The historical trajectories of algorithmic techniques: an interview with Bernhard Rieder

Michael Stevenson and Anne Helmond

ABSTRACT

Bernhard Rieder is Associate Professor of New Media and Digital Culture at the University of Amsterdam and a collaborator with the Digital Methods Initiative. His research focuses on the history, theory and politics of software and in particular on the role algorithms play in social processes and in the production of knowledge and culture. This includes work on the analysis, development, and application of computational research methods as well as investigation into the political and economic challenges posed by large online platform. In this interview, Michael Stevenson (MS) and Anne Helmond (AH) talk to Bernhard Rieder (BR) about his forthcoming book entitled *Engines of Order: A Mechanology of Algorithmic Techniques* (University of Amsterdam Press, 2020). In particular, Rieder discusses how the practice of software-making is "constantly faced with the 'legacies' of previous work" and how the past continues to operate into present algorithmic techniques.

Bernhard Rieder is Associate Professor of New Media and Digital Culture at the University of Amsterdam and a collaborator with the Digital Methods Initiative.[1] His research focuses on the history, theory and politics of software and in particular on the role algorithms play in social processes and in the production of knowledge and culture. This includes work on the analysis, development, and application of computational research methods as well as investigation into the political and economic challenges posed by large online platform. In this interview,[2] Michael Stevenson (MS) and Anne Helmond (AH) talk to Bernhard Rieder (BR) about his forthcoming book entitled *Engines of Order: A Mechanology of Algorithmic Techniques* (University of Amsterdam Press, 2020). In particular, Rieder discusses how the practice of software-making is "constantly faced with the 'legacies' of previous work" and how the past continues to operate into present algorithmic techniques.

MS & AH: First of all, congratulations on completing your book! If the pre-publication buzz on Twitter is any indication, this book will certainly find a large audience in Media Studies and beyond, and deservedly so. The book is also of particular interest to readers of *Internet Histories*, as it historicizes the various recommendation systems and other information ordering devices that are so central to Internet culture and the impact of new media and computational systems on society.

Importantly, the book focuses on a class of objects that you call "algorithmic techniques," which is distinguished from, say, specific ranking algorithms used for a search engine or social media platform. Before we turn to the connections between the book and our special issue on "Legacy Systems," could you walk us through this concept and discuss why it represents an important object for media scholars (and in particular historians of media and technology) to address? Could you tell us how this perspective is both different from contemporary critiques of "algorithmic bias" and can perhaps inform these debates from a different angle?

BR: Thanks! And yes, the concept of "algorithmic technique" is really central to the book and an attempt to find a way to deal with the increasingly sophisticated technologies that surround us. Let's face it, computers and software are incredibly complex, but we need some way of achieving a robust critical understanding of how something like Google's search engine or Facebook's newsfeed actually works. We can go some way towards this by scrutinizing their results for bias or by critically assessing how these mechanisms reflect the balancing acts these companies must play as "platforms" that mediate between users, content producers, and advertisers. But the technical principles at work should not be ignored and that is where the concept of algorithmic technique comes in. It takes some of its cues from Simondon's work and starts from the question how to conceive the "substance" of technology and, in particular, its "units". This sounds a bit complicated, but in the end, it is an attempt to find a middle ground between very broad categories such as computation or artificial intelligence on the one side and concrete programs or systems on the other.

The concept also aligns itself with technical practice and the relationship between technical knowledge and concrete material artifacts: if you open a textbook on machine learning or look at a programming library like Python's scikit-learn, what you will find is not some singular principle, but a whole range of techniques – from Bayes classifiers to neural nets – that do broadly similar things but which have their own "character", advantages and disadvantages, performance characteristics, and so forth. These techniques often have their own histories and draw on different heuristics and concepts. They have been laid out in scientific papers, studied, improved upon, tested, measured, and dissected. But they are still "broad" in the sense that there are variations, parameters, different ways to prepare inputs and outputs, and so forth. When a developer builds an actual system – and this is very much the situation the book thinks about – they will draw on existing techniques and associated knowledge, but they will then embed them within a working system, adapt and tune them, surround them with many other components that draw on their own historical lineages.

