quality and social acceptability of outcomes.

Before discussing this, however, it is worth noting that software process models were, in fact, developed that included an explicitly iterative approach that uses preliminary work of limited scope in order to test assumptions and decisions made before committing to a full-scale implementation. An example of such a model is Barry Boehm's "Incremental Commitment Spiral Model,"[41] first introduced in 1986 and subsequently refined. It is an excellent example of a process model that employs a phased approach but aims to avoid the problems the "waterfall" gives rise to. In particular, it introduces an iterative approach that avoids premature commitment to full-scale implementation as well as allowing teams to learn from preliminary work. A central idea is that risk identification and mitigation should guide the process. Crucially, and of particular interest in the context of human rights, Boehm and his co-authors emphasize the need to achieve and maintain alignment between the interests of many stakeholders through a process of negotiation and satisficing.

The model by Boehm et al. is only one example of a phased process model that incorporates iteration and incremental work to overcome the problems of a purely linear model. Its focus on managing project risks and multi-stakeholder alignment makes it particularly interesting for rights-respecting software engineering. While models such as Boehm's have influenced the development of agile approaches, these have taken up the idea of iterative development but have tended to focus it on short iteration and feedback cycles in the development process. Agile enthusiasts tend to shorten their treatment of the prior history of software engineering to a comparison with the waterfall model, which deletes from the discourse a lot of what was and is useful in earlier process models. I will argue that the result of this is that the kinds of longer-running concerns that Boehm's model foregrounds are systematically neglected.

[40] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," in *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87 (Washington, DC, USA: IEEE Computer Society Press, 1987), 328–38.

[41] B. Boehm, "A Spiral Model of Software Development and Enhancement," *SIGSOFT Software Engineering Notes* 11, no. 4 (August 1986): 14–24, https://doi.org/10.1145/12944.12948; Barry Boehm et al., *The Incremental Commitment Spiral Model: Principles and Practices for Successful Systems and Software*, 1st ed. (Addison-Wesley Professional, 2014).

The "Agile" Turn

A turning point in the history of software engineering was the emergence of a new breed of methods in the late 1990s and early 2000s and the release of the Manifesto for Agile Software Development.[42] The manifesto gave voice to frustrations with traditional plan-based software development methods that had been brewing for a while. Software development had long been plagued by a sense of crisis as too many projects overran in cost or time or both. Too many projects were getting canceled or led to software that was not fit for purpose.

Practitioners realized that up-front planning is difficult to get right and that it is better to work incrementally to reduce the risk that problems are detected only once a system is launched into full production. Projects risked being overtaken by changing requirements and circumstances if they were planned with up-front analysis, followed by a long implementation period, and testing and deployment at the end of the process. There was also a sense that software engineering was hamstrung by placing too much of an emphasis on the production of artifacts that were not contributing directly to the development of a working system. Examples include project plans, long-form requirements documents, as well as detailed architectural and design specifications. The feeling was that suitably crafted code could replace many of these and that an incremental, iterative development and delivery approach could replace up-front planning, reducing planning horizons to weeks instead of months or years.

> "Software development had long been plagued by a sense of crisis as too many projects overran in cost or time or both."

Instead of seeking to establish contracts for the delivery of software and specifying the requirements in detail, parties should instead interact with each other throughout the software process to guide development by generating feedback on each of a series of incremental versions of the software, produced in short, iterative cycles. The manifesto expresses these changing values:[43]

Individuals and interactions *over processes and tools*
Working software *over comprehensive documentation*
Customer collaboration *over contract negotiation*
Responding to change *over following a plan*

In addition to these overall values, the authors of the manifesto also spelled out twelve principles.[44] They are clearly designed to address the problems that traditional software engineering practices were suffering from. However, a set of values and principles does not make a method for producing software, so it is not surprising that methods that were available at the time have come to represent what it means to be "doing Agile" more than the manifesto itself does.

