Kathrin Passig

The black box is a state of mind

Published 2 February 2018

Original in **German**
Translation by **Saul Lipetz**

First published in **Merkur 823 (December 2017) (in German); Eurozine (English version)**

Downloaded from eurozine.com (https://www.eurozine.com/black-box-state-mind/)

Rapid advances in machine learning have prompted much debate about the sinister implications of 'black-box' algorithms. Yet fears about the opacity of computer code are as old as software itself, writes Kathrin Passig. Indeed, black boxes are all around us, not just inside our computers…

In 2012, I wrote an article [1] about algorithms, then a hot topic, in which I mentioned a *New York Times* article published four years earlier about the Netflix Prize. The algorithms for recommending films were getting better and better, explained the *Times*, but also harder to understand – not only for users, but also for developers: ‘Chris Volinsky admits that his team’s program has become a black box, its internal logic unknowable.’ I quoted this remark in my article.

More and more articles then started appearing describing ‘algorithms’ as ‘black boxes’. They annoyed me, especially because I too had been guilty of mystifying algorithms. It starts with the terminology. We could simply say ‘software’, but that brings to mind dusty floppy disks rather than anything more sinister. In fact, an algorithm doesn’t necessarily involve a computer. It once simply denoted a clearly defined set of instructions, for instance a sorting method or a guide for assembling an IKEA bookshelf. The critique of algorithms, however, is not about small building blocks but large systems. What can you do? Colloquially, ‘software’ has been replaced with ‘algorithm’ and protesting against shifts in language is pointless.

The critique of algorithms at the beginning of the decade was different than it is today. After the eruption of Eyjafjallajoküll, *Frankfurter Allgemeine*

Zeitung editor Frank Schirrmacher argued that it was wrong to shut down European air traffic on the basis of simulations. Journalist Miriam Meckel criticized the recommendations on Amazon, Facebook and iTunes for failing to account for coincidence and unpredictability. Media theorist Geert Lovink protested against the 'ruthless algorithms' of 'Finazism'.

Since then, there have been massive advances in 'machine learning' or 'deep learning', advances that have now become visible in everyday life. In 2008 I wrote a piece of code, as part of a Twitter bot, that would cause the bot to follow its own followers while ignoring spammers. The code consisted of questions like: Do more than 17 of the last 20 tweets contain a link? Are there no replies in the last 20 tweets, i.e. is the person just tweeting to herself? Does the account description contain words like 'free', 'bargain' or 'credit'? Each 'yes' would add a point for a likely spammer, while for each 'no' a point would be deducted. Anyone with three or more points would be treated as a spammer. This was the traditional method.

With machine learning, you also start off by considering a range of features that might be relevant. But instead of pre-determining their relative significance, as in my code, you use trial and error. If, when operating a particular random setting, the controller can't tell between a known spam sample and a sample known *not* to be spam, then the code is adjusted and re-tested. If the system works, then it can then be tried out on unknown material. (Caution: gross over-simplification – do not programme any missile defence systems on this basis.)

The critique of algorithms has begun to focus on this area. 'Deep learning is creating computer systems we don't fully understand' ran the headline of an article in the technology magazine *The Verge* in July 2016. Apparently, until now we had been able to fully understand the computer systems we were using. Elsewhere it is stated even more explicitly that transparency has only now ceased to exist: in 2016, for example, the *Scientific American* wrote that 'we no longer fully understand *why* algorithms are deciding things the way they do'. Media theorist Felix Stalder has written that these self-learning algorithms are 'in many cases so complex that they can't even be understood in retrospect. They can be tested experimentally, but no longer logically. Such algorithms are essentially black boxes – objects that can only be understood through their outward behaviour, but whose inner structure defies cognition.' [2]

All this is right, and yet not quite. It makes it sound as though the inner structure of software was, until very recently, readily comprehensible. Yet the opacity of software systems has preoccupied journalists and developers for much longer. In 1987, *Der Spiegel* published a conversation between the computer scientists Klaus Haefner and Joseph Weizenbaum entitled ‘An explosion of nonsense’. [3] At the time of the discussion, Haefner had been in the business for eighteen years and Weizenbaum for thirty-six. Michael Haller, the editor of *Der Spiegel* who was moderating the debate, says the following: ‘Complex operations are often impenetrable even to the programmers who wrote the program. What exactly goes on in ‘inside’ them remains a mystery. We then fall back on the “black box” model and merely ask what results the machine gives when we enter this or that data.’ Haefner agrees: ‘Here we reach the crux. Unfortunately, more and more systems are being built that are so complicated that we can no longer get a proper picture of them … That human nature has its unpredictable sides is something we’ve been able to live with thus far. But a computer whose inner workings are unpredictable is dangerous and contradicts the very direction in which evolution is supposed to run.’