I hope that thinking about algorithmic techniques, their historical trajectories, and central characteristics helps media scholars take a step closer to what information and computer scientists are actually putting into the world, without having to fully engage

the extremely detailed and specialized meanderings of the full research landscape. There are thousands of research papers dedicated to describing, prodding, or extending the Bayes classifier, a simple machine learning technique that has been around for almost 60 years, and there are thousands of implementations that are used in all kinds of settings (see Rieder, 2017). It would be hard to engage all of this in detail, but there are certain basic characteristics that allow us to develop an understanding of what actually hides behind a loaded term like "artificial intelligence" in a way that is not a caricature. And from there we can develop a more nuanced appreciation of what something like "algorithmic bias" could look like and what kinds of effects it could have as part of a larger system. Simondon had the idea that machines "signify" by what they do (Simondon 2017), that they have meaning through their function or operation and the notion of algorithmic technique is maybe first and foremost an attempt to make that more broadly readable, to develop a conceptual vocabulary to talk more concretely about what machines do and how they do it.

MS & AH: How is the study of algorithmic techniques, in combination with Simondon's philosophy, distinguished from the search for basic or even universal principles of computation that give us a broad understanding of its effects, an effort that seems to stem from formalist media theory a la McLuhan? For example, your book could be understood in some sense as a critical response to the likes of Lev Manovich's earlier work on *The Language of New Media* (Manovich 2001), while closely aligning with Manovich's later work *Software Takes Command* (Manovich 2013). Perhaps this question can also be seen as an invitation to elaborate on how you build on the concept of "cultural techniques" (Siegert, 2013)?

BR: In a certain sense, Simondon's idea of functional meaning that I just laid out is kind of a "the medium is the message"-style call to pay attention to the material properties of contemporary technology, even if social effects are certainly not its main concern – also because technology is seen as a constitutive part of the human adventure, a type of expression rather than some kind of outside force that operates through "effects". And crucial to that idea is that technology is a constructive and cumulate affair rather than a singular logic or *techne*. It is not a tree that splits into branches, but a forest that has many different trees, some thin, some thick, some small and bushy, others growing on top of other trees. When we look at the sprawling landscape of algorithmic techniques – and the book only discusses a small selection from the subfield of information ordering – we can appreciate that all of this is made possible by the very fact that mechanical computation exists, while recognizing that there is no way to bind everything that has been happening in computing over the last 70+ years to that basic fact. Proust's *À la recherche du temps perdu* cannot be explained by the existence of the French language. And I think that the realization that looking at computational foundations is useful but ultimately far too little to understand what has been built out of it has always been a central part of Manovich's work. For the technical practitioner – and he is one – it is almost impossible not to marvel at the vast reservoirs of *stuff* that have been built by millions of people on top of computation. His earlier work is certainly still a more classic inventory of interface forms and key principles and, in that sense, less clearly marked by a resistance to

foundationalism than his more recent writings. But there has always been an attention to variation, I believe, and a desire to think about what makes the forms and functions we see on the screen possible. *Software Takes Command* (2013) indeed looks at a more demarcated space – what Manovich calls "cultural software" and no longer just "new media" – and the inventory of forms and principles necessarily becomes more detailed and specific. Simondon's long passages on the particularities of diodes and triodes come to mind, but indeed also the German cultural techniques tradition, which has always been interested in the many procedures and technicities that run through everyday life. What connects all of these efforts, my own included, is what Siegert calls a preference for "empirical historical objects" over "philosophical idealization" (Siegert, 2013, p.10). Again with an eye on Simondon, one could say that this locates the "substance" of both culture and technology not in some underlying principles, but in the massive webs of relationality that have been spun on top of whatever base layer of being there may be. Anti-platonism is one way to label this, but within media theory it basically means that we exchange McLuhan's classic inventory of media "verticals" for a transversal and increasingly integrated media landscape where all kinds of techniques produce all kinds of visible and invisible forms and functions, and software developers build new things out of old stuff, new stuff gets invented, and so forth.