Scrum

By far the most popular[45] of these "agile methods" today is Scrum,[46] developed by Ken Schwaber and Jeff Sutherland. Scrum defines a number of roles, events, and artifacts that guide software development activities. Its central artifact is the *Product Backlog*, which contains a prioritized list of work items and is managed by the *Product Owner*. The effort required to implement them is estimated by the developers and they are assigned a value by the product owner, to create a prioritized list of work items (valuable + easy = high priority). Work in Scrum happens in short *Sprints*, often two weeks long. The Scrum Guide[47] describes work items rather vaguely as "what is needed to improve the product" but they are usually *User Stories* that each represent a feature of the system. User stories are simple statements that do not flesh out all the requirements for a feature but serve as reminders that a further discussion of the work item needs to

[42] Kent Beck, et al., "Manifesto for Agile Software Development," https://agilemanifesto.org/.

[43] Kent Beck, et al., "Manifesto for Agile Software Development," https://agilemanifesto.org/.

[44] Kent Beck, et al., "Manifesto for Agile Software Development," https://agilemanifesto.org/principles.html.

[45] Digital.AI, "15th Annual State of Agile Report: Agile Adoption Accelerates Across Enterprise," https://digital.ai/resource-center/analyst-reports/state-of-agile-report.

[46] Jeff Sutherland and J. J. Sutherland, *Scrum: The Art of Doing Twice the Work in Half the Time* (Westminster: Crown/Archetype, 2014); Kenneth S Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, The Addison-Wesley Signature Series (Addison-Wesley Professional, 2012). Also see https://scrumguides.org/.

[47] Ken Schwaber and Jeff Sutherland, "The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game," November 2020, https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf. p. 8.

take place between the product owner and developers during the *Sprint Planning meeting*.[48] The development work happens during the *Sprint*, a time-boxed period during which developers are protected from any changes in requirements as well as from changes in the composition of the team and its work environment. At the end, feedback on the developed increment is provided in the *Sprint Review* and possible improvements to working practices are discussed in the *Sprint Retrospective*.

There are a number of things to note about the process. The first is that the product owner is the source of work items, though the Scrum Guide notes that the team can invite others into the process to provide advice.[49] There is no mention of systematic stakeholder involvement. Nor is there any mention of where the product owner gets work items and their description from. They are treated, effectively, as an oracle that abstracts away any user research done, any work to define product features and work items. They also play a central role in providing feedback on the developed increment but it is not clear by what criteria they evaluate the increment or what methods they would use.

The Manifesto for Agile Software Development emphasizes "customer collaboration." In Scrum, this gets reduced to the role of the product owner. The Scrum Guide states clearly that the product owner must be a single person, not a committee, and that their task is to represent the needs of many stakeholders.[50] It does not mention what form any consultation, user research, or stakeholder engagement might take. There is no mention either of requirements or how they should be documented other than there should be a definition of when a feature implementation is "Done."

There is no suggestion of what is to be done about requirements that are difficult to specify precisely. Requirements that are notorious are those that state that a system should not do something as well as those relating to quality attributes such as usability. The emphasis on individual features also makes it difficult to express cross-cutting concerns such as security, dependability, or data protection. Following the dictum of "working software over comprehensive documentation," Scrum also does not say much about system architecture and

design, topics that fill their own chapters in traditional software engineering textbooks. For example, it is noticeable that Ian Sommerville's latest book, *Engineering Software Products*,[51] is missing a lot of content from his long-running textbook *Software Engineering*,[52] which reached its 10th edition. It seems that going with the times means ignoring lessons from the past. Both books are available but which one is going to generate more sales going forward?

That the Scrum Guide itself leaves a lot unspecified is perhaps not surprising since it is the most general document. The problem is that it is not alone in neglecting topics that would be of immense value if we wanted to practice rights-respecting software engineering. Scrum and its literature do not just leave out important topics, they *eschew* anything that would require some form of up-front work or continuous attention. This potentially affects requirements engineering, system architecture, consistent design of user interfaces, attention to user experience, dependability, security, and maintainability. Meyer critiques this, stating that:

> *There is, however, no argument for shunning the normal engineering practice—the practice, in fact, of any rational endeavor—of studying a problem before attempting to solve it, and of defining the architecture of the solution before embarking on the details.*[53]

[48] Mike Cohn, *User Stories Applied: For Agile Software Development*, 1st edition, Addison-Wesley Signature Series (Boston: Addison-Wesley, 2004). p.40.

[49] Schwaber and Sutherland, "The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game." p. 8.

[50] Schwaber and Sutherland. p. 6.

[51] Ian Sommerville, *Engineering Software Products: An Introduction to Modern Software Engineering* (Pearson, 2020).

