schließen. Das ist dramatisch, das sind Zustände, das ist Wildwest, das können wir so nicht dulden und nicht tolerieren als demokratische Gesellschaft, deshalb ist es wirklich wichtig, dass wir hier vorangehen und sagen, wir brauchen dafür klare Regelungen.

Gemeinwohl nicht IT-L

Wir wollen neue und innovative V
Staat im Sinne des Gemeinwohls, mit neuen Bümdnispartnern, wie beispielsweise Onlineplattform-Genossenschaften, weil da

ist das dann ganz anders gelagert, da haben die Beschäftigten selbst die Kontrolle über die Gestaltung dieser Plattformen und die Regularien. Der Staat sollte für soziale Standards Sorge tragen und sich auch um die künftige Finanzierung der Daseinsvor-
chte ich kurz an das gestrige Re-
das ist sehr wichtig, dass wir uns
sofern das Modell Tax-by-Design
önlichkeitsrechte sind elementare
okratie, die spezifische Machtasymmetrie zwischen Arbeitgebern und Arbeitnehmern erfordern ein eigenes Beschäftigtendatenschutzgesetz.

FIfF-Konferenz 2018

Protective Optimization Technologies (POT): The revolution will not be optimized?

Verschriftlichung des Vortrags von Seda Gürses[1]

In the 1990s, software engineering shifted from packaged software and PCs to services and clouds, enabling distributed architectures that incorporate real-time feedback from users. In the process, digital systems became layers of technologies metricized under the authority of objective functions.

These functions drive selection of software features, service integration, cloud usage, user interaction and growth, customer service, and environmental capture, among others. Whereas information systems focused on storage, processing and transport of information, and organizing knowledge – with associated risks of surveillance – contemporary systems leverage the knowledge they gather to not only understand the world, but also to optimize it, seeking maximum extraction of economic value through the capture and manipulation of people's activities and environments. The ability of these optimization systems to treat the world not as a static place to be known, but as one to sense and co-create, poses social risks and harms such as social sorting, mass manipulation, asymmetrical concentration of resources, majority dominance, and minority erasure. In the vocabulary of optimization, these harms arise due to choosing inadequate objective functions. During the talk, I will provide an account of what we mean by optimization systems, detail their externalities and make a proposition for Protective Optimization Technologies.

Prolog: Funding matters

Zunächst eine kleine Ankündigung: Wir haben kürzlich eine Kritik an der Auswahl der privaten Sponsoren für Privacy-Konferenzen formuliert, die mittlerweile in vielen Ländern sehr üblich geworden ist. Wir haben mit einem eindeutigen Beispiel angefangen: Palantir ist bekannt dafür, Überwachungssysteme für Polizei- und Geheimdienstbehörden zu entwickeln, und ist einer der Hauptsponsoren der Amsterdam Privacy Conference. Wir haben die Organisatoren gebeten, das zu ändern. Bisher haben wir keine befriedigende Antwort erhalten. Wir wollen die Kritik ausweiten, ausgehend von Palantir, wo das Problem offensichtlich ist. Ebenso gibt es berechtigte Einwände gegen Google, Facebook, Amazon und zahlreiche andere Sponsoren im Kontext von Privacy-Konferenzen. Wir müssen eine breite Diskussion darüber führen. Mittlerweile haben wir über 200 Unterschriften. Sie sind eingeladen, das heute auch zu unterzeichnen oder einfach die Diskussion auch in Deutschland fortzuführen, wo es Konferenzen gibt, die von problematischen Zusammenstellungen von Sponsoren getragen werden.