MS & AH: We often conflate today's "engines of order" with a kind of neo-Enlightenment: the history of digital culture is littered with objects and ideas that carry more than a whiff of the "Universal Language" that Borges (2000) famously lampooned and Florian Cramer (2007) identified as a tendency in certain semantic web search initiatives. In addition to Ted Nelson's Xanadu, we could think of Wikipedia or even Julian Assange's notion of "scientific journalism" as interesting real and imagined examples of how digital culture extends the political liberalism and positivist epistemology associated with the Enlightenment. However your book – in addition to being much more careful in its historiography than this short interview format allows – shows us that this conflation obscures a very different epistemology operating within the research labs of today's search and social media companies. Could you talk about your specific interest in "information ordering" as a set of algorithmic techniques that developed outside of public libraries and how this relates to the rise of a different set of epistemological commitments, one that underlies much of the infrastructure for the data-intensive, algorithmic media we use today?

BR: Yes, this is spot on and a really important problem, I think. One of the points I am trying to make in the book is indeed that there are all kinds of epistemological stances and commitments within the information ordering space and computing in general. Conceptually, I am drawing quite a bit on well-known work, here: Hacking (1990) talks about different "styles of reasoning" in science, Desrosières (2001) lists four possible "attitudes" toward the "reality" of statistics, and Hjørland (2011) argues that we can distinguish rationalist, empiricist, hermeneutical, or critical "theories of knowledge" in information science, even if these positions are not often clearly fleshed out. These authors do not necessarily talk about the same things, but they are all sensitive to the variety of epistemological commitments we can find in technical or

scientific disciplines. In the book, I dedicate much space to early information retrieval and its pretty fundamental opposition to the library tradition. Interestingly, this story does not pit "literary librarians" against "positivist scientists", but rather the universalist positivism of the public library, very much tied to the Enlightenment tradition, against what you could call an empiricist perspectivism that eschews universalism and installs efficient decision-making in its stead. Practically, this means that early information retrieval techniques based on simple boolean keyword matching, such as coordinate indexing, are not evaluated in terms of whether they correspond to some universalist ideal, but rather through empirical tests and competitions that measures some kind of "retrieval performance" against other systems and human experts. If you look at sites like Kaggle (bought by Google in 2017), you can easily see that this basic evaluative setup – as well as measures like recall and precision – continue to structure the wider information ordering field. If one wants to describe this as "rationalist" or "positivist", why not, but it is not a Chomsky-style rationalism based on some logic-driven universalist ontology. People like Stephen Wolfram certainly fit that description, but I would argue that fields like information retrieval and machine learning have been dominated by output- and performance-driven thinking that has (maybe) surprisingly little interest in ontological concerns.

Whether Google's machine translation system "understands" human language is simply much less relevant than the various measures of user "satisfaction" the company can derive from its interface and other forms of practical evaluation. I think that we really underappreciate how important that difference is. I still often read that informing all of these efforts is some kind of Laplacian determinism that imagines a computational universe where everything can be known and predicted given the right technique. I would guess that very few people in the field actually think that way, but in the end, it is really not very relevant. What is much more important is that the prevalence of computational infrastructures – think online platforms – make it easy to submit every algorithmic decision, from ranking to recommendation to price modulation, to some kind of empirical test. Do users click on the ad I put on the screen? Do they buy the product? Do they pay back the loan? Again, there are different epistemological strands running through computing and stuff like the Semantic Web could indeed be seen as a continuation of the library project; but what Desrosières calls "accounting realism" (2001), where truth is simply a function of performance, seems to be what is really dominating at the moment. It is all about practical outcomes, "solving problems", and so forth, pretty much in line with the morally and intellectually unambitious utilitarianism we have come to know so well.

MS & AH: For this special issue we have taken the concept of "legacy systems" outside of its industry and computer science contexts to try and think through the intersection of key themes in Internet histories research – continuity and discontinuity, the materiality of ostensibly immaterial digital production, the productive and creative practices enabled by "old" systems and the way these systems may also constrain future action, the question of how disappeared technologies continue to operate into the future at a cultural, economic, and social level, and so on. We would like to invite you to play with this concept – in particular we wonder how you might theorize or redefine legacy systems (from its original technical sense) within the framework you

offer in your book. How does the concept relate to the kinds of accumulation and concretization of software technologies that you discuss? It seems that the hybrid systems you describe at times (e.g., that combine traditional library cataloging and information retrieval) could also offer a perspective. And much more speculatively, could you imagine a situation in which the information ordering techniques that you analyze themselves become legacy systems? What does history tell us about the possibility for some new schema that makes us reassess or doubt the value of the various algorithmic techniques at the center of your book?