Where does Haller get this notion? Probably from Weizenbaum, who that year had told the magazine *LOG IN* how ‘I once visited Karl Deutsch when I happened to be in Berlin – at the Science Centre in Steinplatz. We chatted for a while in his office, and I told him about the opacity of large systems. I told him that there is no large system in existence that can be fully grasped by an individual, that it is too late to control such systems, because they are a consequence of their history and that history is no longer available.’ [4] As an example he cites Globus, a program for simulating the global economy developed at around that time in Berlin.

Globus consisted of approximately ten thousand lines of code in Fortran 77. ‘These are vast programs which I don’t think anyone could ever properly comprehend,’ Weizenbaum says. The average iPhone app was about five times that length in 2013. The Hubble space telescope has two million lines of code, the Firefox browser has about ten million. Facebook has over sixty million and Google roughly two billion. [5] But that doesn’t mean Weizenbaum was wrong. Ten thousand lines of code can be impenetrable. So can a single line!

These ideas had already appeared in Weizenbaum’s book *Computer Power and Human Reason*, published in 1976. [6] It contains a whole chapter on ‘incomprehensible programs’. ‘These gigantic computer systems have usually been put together (one cannot always use the word ‘designed’) by teams of programmers, whose work is often spread over many years. By the time these systems come into use, most of the original programmers have left or turned their attention to other pursuits. It is precisely when such systems begin to be used that their inner workings can no longer be understood by any single person or by a small team of individuals.’

Weizenbaum quotes a text by the AI researcher Marvin Minsky, published in 1967, nicely titled: ‘Why programming is a good medium for expressing poorly understood and sloppily formulated ideas’. [7] This discussed the ‘widespread superstition’ that it is impossible to write a program ‘unless one has an extremely clear, precise formulation of what is to be done, and exactly how to do it’. Taking the example of the software able to beat humans at checkers, Minsky described how ‘once past the beginner level, programmers don’t simply write “sequences of instructions”. Instead, they write for the individuals of little societies or processes. For try as we may, we rarely can fully envision, in advance, all the details of their interactions. For that, after all, is why we need computers.’

The illusion that programming is a precise, rigid form of expression is based on a confusion of form and content. ‘It’s perfectly true that you have to be very precise in your computer (syntax) to get your program to run at all. No spelling or punctuation errors are allowed! But it’s perfectly false that this makes you have a precise idea of what your program will do.’ The more a program grows, the more the programmer will lose track of the details and no longer be able to say for certain what will happen, ‘watching the program as though it were an individual of unpredictable behaviour’. That, according to Minsky, was already the case with some large-scale programs, but the era of multi-user systems was only just beginning, and this was leading to a serious problem. Several programmers would be working on one program together, each from their own terminal, and none would understand the whole thing. ‘Now we see the real trouble with statements like “it only does what its programmer told it to do.” There isn’t any one programmer.’

Three years earlier, in 1964, Stanisław Lem’s *Summa Technologiae* had appeared. [8] It contains a chapter on ‘The Black Box’ that opens with the statement that no one person understands the construction of all the machinery of modern industrial society. Only society as a whole possesses this knowledge. ‘Cybernetics furthers this process, moving it to a higher

level – since it is theoretically capable of producing things the structure of which will not be understood by anyone. A cybernetic device thus becomes a “black box” (a term frequently used by experts).’ The black boxes built until then had been so simple, Lem argued, that ‘engineer-cyberneticists’ could still understand the link between the input states and output states. ‘Yet a situation may arise when even he will not know a mathematical representation of this function’. Complex systems such as ‘the yet nonexistent “very large black boxes”’ are impossible to understand ‘as systems of this kind do not have algorithms.’ The algorithm, Lem reasons, must be replicable and allow for the prediction of future conditions, ‘while a society that finds itself in the same situation on
two different occasions does not have to behave in the same way. This is precisely the case with all highly complex systems.’

Finally, we tend to end up with Norbert Wiener, the originator of cybernetics. In Wiener’s article ‘Some Moral and Technical Consequences of Automation’, published in 1960, checkers programs again serve as proof: ‘The present level of these learning machines is that they play a fair amateur game at chess but that in checkers they can show a marked superiority to the player who has programmed them after 10 to 20 playing hours of working and indoctrination.’ [9] Anyone who had played against these machines, wrote Wiener, would admit that they had original strategies. This, he believed, refuted the widespread assumption that machines could not create anything original, as well as the notion that they were essentially inferior to their creator and subject to her complete control.