[52] Ian Sommerville, *Software Engineering*, 10th ed. (Pearson, 2016).

[53] Bertrand Meyer, *Agile!: The Good, the Hype and the Ugly* (Springer International Publishing : Imprint: Springer, 2014).

As a result, the process that seems to work "in the small" during development—build something, evaluate, correct problems—gets scaled up to the whole system. This is why half-baked ideas are let loose on an unsuspecting public while up-front work as well as ongoing concerns are sidelined. This does not just affect up-front design and architectural considerations but such things as the consideration of the key concepts inherent in a piece of software and the question of whether they meet the needs and expectations of the intended users. Daniel Jackson writes:[54]

> *…the key decisions that determine whether a software application or system is useful and fulfills its users' needs lie elsewhere, in the kind of software design in which the functionality and the patterns of interaction with the user are shaped. These big questions were at one time more central in computer science. […] But as time passed, they became less fashionable, and they faded away. Research in software engineering narrowed....*

There has been debate around the tendency of people "doing Agile" to avoid defining an explicit software architecture[55] but the lack of attention to requirements specification has only recently been criticized. Meyer[56] undertook a critical analysis of methods of agile software engineering and concludes on the topic of requirements:

> *The resulting systems are narrowly geared to the specific user stories that have been identified; they often do not apply to other uses; and they are hard to adapt to more general requirements. User stories are no substitute for a system requirements effort aimed at defining the key abstractions and the associated operations (the domain model) and clearly separating machine and domain properties.*

It seems fair to say that with the wide adoption of Scrum, software engineering has experienced a regression. To be fair, there are many books on the market that further elaborate on Scrum and give useful advice but, and this is crucial, the literature is quite scattered, and it is difficult to ingest as well as to teach[57]. There is, to my knowledge, no book on the market that does for requirements what George Fairbanks' book *Just Enough Architecture*[58] does for system architecture. Fairbanks argues that instead of eschewing explicit work on architecture entirely, the amount of effort invested should be proportional to the risk involved in getting it wrong. Likewise, many systems built today may not require a full-scale human rights impact assessment but it seems that we have lost the ability to distinguish those from the ones that do and to take appropriate action.

A/B Testing

What is more, instead of conducting user research that would seek to answer what behavior stakeholders might expect from a system or how outcomes might affect them, companies often develop different versions of a feature and roll them out as part of an A/B test[59] where users are randomly assigned to one version and their behavior[60] is observed as one might observe wild animals' behavior. The version that leads to desirable user behavior is then carried forward. What is desirable, of course, is defined by those who control the system. It is not something people more broadly have input on and have agreed to. A/B testing is enabled by methods of agile software development that allow for large numbers of increments to be developed in relatively little time. The practices are, in fact, mutually supportive as A/B testing can serve to provide the rapid feedback in fortnightly cycles that "Agile" demands. This integration is enabled by recent advances in software engineering to support continuous testing and deployment.[61] Setting up the whole package of agile software development,

[54] Daniel Jackson, *The Essence of Software: Why Concepts Matter for Great Design* (Princeton: Princeton University Press, 2021). p.9

[55] George Fairbanks, *Just Enough Software Architecture: A Risk-Driven Approach* (Boulder; Marshall & Brainerd, 2010).

[56] Meyer, *Agile! The Good, the Hype and the Ugly.*

[57] I have presented the subject of software engineering at the University of St Andrews for about a decade before recently leaving to do more software engineering in practice and it has been increasingly hard to shift students' focus away from the Scrum Guide and towards topics like requirements engineering and stakeholder involvement

[58] Fairbanks, *Just Enough Software Architecture: A Risk-Driven Approach*.

[59] Caitlin Tan, Rochelle King, and Elizabeth Churchill, *Designing with Data*, 1st ed. (Sebastopol; O'Reilly Media, Inc, 2017).

[60] Shoshana Zuboff, *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*, First edition. (New York: PublicAffairs, 2019). p.204ff.