Zusammenfassung

Es geht wesentlichen um drei Punkte: (1) Ich werde differenzieren, was Künstliche Intelligenz in der Wissenschaft, auf dem

Markt und in der Technologiepolitik ist. Für die letzten beiden möchte ich betonen, dass hier in der Vergangenheit schon viele Investitionen in beliebige Technologien gemacht worden sind. Es gibt also ein großes Angebot an Investitionen und es es wird versucht, durch die Technologiepolitik einen Bedarf zu schaffen. Es ist sozusagen eine umgekehrte Ökonomie. (2) Wir hören seit 60 Jahren, dass Künstliche Intelligenz kommt und die Welt ändert, und ich habe das Gefühl, dass das, was als Künstliche Intelligenz bezeichnet wird, sich komplett geändert hat. Es wird immer gesagt, Künstliche Intelligenz kommt, aber es ist immer etwas anderes, was kommt. Das führt zu einer Art von moralischer Panik, so dass Städte und Institutionen und Staaten immer mehr Geld investieren, damit diese Investitionen überhaupt erfolgreich werden. Zudem wird in den Künstliche-Intelligenz-Diskussionen, nicht in der Wissenschaft, aber in der Technologiepolitik, immer von abstrakten Technologien gesprochen. (3) Ich werde heute die Perspektive auf diese Technologien mit einem Blick auf den Wandel der Praxis der Softwareentwicklung beleuchten. Ich werde nicht diskutieren was Künstliche Intelligenz und diese abstrakten Technologien sind, aber ich werde fragen: Wie sieht Softwareentwicklung aus, wie hat sie sich verändert und wie hängt das damit zusammen, was jetzt Künstliche Intelligenz genannt wird? Ich werde zeigen, dass wir mittlerweile sehr datenintensive Services haben: Software wird nicht mehr als Produkt, sondern als Service angeboten werden, Services, die dann innerhalb der Logik von Optimierung funktionieren.

Es geht hier um die Optimierung aller Software-Umgebungen und Populationen. Wir müssen ausgiebig darüber diskutieren. Ich werde mich hier auf die externen Kosten der Optimierungen konzentrieren. Jedes optimierte System birgt externe Kosten, Externalitäten, die durch die Optimierung entstehen. Diese sollten einerseits internalisiert werden, das heißt Technologiefirmen sollten diese Kosten tragen, andererseits sollten wir als Nutzer auch von außen Widerstand leisten können. Die Idee von POTs, Protective Optimization Technologies, baut auf die Idee von PETs auf, Privacy Enhancing Technologies. Analog zur Idee, Werkzeuge zu entwickeln, die Nutzer befähigt, ihre Privacy sozusagen selbst schützen zu können, sollen Nutzer sich auch vor den durch Optimierungen verursachten Externalitäten schützen können. Wir wissen, dass solche Schutzmechanismen nicht revolutionär sind, und die strukturellen Probleme von Daten und Optimierungen nicht völlig lösen, aber sie sind ein hilfreicher und wichtiger Zwischenschritt auf dem Weg zu einer Lösung.

Introduction: Artificial Intelligence – waiting for Godot

Waiting for Artificial Intelligence is like waiting for Godot. It is about creating a demand for an investment that is in the process of creating an immense supply. Artificial Intelligence has been coming for 60 years. However, what we referred to Artificial Intelligence 60 years ago is different from its manifestation today. And probably this will hold for whatever is claimed to be the Artificial Intelligence in the future. And we have not only been waiting for Artificial Intelligence. Consortia of tech companies first asked us in the last 20 years to expect things like the Internet of Things. That was in the beginning of the 2000s, for those who are old enough, and it was following the trends in the 90s in research labs like Xerox Parc where people said we will not get the Internet of Things, but we will get Ubiquitous Computing. What we instead got, instead of Ubiquitious Computing, was Social Networks.

First, governments, companies, and journalists alike argued that Social Networks would lead to revolutions and bring to us democracy – and also bring democracy even in the authoritarian world. In the last three years now we hear the Social Networks bring authoritarians to democracies. So there is an underlying logic to the argument that there is a technology that is unstoppable, inevitable, revolutionary and we should prepare ourself for receiving it.