Wiener was not claiming the existence of a technology that was incomprehensible *per se*. All he was saying was that it takes too long to understand it: by the time we can react to the information that has been transmitted from our sensory organs and hit the brakes, the car may already have driven into a wall. ‘It may well be that in principle we cannot make any machine the elements of whose behaviour we cannot comprehend sooner or later. This does not mean in any way that we shall be able to comprehend these elements in substantially less time than the time required for the operation of the machine, or even within any given number of years or generations.’ Our comprehension might lag far behind the completion of the task that was originally set. ‘This means that though machines are theoretically subject to human criticism, such criticism may be ineffective until long after it is relevant.’ Sixteen years later, Weizenbaum comments that what Wiener described as a possibility has since become reality.

Weizenbaum’s chapter on incomprehensible software, now over forty years old, contains almost all the formulations and arguments of the current critique of machine learning software. ‘Our society’s growing reliance on computer systems that were initially intended to “help” people make analyses and decisions, but which have long since both surpassed the understanding of their users and become indispensable to them, is a very serious development. It has two important consequences. First, decisions are made with the aid of, and sometimes entirely by, computers whose programs no one any longer knows explicitly or understands. Hence no one can know the criteria or the rules on which such decisions are based. Second, the systems of rules and criteria that are embodied in such computer systems become immune to change, because, in the absence of a detailed understanding of the inner workings of a computer system, any substantial modification of it is very likely to render the whole system inoperative and possibly unrestorable. Such computer systems can therefore only grow.’

The fact that a state of affairs has existed for a long period time without the world coming to an end does not necessarily mean that it is harmless. Perhaps the world will still end. Or perhaps we are the thoroughly indoctrinated products of this development, incapable even of recognizing the problem. In any event, it’s not enough to blame the machine learning methods of recent years and to demand a return to the simple and completely comprehensible software that we had five years ago – okay ten… or twenty… well definitely fifty years ago.

To briefly summarize the reasons for the incomprehensibility of software: for Wiener, it’s our slowness of thought. We’re probably capable of understanding what a program does, but sometimes it takes so long that any criticism comes too late. For Lem, systems beyond a certain level of complexity are fundamentally unpredictable. Minsky cites the interactions of individual processes, the proliferation of code and the effects of several programmers working together. For Weizenbaum, it is the sheer size of systems, the departure of the original programmers and the passage of time.

I would add two more reasons. The first is bugs, i.e. mistakes in the software and hardware. These were no less common fifty years ago than they are now. Perhaps we are now a bit more aware that they are unavoidable. This is a problem that can't simply be got rid of by being more careful. The second reason is that code doesn't consist merely of what one single programmer has devised. Even the simplest program relies on a whole load of third-party code that exceeds its own in size: integrated code libraries and frameworks that tackle frequently recurring tasks; layers of software that translate the code into a language that can be processed; software that displays, stores or transfers all of this.

Occasional programmers like me might understand the uppermost levels of these. Professionals will have a little more insight. There may be some people who understand the entire software of a system down to its lowest level. But even then, the hardware remains opaque. The black box isn't a chest that one only needs to open in order to know its contents. It contains more black boxes. Early programmers would mock the 'software engineers' that succeeded them for taking the hardware as a *fait accompli*. And they had probably been mocked by their predecessors for getting their chips by mail order rather than soldering their circuitry themselves. Everyone relies on the functioning of elements that for them represent black boxes.

Most of these phenomena aren't limited to software. They can be observed in buildings or technical facilities that have been rebuilt over the decades and adapted to new requirements. The documentation is seldom up to date or even in the same place as its object. The James Gregory Telescope in St Andrews is Scotland's largest telescope; I am writing parts of this article nearby. The functioning of the device, which is over fifty years old, depends to a great extent on one retired computer scientist, who in his spare time has acquainted himself with the different historical layers of the telescope's technology. Documentation of the various modifications to the device only exists – besides inside the head of Roger Stapleton – in fragmentary form and in disparate locations. This is not the exception but the rule.

I wasn't sure whether Chris Volinsky had been quoted accurately in the article I mentioned at the start. So I went on Twitter to ask. 'It's a fair quote', Volinsky replied. 'The final solution was an ensemble of 800+ models. No way to comprehend that fully.' But what do 'comprehend' and 'fully' mean here? What is the 'transparency' and 'comprehensibility' of algorithms that are constantly being called for?

