Privacy After the Agile Turn[1]

Seda Gürses[2] and Joris van Hoboken[3]

In this chapter, Seda Gurses and Joris van Hoboken explore how recent paradigmatic transformations in the production of everyday digital systems are changing the conditions for privacy governance. Both in popular media and in scholarly work, great attention is paid to the privacy concerns that surface once digital technologies reach consumers. As a result, the strategies proposed to mitigate these concerns, be it through technical, social, regulatory or economic interventions, are concentrated at the interface of technology consumption. The authors propose to look beyond technology consumption, inviting readers to explore the ways in which consumer software is produced today. By better understanding recent shifts in software production, they argue that it is possible to get a better grasp of how and why software has come to be so data intensive and algorithmically driven, raising a plethora of privacy concerns. Specifically, they highlight three shifts: waterfall to agile development methodologies; shrink-wrap software to services; and, from software running on personal computers to functionality being carried out in cloud. They shorthand the culmination of these shifts the "agile turn". With the agile turn, the complexity, distribution and infrastructure of software has changed. What are originally intended to be techniques to improve the production of software development, e.g., modularity, agility, come to also reconfigure the way businesses in the sector are organized. In fact, the agile turn is so tectonic, it unravels the authors' original distinction: the production and consumption of software is collapsed. Services bind users into a long-term transaction with software companies, a relationship constantly monitored and improved through user analytics. Data flows, algorithms and user profiling have become the bread and butter of software production, not only because of business models based on advertisements, but because of the centrality of these to a successful disruptive software product.

---

[1] Suggested citation: Seda Gürses and Joris Van Hoboken, Privacy After the Agile Turn (September 19, 2017). Cambridge Handbook of Consumer Privacy, eds. Jules Polonetsky, Omer Tene, and Evan Selinger (Cambridge University Press, 2017). Available at: https://osf.io/preprints/socarxiv/9gy73/ or https://osf.io/ufdvb/

[2] Dr. Seda Gürses is a Flanders Research Foundation (FWO) Postdoctoral fellow at the Computer Security and Industrial Cryptography Group (COSIC) at the University of Leuven and is an affiliate at the Center for Information Technology and Policy (CITP) at Princeton University. The research for this paper was conducted as a Postdoctoral Research Fellow at the Media, Culture and Communications Department (MCC) as well as the Information Law Institute (ILI) at NYU, and a Research Fellow at CITP, Princeton University.

[3] Dr. Joris van Hoboken is a Senior Researcher at the Institute for Information Law (IViR) at the University of Amsterdam, an Affiliate at the Stanford Center for Internet & Society and an Affiliate at the Interdisciplinary Research Group on Law Science Technology & Society (LSTS) at Vrije Universiteit Brussels (VUB). The research for this paper was conducted as a Postdoctoral Research Fellow at the Information Law Institute (ILI), New York University and Visiting Scholar at the NYU Stern Center for Business & Human Rights, New York University.

Understanding these shifts has great implications for any intervention that aims to address, and mitigate, consumer privacy concerns.

1. Introduction

The objective of this chapter is to explore how recent paradigmatic transformations in the production of digital functionality have changed the conditions for privacy governance. We are motivated by the lack of scholarship that focuses on these transformations and attends to how they matter to privacy in the future.[4] The introduction of information systems in different spheres of societal activity continues to spark privacy issues. But for those that are trying to understand the issues and come up with solutions, what is our mental model of how information systems and digital functionality are produced? Is privacy research and policy sufficiently informed by the predominant modes of production? This chapter originates from the realization that this may not sufficiently be the case.

Generally, the aim of this chapter is twofold. First, we wish to stimulate privacy researchers and policy makers to pay more attention to the production of digital functionality, instead of merely looking at the results of such production for privacy. Second, our goal is to help construct a starting point for doing so. To get there, this chapter looks at a combination of three transformations in the production of digital functionality, which we jointly denote as 'the agile turn'. In short, these are the shifts from waterfall to agile development, from shrink-wrap software to services, and from the PC to the cloud. After clarifying and situating the agile turn, we study its implications for privacy governance through the lens of three high-level perspectives: modularity, temporality and capture. These perspectives allow us to foreground the directions in which the agile turn requires privacy research and policy to focus more of its attention. In the process, we also underline when and how privacy scholarship and policy implicitly rely on modes of software production that are long outdated.

Academic work on privacy has focused on several framings to grasp, reflect on and criticize the developments in technology and society. One theme has emerged under the overarching frame of *data*, paralleling the focus in data privacy regulation on personal data. Privacy concerns in this frame are expressed in terms of the effects of data flows. Accumulations of personal data and the exchange of data in markets are seen to have the potential to lead to information and power asymmetries between individuals and data hoarding organizations. Such asymmetries can have negative consequences for freedom to have a private life, informational self-determination, autonomy and on contextual social norms. In this approach, data in itself has great agency in determining the conditions of surveillance and knowledge and violations of privacy.