Demand for technological solutions is constructed

First of all, what is expressed as unstoppable technological change is in fact a story about companies already investing in a technology, meaning they are already creating a supply. The objective of waiting for the technology is to create an expectation that its arrival is inevitable, and its reception is welcome – a necessary condition for economic and social progress and therewith creating a demand. In the current case of Artificial Intelligence the argument is that if your health system, your city, your car, your policing are not "smart", i.e. not made "smart" with Artificial Intelligence, you are going to fall behind. Hence cities, health and educational institutions demand it and do so without much regulation.

So the moral panic that is then surrounding this promised technology, Artificial Intelligence, is part of creating a demand. We are told that Artificial Intelligence will deliver an industrial revolution, a market revolution. However, it may also cause some problems. These include for example, according to A.I. Now, that we might lose jobs, experience bias and exclusion, have an infringement upon rights and liberties and may have problems with safeties in critical infrastructures – a short, but very worrying list.

However, this is a list that again creates moral panic. And remarkably this moral panic is cultivated with industry funded think tanks, which have apparent ties with universities who make them look very neutral. Both start from the assumption that these technologies will come, but only with limited side effects. These research centers and think tanks ask us not only to expect these technologies, but to imagine how we will live with them once they arrive, how they should be tweaked to better fit our lifes. But it's only about tweaking, and never about questioning whether we should get these technologies or not.

One of the recent manifestations of this approach is an effort to achieve algorithmic fairness which would take too much time right now to elaborate in detail. But we clearly see in the algorithmic fairness that it is merely about tweaking algorithms, but not actually questioning whether the introduction of some of these systems will cause inequalities which are much grander than what can be corrected by any tools. Well, it is signifcant in all of these framings of artificial intelligence coming, that they allow those engaged in the topic not to name any actual actors who are engaged in the political economy of Artificial Intelligence.

Artificial Intelligence is an abstraction, very much like algorithms, big data and internet of things are all abstractions. Abstraction also suggests nobody is pushing for them, they just come from the sky, they fall upon us. In this notion Artificial Intelligence can be abstractly discussed and criticised as if technologies are neutral and are only colored by the shortcomings of the world around them. This is manifest in statements like "Algorithms are not biased. They do not discriminate, but the data sets represent our bias as a society and the algorithms reflect them back to us." Algorithms in that sense are presented like neutral mirrors. They just present us to ourselves. It is conceived as an accident rather than the condition enabled by the ways in which we produce knowledge and technology.

As an alternative to this abstraction and as an alternative to this idea that artificial intelligence is inevitable and it's coming, I want to look at the past. I propose that we analyze software engineering practices and systems developments which are made possible through concrete social economic production decisions. I locate some of the origins of the current discussions on Artificial Intelligence in the shifts of software engineering from shrink wrap software and Personal Computers to services and clouds which allow software developers and companies to incorporate real-time feedback from users and their environments in the business and software development processes.

So here are the three shifts that we see at least since the 1990s. If you have been in the computer business since the 1960s you will actually see a pendulum where we used to have mainframes and services, it went to PCs and coming back. What we see now since the 1990s is the starting with shrink wrap software, waterfall models, and personal computers to what I would say is services, agile programming, and the cloud.

Interlude: "Shrink wrap software" and "waterfall model"

Formerly customers went to a shop to buy software in a box. It came with a plastic wrap around it, that was the shrink wrap. And when they opened the shrink wrap they went into a contractual agreement with the software company before even installing it. Then they took the disquettes home and installed it on their Personal Computer, if all went well had the software ready and started using it. The relationship with the company was mostly at this moment where the customer opened the box. And afterwards what they did with the software was mostly their own issue, unless people used illegal copies of the software and the companies came after this illegitimit use.

The waterfall model is a form of developing software. It came about in the 1960s. If you look at the history of sofware engineering, there is always a battle of people who think software engineering is a managing issue, that it is just breaking down tasks and managing them well, and others who think that it is a matter of experts building systems bottom-up. What we have seen in change is the waterfall model which was much more managerial and top-down being questioned in the last 20 years by agile programming. Developers who propose a whole host of different methods which I am calling agile programming that would basically put the developers and the users at the center of software engineering. And finally with a shift of services we moved away from having a lot of processing done on our computers to the cloud – what will be elaborated in more detail below.