BR: This question can be approached from different directions, but one way would be to start with the notion of "abstraction". In a software context, it basically means building function on top of function, like when higher level programming languages package a series of underlying operations into a single command. For example, most programming languages already come with the capacity to retrieve data from a remote server quickly and easily, "hiding" a lot of complicated network stuff from the programmer. One can lament, à la Kittler (1997), the loss of direct access to the underlying "hard stuff", but the fact is that modern development environments make it possible to do a lot of things very quickly, because of the layers of abstraction – or accumulated functionality – they sit on top of. Almost every piece of software is already a hybrid in that sense, tying together various elements that each have their own temporal trajectories.

Software-making is thus constantly faced with the "legacies" of previous work: since we necessarily build on top of other things, our space of expression is both *widened* and *structured* by what came before. Widened, because we can rely on other people's work, which is not just convenient, but allows us to integrate function that we would not be able to program ourselves – think about complex "mathy" stuff like machine learning, but also heavy "plumbing" like operating systems, network stacks, or database management systems. Structured, because all of that stuff we build upon comes with its own capabilities, idiosyncrasies, and ways of doing. I spend so much time on the book on these questions, because the information ordering techniques I discuss in the second part *have already become* hugely important legacies in that sense. The relational database model, for example, is still the standard way of thinking about data organization and access not just because it's a pretty powerful idea, but because the systems that implement it – from Oracle to Postgres – are everywhere. Every webhoster includes some MySQL storage in even the most basic bundle, every WordPress installation runs on top of it, every web developer knows how to use it, and so forth. Sure, there is new stuff around the margins, e.g., NoSQL databases that trade query power and consistency for speed, but so much of the technology we build upon today is really old, despite the constant improvements and enhancements Simondon calls "concretization" (Simondon 2017). Every new version of a programming language brings some new features, some performance enhancements, and maybe some stuff breaks somewhere down the line. And because some of that stuff is so ingrained, it can be hard to steer away – the concept of path dependence, which holds that past choices continue to structure future possibilities, applies particularly well to computing. This does not mean that things that were broadly adopted at one point never disappear, but most often it is some kind of rotting away as the version numbers climb.

HTML and JavaScript, for example, kind of trod along, newer pages starting to break on older browsers and vice versa. Flash would still be with us if Apple had not decided to kill it. But only megacorps like Apple, Google, or Microsoft have the power to intervene that hard in what is generally a rather steady stream, despite the constant buzzing at the margins. But make no mistake, these companies are certainly trying to pull that stream into their direction.

But there is really no regularity to these processes and this makes historical work so important: continuity and discontinuity emerge in complicated ways that sometimes have technological reasons, sometimes economic ones, and sometimes they come out of the haphazard work of some standards committee or some freak coincidence. But once established, technologies can echo through the decades and in computing almost everything is at least in part a legacy system.

MS & AH: In your book you take a software-building view to write a history of important software-making practices that have defined contemporary Internet-enabled services. If indeed "software is eating the world" (Andreessen, 2011) you ask the important question of how this software has been developed and which fundamental ideas and techniques have been built into it. You argue that a key starting point to explore this is to understand "software as historically accumulated archive of technical possibilities and of software-making as technical creation" (Rieder, 2020: p. 37). Software is seen as an assemblage of techniques and of old and new building blocks and ideas. A central tool in current web and app development is the employment of so-called software development kits (SDKs) which consist of reusable development tools libraries and as such provide predefined ways to develop applications. If most of such tools for building mobile apps or apps for social media platforms are offered by a single company (e.g., Facebook, Twitter, Google, Apple), then are we experiencing the platformization of app production? Additionally, as these companies retain control over when and how apps should be updated, are we witnessing an update culture where legacy systems and techniques have become impossible?