[4] One notable exception is the work on privacy decisions of IOS developers by Shilton and Greene 2016.

A second frame has taken to analyzing algorithms (that crunch data) as its main direction of inquiry. Algorithms are seen as (opaque) gatekeepers of access, life chances, decision making and social formation. Their implementations trouble some of the core concepts of liberal democracies like accountability, fairness, autonomy, and due process (Ziewitz 2016). The algorithmic lens brings to focus the relational and messy agency of artifacts in socio-technical systems, exploring ways to demonstrate and make accountable the potential of mathematical constructs to influence everyday activities in complex ways.

Finally, much scholarship has focused on the social turn, user agency and privacy. With the rise of Web 2.0, a more dynamic and participatory environment was created, which gave rise to social networks and related technologies. In a deliberate attempt to resist techno-centric narratives, critical work in this area focuses on how users interpret and make meaning in social platforms (Jamieson, 2016). Researchers in this frame surface the many ways in which users, far from being victims of surveillance or passive receivers of consumer design, actively shape everyday technologies.

All this scholarship is very valuable, but, falls short of asking why these particular constellations of data and algorithms and user experiences that we confront have come into being in the first place? In understanding how data and algorithms are deployed in digital infrastructures, and how these come to matter for users specifically and society generally, existing scholarship on data and algorithms makes great strides in understanding how current day technologies are *consumed*. However, the ideological markers, pools of desirable knowledge, and practices of technology *production* that bring these sets of conditions forth and not others tend to go unquestioned. With the exception of some scholars who have focused on different framings, like platforms and infrastructures (Jamieson 2016; Helmond, 2015), most privacy scholarship assumes that data flows and algorithms are inevitable building blocks of our current socio-technical systems.

But if data flows and the algorithms we experience today are not just the product of a natural progression of technological innovation, why are they so prominent, and how have they come to be as such? To do justice to such questions and their policy implications, we argue that exploring their roots in the context of *software production* should also be the subject of critical inquiry and technology policy. As we will discuss later, our claim is not that the production and consumption of software can be neatly separated -- especially not after the agile turn. This is also because technologies continue to evolve after reaching the consumer market. Yet, our claim is that the ongoing focus on their consumption only is not sufficient. Rather, we believe that inquiries into their production can help us better engage with new configurations of power (Zuboff 2015) that have implications for fundamental rights and freedoms, including privacy.

We are aware that because of the broad aims and subject matter of this chapter the reader will be confronted with a variety of omissions. We cannot cover software production in all its historical and political economic glory, but only highlight major shifts in the industry relevant to our inquiry. In addition, and to give force to our argument, we sometimes rely on idealized depictions of industry practices as indicated in online materials and in the articulations of interviewees -- software developers and product managers -- we reached out to in the research leading to this article. We do hope that questions such as the discrepancy between ideals and actual practice, with an eye on the discursive work that some of these depictions do, will also be the subject of future research.

While we are interested in studying the wider societal implications of the agile turn, this chapter is concerned with its implications for privacy governance. We understand privacy governance as the combination of technical, organizational and regulatory approaches for the governance of privacy. Privacy engineering is an emerging field of research that focuses on designing, implementing, adapting, and evaluating theories, methods, techniques, and tools to systematically capture and address privacy issues in the development of sociotechnical systems (Gürses and Del Alamo 2016). Regulatory approaches to privacy include data privacy frameworks such as the FIPPS, self-regulatory guidance and sectoral laws or general data privacy laws such as those that exist in the EU and many other countries. On the interface of regulatory and organizational approaches to privacy governance one finds what Bamberger and Mulligan have denoted as 'privacy on the ground' (Bamberger and Mulligan 2015). We further refer to and rely in particular on two normative theories of privacy in our analysis; Nissenbaum's theory of contextual integrity (Nissenbaum 2009) and Agre's theory of capture (Agre 1994).

For the sake of our analysis, we make a distinction between three parties involved in the configuration of software, services and its privacy implications in the world. These are: (1) *developers and operators*, i.e. the parties that develop software, architect services and operate the cloud infrastructure. Typically, service operators themselves use other services for development and may integrate services into their offering to their customers; (2) *Curators*, i.e. the end-user-facing entities that integrate software structured as services into their own operations (this includes so-called enterprise customers). Curators pick and choose which services to use with implications for their end-users. These curators can be IT departments, local web development teams or individual developers; (3) *End-users*, i.e. the individual users, consumers, employees, workers, students, patients, audiences, whose privacy is affected by the structuring of software as services after the agile turn. We are aware that calling these parties 'users' may hide that they are the product or provide essential labor for the service to operate (see e.g. Scholz, 2012, Fuchs, 2013). We use the term end-users to emphasize that they tend to have little agency in the design of the services discussed in the chapter. We are also aware that from a privacy perspective the different underlying 'roles' will have normative implications for the appropriate flow of personal information, considering contextual integrity (Nissenbaum 2009).