We did an exploratory study including interviews and conversations, analysis of industry white papers and legal and policy literature to understand the phenomenon we call the agile turn – the change of the environment that we live in and its great impact on the development of privacy, but also the corresponding kinds of protective technologies we can create. To give you a feeling about the excitement around shrink wrap let me refer to a video of the Windows 95 release party[2], where people are dancing and celebrating the the new release of the Windows OS. Releasing software was something that happened every two or three years. Apparently, in Microsoft literally after the release everybody would get up and change their position in the offices in this very large company. Windows 95 was like such a big party, it was a big thing to release new software versions. But what we have seen today with services is that software developers can release software updates up to 50 times an hour, a day, whatever, depending on the company. So we can not party as much, something has fundamentally changed.

From shrink wrap software to services

I will start with the move from shrink wrap to services and focus on one or two incidents that provide some insight: There is this one statement, some people claim this was a note sent by Jeff Bezos to his employees in 2001 or 2002, where he basically said: "All the teams will henceforth expose their data and functionalities through service interfaces." So he pushed all his Amazon's workforce into creating interfaces for their databases and making their services available internally, but also externally, which could also be seen as the start of the Amazon cloud that we know today.

Another example, if we just go to the extreme – shrink wrap on the one side and services on the other is between Microsoft Word and Office 365 or Google Docs. Think about the difference in the experience: In the beginning you ran installed software on your machine and you were also maintaining your machine. Customers hoped that the software in the box mapped their hardware. This was not a trivial thing and a big headache for developers. Users had a lot of control. This was seen as a problem for many people, but not for geeks. Also, users paid in advance.

With services however, what we have is that we have no installation anymore, unless maybe installing your browser or an app which is much easier in comparison to what users had to do formerly. The update and maintenance of the code remains on the developer side, because the code remains where the developer is. There is no release party where developers have to let go off their "baby". It actually sticks with them, they can continue to make changes to the code, they can always do updates. And with services what we have, also interesting for the users, is that they don't have this much control. Everything they do can now be datified. Develeopers can basically see where the users are clicking, what services they are using, or how they are using the features.

But it also allows things like user collaboration. Formerly users sent Word documents, or Open Documents if using open source alternatives, through e-mails and then there were all sorts of versioning problems. This new environment basically allows collaboration to be much more easy.

What also changes is that you pay as you use. This is where of course the term "you pay with your data" comes in as well. Formerly users didn't pay with their data when they bought Microsoft Word, they paid for the license or got an illegal copy, but now users can basically pay per use or even try software before moving it to the whole enterprise. So it's a very big shift both for the users and the developers.

So if we look at what this whole shift means is that we have now a server-client model. That means that the transaction that was only at the moment of purchasing shrink wrap software is now throughout the use lifecycle, users are constantly in transaction with the company. We have bundled services, and we have licensing and pricing models that are basically based on use, which means intensified tracking to know who is using what and how. This is especially important, because often we talk about tracking as a problem of the advertisement world but we do not talk about it also as part of the service and licensing architectures. These are important little details.

So basically there were two different phases: production and use. What happened with the services is that developers start producing the software, and do not produce everything themselves. They integrate a bunch of services that have already existed somewhere. Then the users using the service, for instance with a pay-per-use model on-going. While users are paying for the use of the software, the production is still going on. The production and the consumption phases are collapsed, there is no separate production from use. Everything they use is integrated into how this software gets produced.

If you think for example of a website to create pictures, this website might have some specific application features for uploading photos and filters. But the developers do not want to develop things like advertisement, authentication or payment themselves. So they integrate a third party service that allows payments for instance. The same with functionalities to locate pictures or use a social network for discussion, etc. Basically the service provider will integrate numbers of third services into the website. For the users it looks like they are interacting with one party, but in fact behind the surface they are interacting with a couple and hundreds of service providers, which they mostly not not know all.