There are several possible interpretations. The basic question is whether the code is viewable at all. This is the case with open source software, but generally not in the commercial sector, irrespective of whether machine learning is involved. But let’s assume that the code can be viewed. Do demands for transparency mean that it should be readily understandable even by laypersons? Probably not, since that is the case only for the very simplest pieces of code, if that.

Is it a question of whether the results can be verified? There are some computational processes where this is fairly straightforward. An introduction to scientific method from 1972 contains the following about ‘programmable electronic desktop computers’ (which really did take up half the desk): ‘It is important that the program is monitored for accuracy. To do so, the results printed out by the computer must be compared with the results obtained by doing the same calculations using pencil and paper.’ [10] As soon as the calculation becomes more complex, however, pencil and paper will cease to be of much use.

Let’s take a chess program that doesn’t use machine learning but is constructed exclusively from a decision tree containing every possible move. If we were to follow the above recommendation, we would have to print this tree out, with all its ramifications, and then go through all the different branches with a pencil. You would need a great deal of paper. And patience: the tree forms approximately 10^{120} different games. A simple machine learning solution would be easy to comprehend, by comparison. [11]

In many cases, the accuracy of the results can be verified without using a pencil: we can see today whether yesterday’s weather forecast was right or wrong. Results that are harder to check include recommendations of books, films and music. Tastes can be shaped. If my local bookseller recommends her personal favourites with sufficient enthusiasm, then I may well warm to them. If Spotify plays a particular music genre to me for long enough, I may eventually overcome my dislike. An unsuitable recommendation has become a suitable one. The forecast has changed the world. And whether the simulation-based advice to suspend all air traffic did indeed save lives, we will never find out in the absence of a parallel universe identical in every other respect.

The most important version of the comprehensibility question is: ‘Is it possible to retroactively explain or otherwise visualize the route leading to the result?’ There is considerable demand for explanations like this, and intense research is being done on them under the term ‘Explainable Artificial Intelligence’. On the one hand, it would be a clear improvement if such systems were to supply information about how they get their results. On the other hand, the explanation creates new problems. One simple example is the link ‘Why was this recommended to me?’ that appears alongside Amazon’s personalized recommendations. If you click on it, you are given the answer that you had previously ordered wellingtons, a complete edition of Goethe and a set of wax crayons and might therefore be interested in an electric nose-hair trimmer. That’s not always helpful. And it is probably a massive simplification of the inner workings of the Amazon software. The existence of an explanation reassures customers but doesn’t clarify much.

A paper published in 2016, entitled *Generating Visual Explanations*, describes software that identifies bird species in photographs and offers a textual explanation: ‘This is a Kentucky warbler because this is a yellow bird with a black cheek patch and a black crown.’ [12] On closer inspection, it turns out that this sentence does not even approximate to a description of the path taken by the software. The bird is actually identified by a first system, which remains unexplained. The name is then passed on to a second system, which selects a description that humans can understand. The description gives the impression of being a self-description, without being anything of the sort. [13] Identifying birds is one harmless example. When business interests or the social desirability of the answer come into play, the operators – consciously or unconsciously – will offer misleading explanations. The problem is now exacerbated, because even the existence of a lack of explanation is concealed.

Because the problem of the black box is an old one, people have been working on mitigation strategies for some time. In aeroplanes, ‘fly by wire’ systems have been used since the 1960s. The pilot’s actions are not relayed mechanically or hydraulically but electronically. This means that software is used which, if it fails, can have fatal consequences. For that reason, electronic systems crucial to safety are duplicated two, three and sometimes even four times. Duplication does not mean that the exact same system is copied, in case one breaks down. Different software is used, which runs partly on different hardware and is developed by separate teams. In tests, the systems have to reach identical results on the basis of identical inputs. For a specific problem – be it a mistake in the hardware, a mistake in the software, or a mistake caused by adverse environmental conditions – to lead all these systems astray in the same way is not impossible, but it is unlikely. In a system duplicated three times over, two elements that reach the same result can overrule a third element with a divergent outcome. (For the same reason, it was recommended to take at least three chronometers on board for sea travel in the nineteenth century; HMS *Beagle* had twenty-two of them on board for Darwin's voyage. [14])

Software runs on hardware. Microprocessors and circuit boards have not been designed manually since the 1970s, but by other software ('Electronic Design Automation'). This is the only way to ensure that several billion components can be accommodated on a single modern-day microprocessor. There are no human beings who attempt to understand or check with a pencil whether these components have been correctly installed – this job is done by more software systems.