[61] Jez Humble and Dave Farley, *Continuous Delivery: A Handbook for Building, Deploying, Testing and Releasing Software*, 1st edition (Addison-

continuous integration, and A/B testing is a significant technical and organizational challenge, which is perhaps why we see these practices mainly at the largest IT companies. However, there are efforts to commodify them and make them more widely available. The smart money bets that we will see much more of this practice in the near future. The practice of online experimentation is certainly not uncontroversial, as the debate about Facebook's Emotional Contagion Experiment [62]has shown, which raised serious questions about the ethics of online experimentation. Unlike this case, which came to public attention because researchers involved published their findings, much online experimentation done in the IT industry goes unremarked.

> "...the values [of Agile] have been totally lost behind the implementation."
>
> —Martin Fowler

We might stop at this point and ask if the large-scale, routine experimentation on unsuspecting users is not the antithesis of meaningful stakeholder engagement and of any decent attempt to elicit and negotiate user requirements. Online experimentation relies on measures defined in advance by the experimenter to measure user behavior observable through the software artifact. Without dwelling on the topic, it seems clear that while this method may work well to establish whether people react to advertising or how long it takes them to perform a given task, it is not suitable to answer more complex questions about the social world and the impact a system has on its users and their rights. It is thus an example of a social science method that is not useful for rights-respecting software engineering. It is not a way to elicit and negotiate requirements but merely a way to validate designs against very narrow success criteria defined by the developers of technologies.

Critiques of Scrum & the Agile Industry

It is not that there has not been a critique of what "Agile" has become, but as we will see the critique is somewhat different from what I have offered above. The world now has two decades of experience with agile software development. Since about 2015, when Dave Thomas, one of the authors of the manifesto, gave a talk entitled "Agile is Dead,"[63] there has been an increase in voices that criticize either the state of the industry that has developed off the back of the agile movement or that criticize common trends and practices associated with "Agile."

Thomas' critique is that "Agile" has been turned from an adjective into a noun, even one that gets capitalized and so is turned it into a thing that can be commercialized. What started out as an expression of values and principles offered to the world by a group of likeminded individuals has turned into a sizable industry that has commodified "Agile" into products like books, training courses, conferences, certificates, consultancy offerings, software products and services, market reports, and so forth. Martin Fowler calls it the "agile industrial complex."[64] Thomas suggests that, as a consequence, "the values have been totally lost behind the implementation."[65] The commodification has gone hand-in-hand with the establishment of "Agile" as a dogma that is difficult to ignore—for individuals and companies alike.

I would agree with Thomas that the spirit of the manifesto for agile software engineering has been lost in the implementation (Scrum). However, in contrast to him, I do not think that the values of the manifesto are just fine. After this quick look at Scrum and the critique of "Agile," it is time to revisit the four values.

Revisiting the Values

Thomas summarizes the sentiment of the authors of the manifesto at the time like this: "How can we cut down on all of the bullshit, basically, and just focus on writing software."[66] The word "bullshit" is a gloss that is worth unpacking. It is true that the degree of up-front planning and detailed documentation in traditional software engineering all too often was not helpful. However, the sentiment expressed here is rather close to the "move fast and break things" motto that Mark Zuckerberg

Wesley Professional, 2010).

[62] e.g., Evan Selinger and Woodrow Hartzog, "Facebook's Emotional Contagion Study and the Ethical Problem of Co-Opted Identity in Mediated Environments Where Users Lack Control," Research Ethics 12, no. 1 (2016): 35–43, https://doi.org/10.1177/1747016115579531; David Shaw, "Facebook's Flawed Emotion Experiment: Antisocial Research on Social Network Users," *Research Ethics 12*, no. 1 (2016): 29–34, https://doi.org/10.1177/1747016115579535.

[63] Dave Thomas, "Agile is Dead," https://www.youtube.com/watch?v=a-BOSpxYJ9M.

[64] Martin Fowler, "The State of Agile Software in 2018," August 2018, https://martinfowler.com/articles/agile-aus-2018.html.

[65] Dave Thomas, "Agile is Dead," https://youtu.be/a-BOSpxYJ9M?t=548 (at 9:08).