What's also interesting is that this service model also applies to the developement process. The developers themselves use a number of services to develop software. Anywhere from team integration tools, to data brokers to get data about your users or analytics like Google Analytics to test your features, or for A/B-testing. You basically use a bunch of other services to even develop the service itself. So also these are hardly visible to the users. For example FullStory was a service that a website could integrate it to give the developers a complete replay of the users' interactions in a video form. So this did not not just reveal clicks, but basically showed completely how users moved around the website, which services they used, what things they typed into the boxes that were in the forms, etc. We found that in the top one million sites we already had quite a few of the websites using this service. Colleagues at Princeton even wrote out about how they could actually see people's logging credentials and private information using FullStory.

The background why developers probably are integrating FullStory is to see if the users have problems with their services and where did they struggle with certain kind of interface elements. But it actually ends up being like a privacy nightmare. And it becomes very difficult to make these things visible to the users.

The agile turn

Another main change we see is the move from waterfall to agile programming.[3] The waterfall model was basically a model of software engineering with discrete development-phases that come one after the other and finally lead to the final software programme. The company starts with a requirement analysis and specification. Software engineers go into the company to understand what the requirements of the system were or go to the environment where a system is going to be introduced. Then the requirements are turned into a specification. Afterwards they do architectural design and start implementing and integrating the software. Two or three years later the software is ready, and the engineers go back to the company who ordered the software and start doing verification and testing to see that what was produced in the last years has anything to do with what the company originally wanted. Then engineers go into the mode of operation and maintenance.

Then it turned out that with this model roughly about two thirds of the projects failed. Failed meant that they either failed completely or they failed in terms of costs, they cost much, much more than it was estimated or it took much, much longer to develop. So this model was not really working, especially from an economic perspective which is really important in my line of argument here. It also turned out that 60 % of software costs is due to maintenance and of that 60 % is adding new functionality to legacy software. So you can imagine that when service architectures came and the code could remain on the side of developers, the companies that were losing so much money and basically bleeding with these waterfall models were very happy. And so were the developers who claimed the Agile Manifesto.

Here are basically, in a very quick and unfair manner, the main principles of agile programming:

- Individuals and interactions over processes and tools: It is very interesting what gets left out here and how that actually impacts the practice of development: The process itself is gone.

- Working software over comprehensive documentation: If you think about the European General Data Protection Regulation you see the conflict with its obligations on documentation.

- Customer collaboration over contract negotiation: It is very much about reducing costs, and there goes the requirements document.

- Responding to change rather than following a plan.

Of course, there are many different approaches of Agile Programming. Another approach is called Extreme Programming which says a lot about getting things even faster and which is very focused on exhaustive testing.

What happens with the move to agile development methods is that testing becomes very central. So does user centricity or client centricity most of the times. Because we have services and we can constantly collect data about how features are being used, we can also do all sorts of testing using that data. Accordingly, it is no longer about testing the correctness of software but testing the interactions. Development is supposed to be in short iterations. So in order to develop a photo album service, developers start with the uploading and then go feature by feature what and every step they iterate and test to see how the users are reacting. Another goal is simplicity, a main goal is re-use and modularity which means to break things down as much as possible.

Optimization

What we start getting as a result of all these developments is a feature based optimiziation. What we start seeing is that if software development was a cost issue with services and with the kind of data we can collect, it comes completely out under the logic of optimization. So when Annette Mühlberg in this issue is talking about workers being put under the logic of optimization or municipalities or health systems, we see also software developers are put under the logic of optimization and all the resources they use as well. That becomes possible because we have services and because we have agile methods.