In aeroplane electronics, as with hardware design, the systems are too complex to be completely understood within a reasonable period of time. The strategies for dealing with this problem do not involve calling for 'transparent' systems or demanding that their use be stopped. They include developing new auxiliary tools for visualizing events, carrying out automated testing procedures and comparing results from various systems that work differently.

There is a whole range of software testing methods that require no understanding of the inner workings of the system. They are known as 'black-box testing'. Black-box testing is when, before buying a used car, you take a quick look under the bonnet and, instead of inspecting all the parts, test drive it to check it can do the things cars are supposed to do: drive, accelerate, brake, change gear, not make any strange noises. Except, in computer science, there is no single test drive, but a great many, designed to test rare and unexpected events: for example, what happens if the car tries to drive up a wall?

Black-box testing is a technique suitable not only for situations where you have no understanding of the code. The procedure is also applied to open software, since it presents certain advantages. Especially when it comes to large, complex and imprecise tasks, black-box testing allows far more mistakes to be detected within a given time than through logical verification. [15] The robotics scientist Rodney Brooks calls the attempt at mathematical proof, in connection with artificial intelligence, a fundamentally futile endeavour: ‘With multi-year large team efforts we cannot prove that a 1,000-line program cannot be breached by external hackers, so we certainly won’t be able to prove very much at all about large AI systems.’ [16]

With machine learning programs, currently the focus of heated discussion, duplicated systems and other mitigation strategies are hardly used yet. Apart from the fact that the whole research field is pretty new, this is because of two unfortunate factors. First, in many areas, the consequences only affect individuals, who at first sight don’t appear to be victims of the same faulty system, as with a plane crash. Second, there is often no competition. If my bank behaves idiotically and offers the excuse ‘It’s the software’s fault, we can’t do anything about it’, I switch banks. If airline A keep crashing, passengers will fly with airline B. But this option doesn’t always exist. It’s not easy for residents to switch cities, prisoners can’t choose which software should determine the likelihood of their reoffending, and consumers can’t do much about their credit rating. The isolation of the cases and the lack of competition mean that software manufacturers have fewer incentives to limit errors.

The alternative is for a human to decide whether someone is released from prison early, whether they get a loan, a medical diagnosis or an insurance policy. (These kinds of decision have long been taken using software that is not especially transparent for its users *or* those affected. But let’s not go there now.) In a paper published in 2016 on ‘The ethics of algorithms’, it says that ‘Algorithmic processing contrasts with traditional decision-making, where human decision-makers can in principle articulate their rationale when queried, limited only by their desire and capacity to give an explanation, and the questioner’s capacity to understand it.’ [17]

There is one major problem with this. It lies in the word ‘capacity’. Our decisions are influenced by a vast range of factors of which we are only dimly aware. This is an old and much researched observation. When a bookseller recommends something, the logic that led to the recommendation is more unclear than any algorithm. The bookseller will offer an explanation, but the connection between this and the actual background behind the recommendation remains opaque. The human brain is a black box, both objectively and for its owner.

Stanisław Lem makes this point in *Summa Technologiae*. We know that it is possible to construct black boxes without planning them or understanding them entirely, he says, because we ourselves are black boxes. ‘Every human being is thus an excellent example of a device that can be used without knowing its algorithm. Our own brain is one of the “devices” that is “closest to us” in the whole Universe: we have it in our heads. Yet even today, we still do not know how the brain works exactly. As demonstrated by the history of psychology, the examination of its mechanics via introspection is highly fallible and leads one astray, to some most fallacious hypotheses.’ But the fact that nature has created black boxes shows that their construction is possible and that their impenetrability is not an obstacle to their functioning *per se*.

Gaps in our knowledge do not stop us thinking. They do not stop us using general anaesthetics, even though we don’t properly understand how and why they work. [18] The advantages of anaesthetics are so great that we are willing to overlook their mysteriousness. I’m not arguing that the brain, anaesthesia or software are better left unexplained, but that we shouldn’t insist on stricter standards for software than we would for other kinds of ‘black box’. It’s not unusual for something to be in use even if we cannot fully explain it. Lemons were used for centuries to prevent scurvy, long before it was understood why they had this effect.