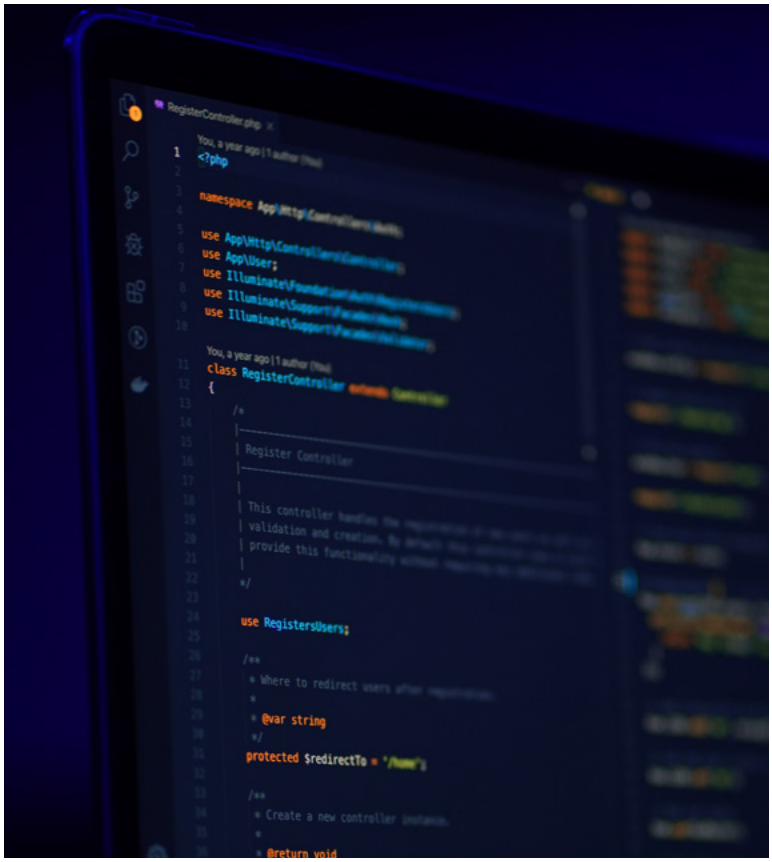
[66] Dave Thomas, "Agile is Dead," https://youtu.be/a-BOSpxYJ9M?t=548 (at 9:08).

popularized. I have argued above that the fathers[67] of agile software development have, perhaps inadvertently, cut down on some things that might have been worth preserving.

With a focus on the negative consequences of our widespread application of IT described in the introduction and with (human) rights issues in mind, we can take a critical look at how the way the values espoused in the manifesto have sown the seeds for what Scrum would become. This helps to understand how the IT industry has lost sight of important achievements in software engineering and other fields. A consequent reconsideration of values and methods might provide some of the wherewithal for dealing with the current crisis and for bending software development practices towards less dystopian futures.

Before starting, however, we need to look at the format that was used to express the values: "X over Y," with X being something they wished to endorse while Y was something valued in the past that they wanted to de-emphasize. They wrote, directly underneath the list of values: "while there is value in the items on the right, we value the items on the left more."[68] The authors were careful to note that they did not wish to throw out traditional practices of software engineering but such subtleties, it would seem, were lost in the implementation of what became "Agile," the thing that could be sold and that became the new dogma of software engineering. Here is the list of values again:

> **Individuals and interactions** *over processes and tools*
> **Working software** *over comprehensive documentation*
> **Customer collaboration** *over contract negotiation*
> **Responding to change** *over following a plan*



The emphasis on individuals and interactions was a reaction to the experience that the process models and software engineering tools of the day were sometimes holding up development. This was, to a large extent, due to the fact that process models of the day had been developed at a time when much software development was still bespoke development. There was also a lack of a clear leader in the field, with a multitude of options competing and struggling to replace the idea that software should be developed according to a strict sequence of analysis, implementation, and testing—the infamous "waterfall model."

Compared to the waterfall model, the shift towards reliance on small, closely knit teams bringing together the necessary skills required and working incrementally was certainly a sensible move. The problem is that, in the process, values and concerns that were embedded in the processes and tools were also thrown out.

The emphasis on working software over documentation is understandable but—and this is a big but—it is difficult to reason about a software system and the impact it will have on stakeholder groups by looking at code, no matter how well it is written. Documentation is not just a means for recording facts about a system but also a way to record decisions made and to establish accountability.

On the process side of things, process models that may have included notions of stakeholder involvement in development were replaced with the rather vague idea of "customer collaboration" and the "product owner," a central figure tasked to represent all stakeholders and their needs in the software process. Doing so adequately seems a herculean task at best. In light of questions of inequities and (human) rights it seems an impossible task for a single individual, especially if this individual is recruited from the same demographic as most of the IT industry.