BR: In one way, this is not really new. Big companies – IBM and Microsoft in particular – have always exerted some level of control over what runs on their systems and stuff regularly stopped working for one reason or another. But the last ten years have certainly exacerbated the situation considerably and in new ways. As I said, making software always means relying on other people's work and that is not necessarily a bad thing. But the companies you mention now indeed provide building blocks that bind programs, integrating them into close and continuous relationships that can be severed at any time. These relationships certainly include technical provisions like SDKs, but the main element is platform access and how it is controlled. In the case of Facebook (excluding Oculus) and Twitter, this mostly concerns the possibility to tap into their social graphs in one way or another and the point of control are the web APIs and the legal frameworks that surround them. If you built a social game on Facebook's infrastructure some years ago and the company decided that it wants to move into another direction, that is it. That stuff is gone. In the case of Android and iOS (and Oculus), it is even more complicated since we are talking about full-fledged operating systems that are tied to both hardware and the app store model in

complicated ways. The first problem is that it is pretty hard to just install a different operating system on these devices. You have alternative Android distributions, sure, but real top-to-bottom alternatives like Sailfish OS are hard to install and pretty limited. Huawei's HarmonyOS is a wildcard, but China always is.

Going back to the previous questions, we can say that in the context of smartphones, what developers can technically build on is provided and controlled by two companies. One is locked down further than the other, but building an alternative SDK for Android – which would be possible to an extent – is not trivial at all. And then we have the distribution model, the app store. For Android, you again have some more choice, F-Droid for example, but in both cases, there is a dominant way to get software on the device and this is not just coercion. These stores hide distribution and payment behind an abstraction layer and provide access to huge markets. So what these companies offer – and both MacOS and Windows are moving in the same direction – is a combination of carrot and stick that is hard to pass on. The consequence is what you point out: a fundamentally asymmetric relationship, where software-makers have to constantly adapt to technical changes, but also to changes in legal arrangements, in "norms" concerning e.g., sexuality or special provisions for kids, in marketing practices and visibility on the app store, and so forth. In that situation, a lot of stuff is just going to stop working for one reason or another.

Maybe what is needed is a new form of "unbundling" – that term refers to the 1960s, when IBM, under investigation from regulators, started to charge separately for hardware and software, creating a space for an emerging software industry. But given the size of these companies and the current climate of eroding international cooperation, it is hard to imagine how this could be pushed through. At the same time, there seems to be a growing willingness to consider these issues in thinking about antitrust measures and this is a reason to be at least a little optimistic.

From a preservation perspective concerned with "future histories", there is no one-size-fits-all solution to these problems. In some cases, keeping the old hardware around will work and emulation has certainly come a far way – this could even keep iOS software running when it is no longer available on the app store. But what about software that has a social component or runs on some proprietary network architecture? Walkthrough videos, screenshots, oral history projects, or data dumps like Twitter's (shaky) agreement with the Library of Congress may be the best to hope for here. Research papers dealing with the present will become sources for future generations. Historians have always dealt with imperfect archives and the ravages of time and, at the risk of sounding callous, what a society cares to preserve and what not is a first finding.

MS & AH: At the beginning and end of the book you discuss Simondon's argument that as societies we must prioritize "technical culture" more than we currently do, and you also call for a "widening of technical imagination." It seems these are points on which a broad set of histories of computing and digital culture can really build on your work.

BR: Yes, the related concepts of technical culture and technical imagination have been really inspirational. Just to be clear, this is not the "digital skills" type of thinking and certainly not limited to the push for programming education we are currently seeing in many different areas. These efforts are not detrimental either, but what