Jessa Jones runs a repair centre for Apple devices. She explains how she works on YouTube: ‘Manufacturers’ support for repair is really non-existent. It’s an antagonistic relationship. When we go to fight for the right to repair, they come in and say, “You shouldn’t be able to repair”. There’s no support. We’re completely figuring this out on our own. My background is in molecular genetics, which is all about trying to understand how tiny things that you can’t see work when there’s no user manual.’ [19] Reverse engineering and biology are both fields where researchers investigate complex, non-transparent and non-linear systems. Particularly in biology, demands for better documented and more comprehensible organisms won’t get you very far. Even creatures that are literally transparent won’t make your work much easier. The fact that we are still able to increase our knowledge is down to a collection of strategies and methods. They are known as science and up to now have proven pretty effective.

Footnotes

1. Kathrin Passig, ‘Warum wurde mir ausgerechnet das empfohlen?’ (‘Why did they recommend *that* to me?’), *Süddeutsche Zeitung*, 10 January 2012. The title was not mine: I had suggested ‘Mummy, there are algorithms under my bed’.

2. Felix Stalder, *The Digital Condition*, Wiley 2018.

3. ‘Es ist eine Explosion des Quatsches’ (‘An explosion of nonsense’), *Der Spiegel*, 2 March 1987; online at: www.spiegel.de/spiegel/print/d-13520856.html

4. ‘On the responsibility of teachers in new information technologies: a *LOG IN* conversation with Joseph Weizenbaum’, *LOG IN*, no.4, 1987; online at: www.log-in-verlag.de/PDF-Dateien/Weizenbaum-Interview.pdf

5. Source for all figures: www.informationisbeautiful.net/visualizations/million-lines-of-code/

6. Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation*, San Francisco 1976.

7. In: Martin Krampen/Peter Seitz (eds.), *Design and Planning II. Computers in Design and Communication*, New York, Hastings House, 1967; online at: http://web.media.mit.edu/~minsky/papers/Why%20programming%20is--.html

8. Stanisław Lem, *Summa Technologiae*, Minneapolis 2013.

9. Norbert Wiener, ‘Some Moral and Technical Consequences of Automation’, from *Science*, No. 3410, 6 May 1960; online at: www.nyu.edu/projects/nissenbaum/papers/Wiener.pdf

10. Helmut Seiffert, *Einführung in das wissenschaftliche Arbeiten* (‘Introduction to today’s scientific methods’), Braunschweig: Vieweg, 1972.

11. A more serious version of this argument can be found in Zachary C. Lipton, *The Mythos of Model Interpretability*, 2017; online at:

arxiv.org/pdf/1606.03490.pdf

12. Lisa Anne Hendricks et al., *Generating Visual Explanations*, 2016; online at: arxiv.org/pdf/1603.08507.pdf

13. See: 'Machine Learning Verification and Explainable AI', in *The Foretellix Blog*, 31 August 2016; online at: blog.foretellix.com/2016/08/31/machine-learning-verification-and-explainable-ai/

14. Wikipedia: 'The Admiralty started a general issue of chronometers to H.M. Ships from 1825, but between about 1800 and 1840 the availability of chronometers could not keep up with the demand. The Admiralty therefore only issued one chronometer to each ship unless the Captain personally owned one. In those cases, the Admiralty would issue a second machine to make the total up to three, reasoning that a ship with two was no better off than with one since in the event of a discrepancy it was not possible to identify the faulty instrument.'; online at: https://en.wikipedia.org/wiki/Ship%27s_chronometer_from_HMS_Beagle (retrieved 2 February 2018).

15. See also Yoav Hollander, 'FV has much better PR than CDV', in *The Foretellix Blog*, 14 July 2015; online at: blog.foretellix.com/2015/07/14/fv-has-much-better-pr-than-cdv/.

16. Rodney Brooks, 'The Seven Deadly Sins of Predicting the Future of AI', 7 September 2017; online at: rodneybrooks.com/the-seven-deadly-sins-of-predicting-the-future-of-ai/

17. Brent Daniel Mittelstadt et al., 'The ethics of algorithms: Mapping the debate' in *Big Data & Society*, nos. 1–21, July–December 2016; online at: journals.sagepub.com/doi/pdf/10.1177/2053951716679679

18. See: *Anästhesie* ('Anaesthesia'), by Kathrin Passig/Aleks Scholz, *Lexikon des Unwissens* ('A lexicon of ignorance'), Reinbek 2007.

19. YouTube, 'The Master Microfixer Teaching the World to Fix iPhones'; https://www.youtube.com/watch?v=VNKNjy3CoZ4