The "working" part in "working software" also does an interesting job. It resonates with the decision to favor incremental delivery. If each short period of weeks rather than months is to produce a new increment, then longer-term work is easily sidelined. While the focus on obtaining feedback in Scrum is laudable, the short-term focus can lead to feedback being given on individual pieces, without evaluation of the larger whole or consideration of how features or design decisions may impact stakeholders. To do this longer-term work would mean stepping out of the model provided by Scrum.

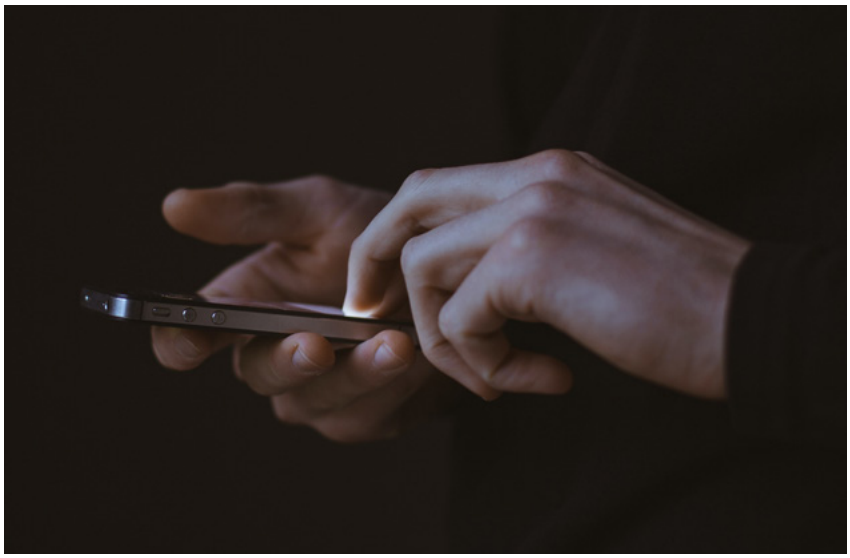
[67] There were no mothers around. The group that met was pretty homogeneous in many other respects. This makes one wonder if things could have been different?!

[68] Kent Beck, et al., "Manifesto for Agile Software Development," https://agilemanifesto.org/

The word "customer" represents a focus away from thinking of stakeholders, direct and indirect "users" of a system, and towards people who sponsor the development. This is no different than the alternative value listed, "contract negotiation." The manifesto may not have made things worse here, but an opportunity was certainly missed to re-orient software development to stakeholder needs. As a result, we now have A/B testing emerging as a major driver of decision making that determines what features a system should have. We can see many systems optimized for the needs of advertisers and their interest in "engagement" rather than for what people may want from the system.

"We can see many systems optimized for the needs of advertisers and their interest in 'engagement' rather than for what people may want from the system."

The examination of the values expressed in the manifesto shows that what I have called the regression in software engineering can be traced back to the values in the manifesto and is not just a consequence of the implementation through Scrum. The idea of rights-respecting software engineering is a call to critically reflect not just on the inadequacies of the dominant approach in software engineering today but to consider the values and principles of the agile manifesto itself and to revive some of what has been lost and sidelined in software engineering and in computing more generally.



Conclusions

The title of this paper asks whether we can move fast without breaking things. As in road traffic, the answer will likely depend on what we mean by "fast." None of us drive the fastest cars in the world to work and for good reasons. Such cars are rightly limited to racetracks and salt lakes where the risks are well managed. In most countries, there are restrictions on how fast one can go—for a variety of reasons.[69]

We may want to think about where it makes sense to move fast in software engineering and where some deliberation is required that will take time. For example, there is absolutely no reason not to work in small increments when implementing specific functionality. Being able to have a potentially releasable product after each sprint is also a good thing. At the same time, sprinting off in the wrong direction is costly and some upfront planning and due diligence will be required, unless our task is simply to burn through endless piles of venture capital.

So, are there alternatives to the "agile industrial complex," as Fowler calls it? The answer, fortunately, is that there are at least the beginnings of alternatives. The problem is that they are not widely enough promoted, see comparatively little uptake so far and have yet to be integrated into teaching and training. Both Ambler's *Disciplined Agile Delivery*[70] and the Essence Standard[71] emphasize the need to tailor software engineering approaches to the context in which software development takes place as well as to consider a broader view of the software lifecycle and its context.