Simondon imagines when he talks about technical culture is a much broader consideration of technology as a central part of human life. This is very much a debate about the general knowledge – *culture générale* in French – a technological society needs to govern itself democratically. A broad understanding of technical principles and how they connect to the spaces and things around them is more important here than the "how-to" component. If you take something like a smartphone, there are many different elements to take into account, including the hardware and the different sensors that define the basic computational capacities, the layering of software, the networking capabilities and how they enable the myriad data flows that go in and out of the device at any second. But also things like the app store logic, the role of advertisement, and the data collection that informs business models. And not to forget the many life-cycle aspects that range from rare earth mining to energy use, waste production, and the planned obsolescence that remains prevalent in the industry. A sensitivity for potential social and environmental effects are part of this as well. Technical imagination is then the capacity to think creatively in these spaces, to come up with new ideas, but also to think about alternatives, limitations, and different ways of looking at things. Free software (and hardware) and civic technology are examples for areas where these sensitivities are given room, but ecological movements such as zero waste, circular economy, off-grid living, or even tiny houses also cultivate relationships with technology that are very attentive to the many connections between technical principles, modes of living, and systematic effects. I am not saying that these movements are going to solve the ecological catastrophes we are facing, but they show pathways towards fighting the alienation that comes from living surrounded by technical objects and systems that we neither control nor understand.

One of the things I emphasize in the book is technological pluralism and that not only means highlighting the differences in epistemological stances or commitments observable in computing, but also to plead for broader forms of education or, more generally, a clearer recognition that different approaches to computing are both possible and desirable. And this is where historiography comes in. Looking at the complex and often weird histories that have been unfolding around computing and other "digital" stuff not only sharpens our sense for contingency, but is also a pretty interesting way to learn about the substance and relational embedding of technologies. Writing a history of HTML, HTTP cookies, or Flash, for example, requires quite a bit of technical understanding and readers of such works inevitably learn about the technical underpinnings of the web, about the different forces that shaped them, and about the spaces of variation that opened and closed. Historiographical work that is attentive to technical principles is, in my view, both an example and a vehicle for technical culture.

Notes

1. More about Bernhard Rieder's publications and work can be found at: http://thepoliticsofsystems.net/ and https://www.uva.nl/profiel/r/i/b.rieder/b.rieder.html.
2. The interview was conducted between December 2019 and January 2020.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work is part of the research programme Innovational Research Incentives Scheme Veni with project numbers 275-45-006 and 275-45-009, which are (partly) financed by the Dutch Research Council (NWO).

References

Andreessen, M. (2011, August 20). Why software is eating the world. *The Wall Street Journal*. http://online.wsj.com/news/articles/SB10001424053111903480904576512250915629460.

Borges, J. L. (2000). John Wilkins' analytical language. In E. Weinberger (Ed.), E. Allen & S. J. Levine (Trans.), Selected non-fictions, (pp. 229–231). New York, NY: Penguin.

Cramer, F. (2007, December 18). Critique of the "semantic web". Retrieved from http://www.nettime.org/Lists-Archives/nettime-l-0712/msg00043.html

Desrosières, A. (2001). How real are statistics? Four possible attitudes. *Social Research*, *68*(2), 339–355.

Hacking, I. (1990). *The taming of chance*. Cambridge, UK: Cambridge University Press. doi:10.1086/ahr/97.1.157

Hjørland, B. (2011). The importance of theories of knowledge: Indexing and information retrieval as an example. *Journal of the American Society for Information Science and Technology*, *62*(1), 72–77. doi:10.1002/asi.21451

Kittler, F. A. (1997). There is no software. In J. Johnston (Ed.), *Literature, media, information systems: essays* (pp. 147–155). Amsterdam: Overseas Publishers Association.

Manovich, L. (2001). *The language of new media*. Cambridge, MA: MIT Press.

Manovich, L. (2013). *Software takes command*. New York, NY: Bloomsbury Academic.

Rieder, B. (2017). Scrutinizing an algorithmic technique: The Bayes classifier as interested reading of reality. *Information, Communication & Society*, *20*(1), 100–117. doi:10.1080/1369118X.2016.1181195

Rieder, B. (2020). *Engines of order: A mechanology of algorithmic techniques*. Amsterdam: University of Amsterdam Press.

Siegert, B. (2013). Cultural techniques: Or the end of the intellectual postwar era in German media theory. *Theory, Culture & Society*, *30*(6), 48–65. doi:10.1177/0263276413488963

Simondon, G. (2017). *On the mode of existence of technical objects* (C. Malaspina and J. Rogove, Trans). Minneapolis: Univocal Publishing.