With services you can capture the behavior of the users, in order to constantly adjust the features, to know which are the features that lead to maximizing profit. We can economically optimize while we are optimizing for example for clicks. But the tracking included in services is not limited to tracking the users. For example, when integrating a service into a website, there is a need to also track those services themselves. Developers need to know that the services are all functioning the way they should be, because their products are dependent on external parties for the services to run. There is a lot of tracking going on to make sure that the service ecosystem is functioning. Finally there are literally layers and layers of tracking to see how many resources are used, which services are working, user tracking and license tracking – we are seeing a whole sort of ecosystem of tracking and optimization that is going on.

Implications and externalities – optimization in practice

Finally I want to give an example in location-based services, Waze. Many people use Waze to reroute around traffic. It is similar to Google Maps, also owned by Google, but does not just give directions, but it tells you how to get there in the fastest way possible. Waze is symbolic of how location-based services have changed with the move to services. Location services no longer just track and profile individuals to generate spatial intelligence, but they now leverage this information to manipulate users' behavior and create ideal geographies that optimize space and time to their investors' interest. So part of this development are population experiments using techniques like A/B testing that drive iterative design in order to ensure sufficient gain for most users while maximizing profits. Waze reroutes users even when there is congestion on a given path allowing these users to act with perfect selfishness. In some happy universe if everybody could get as fast as possible from A to B, you would think this is a good thing. But it turns out, researches from Berkeley showed, that traffic apps might work for the indivdual, but they do so at the cost of making congestion worse over all. Waze also often redirects users off the highways through suburban neighbourhoods that cannot sustain heavy traffic. So while it's useful for drivers, it affects neighbourhoods by making streets busy, noisy and less safe.

Consequently, towns may need to fix and police roads more often. With such systems even when some users benefit, non-users and the environments they inhabit may bear the ill effects of optimization. So this underlines the point that actually the traffic engineers that looked at it show that if only a few users are using Waze they do get from A to B faster, but if everybody starts using Waze there is just chaos.

What are we proposing? Optimization systems have these externalities and we have many, many reports that the companies do not react when people say: Look, my neighbourhood is destroyed, because Waze is rerouting this traffic. Annette Mühlberg had a lot of examples of workers getting weird messaging from their bosses, because they took pauses, etc. There are all these externalities and the companies will not take them seriously. So could we maybe do something else? And we decided to be inspired by the users of these systems for exploring new ways to deal with these things. Namely we found out that people who live in these suburban areas that do not want the Waze traffic will start reporting road blocks on their streets so that Waze would reroute to another road.

So here are some examples: In one of them somebody did a virtual road block. Apparently even the police has a problem, because people can log for example whether there is a police control and speed limit, etc. So they started over-populating Waze so that people do not have the correct information. There were some researchers in Israel that created a bunch of ghost accounts and managed to make it look like there is a lot of traffic on the freeway, got all the cars rerouted and then had the freeway to themselves. Our idea is that we want to be inspired by how these users have been looking at manipulating optimization systems, re-optimize themselves and the environments that they live in. We want to design tools that actually are effective and allow them to do so in a much more successful way. Basically, in computer science terms what we are saying is optimization systems are machine learning systems at this point. What we can do is use adverserial machine learning to re-optimize the conditions of the users and their environments when they are hit hard by optimization systems.

We published a paper on what we mean by POT and how to develop them, but basically we propose that you think of a benefit function.[4] Let's say Waze has a benefit function for all the people including non-users and the environment. The people who benefit the most are on the top-hill, they are happy users and then the ones who are messed up when cars start spying into their yards, because it is a road that shouldn't have that much traffic. They are the ones that are not getting a very big benefit. We think about what kind of inputs we can do to change this curve, the outcome of this optimization system. The question is what should it be changed to? There we have different strategies.