That a significant number of established names[72] in software engineering speak out against the "agile-industrial complex" is encouraging. There is a unique opportunity to establish rights-respecting software engineering on the basis of this development. Many of the elements of what is required to establish rights-respecting software engineering practices can already be found in the body of knowledge of software engineering as well as in related fields such as human-computer interaction, social informatics, science and technology studies, and participatory design (to name a few). Table 2 recaps some of the critiques offered in this paper of the "agile industrial complex" and lists changes that would be required to establish more rights-respecting software engineering.

Of the areas listed, I would highlight the importance of developing a common language for representing rights throughout the process. Without such a language, bringing together people from the wide range of disciplines required will be impossible. A common language also serves to tie together the practices in the area listed. Making rights a central concept within software engineering would help us address the problems we face today in our technicized world. We need to re-discover and activate existing knowledge from a range of disciplines to lose the narrow focus on the work of implementing functionality and the mantra of speed that the "agile industrial complex" has created. The moment seems to be right to do this. In the words, again, of Mullaney: "the time for equivocation is over."[73]

[69] Even Germany is warming to the idea of a national speed limit. Practically, there are already limits on most stretches of the Autobahn.

[70] Scott Ambler, *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*, 1st edition (IBM Press, Pearson PLC, Upper Saddle River, NJ, 2012); Scott Ambler and Mark Lines, *Choose Your WoW: A Disciplined Agile Approach to Optimizing Your Way of Working*, 2nd ed. (Project Management Institute, Newton Square, PA, 2022).

[71] Ivar Jacobson et al., *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!* (Association for Computing Machinery and Morgan & Claypool, San Rafael, CA, 2019). For the standard itself see https://semat.org/en/essence-1.html.

[72] At least Dave Thomas, Martin Fowler, Ivar Jacobson, Bertrand Meyer and Scott Ambler have done so.

[73] Mullaney, *Your Computer Is on Fire*. p. 7.

Table 2

Problems with Current Software Engineering Methods Summarized and Implications for Rights-Respecting Software Engineering

DIMENSION	SCRUM / AGILE	PROBLEM	RIGHTS-RESPECTING
Cadence and focus	Individual features, often 14 days of work, planned just in time	Does not fit with overarching concerns; difficult to bring things into focus that are not features	Some up-front analysis; review at longer intervals at specific points in time, such as around major milestones and before release
Process model	Developer-centric	Ignores work that happens outside development team	Identify process elements outside narrow development focus, such as human rights impact assessment and integrate into wider process model
Source of requirements	From product owner	Lack of method for wider stakeholder engagement	Utilize wider range of elicitation methods including participatory design
Requirements specification	User-stories	Focus on functionality; individualized; difficult to capture things that are not features	Format for specifying rights as input to design and reviews needed
Documentation	Eschewed	No accountability for design decisions	Document where accountability is needed and in ways that are appropriate to the issue of concern
Prioritization	By vague concept of “value”	How to assign a “value” to respecting rights?	Projects require effective oversight and steering to avoid human rights being forever “under the line” in the backlog
Testing	Automation preferred	Feature testing can be automated; more difficult with other concerns.	Manual testing and review with appropriate tool support
Knowledge	Cross functional teams	Great, but not specific about what knowledges is required	Human rights experts unlikely to be part of the core development team; need process model to integrate (see above)
Stakeholder engagement	"Customer collaboration"	Unspecific as to what this looks like; unclear who “the customer” is; narrow focus on that persona	Wider engagement strategy that goes beyond treating stakeholders as sources of information or parties to be informed / consulted but empowers them to make decisions to protect their rights
Integration into wider business specification	Only via product manager	Reliance on product owner to navigate wider business processes	Integrate with business-level concepts, such as corporate social responsibility and responsible innovation

Carr Center for Human Rights Policy
Harvard Kennedy School
79 JFK Street
Cambridge, MA 02138

This publication was published by the Carr Center for Human Rights Policy at the John F. Kennedy School of Government at Harvard University

carrcenter.hks.harvard.edu

79 JFK Street | Cambridge, MA 02138
617.495.5819