And really Waze is not the only place this happens. For example Pokémon Go, if you are into collecting Pokémons it turns out that if you are in a very poor neighbourhood, if you are in rural areas, you are not gonna get a lot of Pokémons. So what the users have been doing is spoofing their GPS so they go to main cities. They even changed the map indicators in Open Street Map to make it look like there are things like water sources, which apparently will lead to spawning more Pokémons. Uber drivers have been known to turn off their apps when a big event is happening so that they can induce search prices so that can be paid well. We have been looking at credit scoring outcomes and asked if you could change for example the amount of money you ask for – in order to more likely to get a credit from the bank. You can read more about our empirical work in the referenced paper.

Conclusion: The cost of optimization

What is at stake with all this? Agile software engineering seems like an efficient way to create software. But what is at stake is that all of a sudden all of our lifes are put under the logic of optimization. All of the social activities that are touched with services are put under the logic of optimization. Think about what it means to put all sorts of social things into a logic of logistics and control. Also politically, often what we get is based on the idea that the outcome is is in some way "optimal" and therefore we are supposed to desire these systems.

There is a lot happening there with this switch from information systems to optimization systems. Trying to summarize what has changed we can say: Not only we went from shrink wrap to services and software engineering has changed, but the quality of the systems has changed. Whereas information systems are focused on storage, processing, transport of information and organize knowledge and are thus about knowing, knowledge and reasoning, optimization systems are not about knowing at all. They basically leverage the data that they gather, to not only understand the world, but optimizing it.

Optimization systems seek to extract economic value through the capture and manipulation of people's activities and environments. All of these services constantly capture your activities, they capture information about our environments. They don't care to know, they care to manipulate in order for improving the economic income. Consequently optimization systems in comparison to information systems treat the world not as a static place to be known, but one to sense and co-create. This is very, very different kind of systems. The kind of problems that we can see with optimization systems have been on the headlines, they have been spoken about by academics and activists for a long time:

- Social sorting – we hear a lot about algorithmic discrimination, but originally academics started talking about this problem in the 1990s;
- mass manipulation – we hear about Cambridge Analytics and the elections;
- asymmetrical concentration of powers – we have the GAFAM: Google, Apple, Facebook, Amazon, and Microsoft;
- absolute majority dominance – where minority users get hunted down almost and there's very little done to protect the minorities.

So these are the ways we talk about these things in general, but if we look at software engineering we can talk about the phenomenons in terms of optimization.

Software engineering has become data-centric and it functions under the logic of optimization. What can we say about these things in terms of optimization systems? We argue that optimization systems come with externalities. There is no optimization system without these problems. There is no "good" optimization goal without those problems. Optimization systems always come with externalities. This is not a complete list, but here is a start: They disregard non-users for sure, see the examples given above. They disregard environments that they function in. They make use of them.

There is an optimization curve and some people are optimal and some people are not. Developers have to explore the world, they sometimes have to try new things out which might lead to exploit users to explore things. The costs and risks associated with that exploration mostly falls on the shoulders of the users.

Anmerkungen

1 *Der vorliegende Vortrag basiert auf zwei Papieren, in denen sich weiterführende Referenzen finden: Gürses S und Van Hoboken J (2017) Privacy after the agile turn. In The Cambridge Handbook of Consumer Privacy, 1–29. Cambridge Univ. Press und Overdorf R, Kulynych B, Balsa E, Troncoso C und Gürses S. POTs (2018) Protective Optimization Technologies. arXiv:1806.02711 [cs], 7. Juni 2018. http://arxiv.org/abs/1806.02711.*

2 *https://www.youtube.com/watch?v=84c7mRP7PLw*

3 *If you believe in one of these things I am going to do a bad job in explaining your color of agile development. I am very, very sorry. I am just giving a very broad overview.*

4 *https://arxiv.org/abs/1806.02711*

Seda Gürses

Seda Gürses ist Informatikerin, Post-Doc am COSIC/ESAT im Department of Electrical Engineering der KU Leuven, Belgien, und forscht am Center for Information Technology and Policy, Princeton University. Sie arbeitet u. a. zu den Themen Privatsphäre in sozialen Online-Netzwerken und der Rolle von subjektiven Vorstellungen im IT-Design.