Because of the broad aims of this chapter, we have relied on a combination of methodologies. This includes over 20 in-person and telephone interviews with relevant industry experts, including software developers, devops, product managers and developers, data engineers, a/b testers, AI experts, and privacy officers. During these conversations, we inquired how the production of software and services is organized, as well as how relevant transformations have come to affect the conditions for privacy governance. In addition to the interviews, we have relied on industry white papers, legal, policy and technical documents, as well as relevant scientific literature, in particular from the fields of computer science and engineering, industrial management, software studies, regulation and law. We build on Yoo and Blanchette's volume on the regulation of the cloud and the infrastructural moment of computing (Yoo and Blanchette 2015) as well as Kaldrack and Leeker's edited volume on the dissolution of software into services (Kaldrack and Leeker 2015).

In the coming sections, we first describe the three shifts that constitute what we call the agile turn. For each of the shifts, we touch on their historical roots and sketch some of its current motions. Next, we introduce the three perspectives through which we explore the implications of the agile turn to privacy governance, namely modularity, temporality and capture. These perspectives also allow us to question some of the underlying assumptions of privacy research and policy when it comes to the production of software and digital functionality more generally.

2. The Agile Turn

Over the last decade and a half, the production of (non-critical) software has been fundamentally transformed as the result of three parallel developments. First, increasingly software producers have moved from the use of heavyweight and planned development models for information systems such as the so-called waterfall model, to lightweight and lean methods[5]. These latter models are categorized under the umbrella term 'agile' software development and involve an emphasis on user-centricity, short development cycles, continuous testing and greater simplicity of design (Douglass 2015).

Second, pervasive connectivity and advances in flexible client-server models have made possible a shift from "shrink wrapped software" products to software as services as the model for architecting and offering digital functionality. In this so-called service-oriented architecture (SOA) model, software no longer runs only on the client side, but is redesigned to run on a *thin client* that connects to a *server* which carries out most of the necessary computation. In addition, the core functional components of a

[5] A 2015 survey conducted by HP as part of their report titled "State of Performance Engineering" with 601 IT developers in 400 US companies indicated that two thirds of these companies are either using "purely agile methods" or "leaning towards agile". "Is agile the new norm?" http://techbeacon.com/survey-agile-new-norm

service (e.g. authentication, payment) can now be modularized into self-contained units and integrated on demand through automated programming interfaces (APIs), optimizing business agility.

Third, the service model came along with scaling and performance challenges that have boosted the development of large data centers offering flexible computing resources, also known as cloud computing (Blanchette 2015, Weinman 2015). As computing resources in the hands of consumers have become mobile, smaller and, hence, constrained in capacity, cloud services have gotten further cemented as the dominant way to produce and provide digital functionality and host the related processing and data storage capabilities.

As a result, "hardware, once objectivized as a physical computer, is becoming distributed across different data centers and dissolving completely into infrastructures. And software [...] is dissolving in a cascade of services that organize access to data and its process-ing" (Kaldrack and Leeker 2015). Driven by an interest in programmer and code productivity, increased volatility in responding to customer requirements as well as cost-efficiency, the agile turn has offered the possibility of programming, business and computing on demand (Neubert 2015).

2.1 From waterfall to agile development

Around 1968 software engineering came to be recognized as an engineering discipline of its own (Mahoney 2004 in Neubert 2015). This recognition came shortly before IBM decided to unbundle its hardware and software, paving the way to the commodification of software products (Neubert 2015). While a variety of management and engineering literature proposed wildly different models on how to produce software, until the 1990s, structured processes, such as the waterfall or spiral model, dominated the industry. These models rely on rigorously regimented practices, extensive documentation, and detailed planning and management (Estler et al. 2014). In waterfall models, software projects have a clear beginning during which the requirements and design are settled, and a final stage during which a version of a software is tested and released to its users.

Starting with the 1980s, and continuing with the 1990s, programmers started proposing more lightweight models that promoted greater autonomy to developer teams. One of these proposals culminated in what was titled the "Manifesto for Agile Software Development" in 2001. The supporters of this manifesto value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan (Agile Manifesto, 2001).

Like many of its contemporaries, the manifesto underlines a "no design up front" attitude and brings together a series of lightweight software engineering methodologies. Some of these methodologies focus on techniques (e.g. pair-programming), while others focus on managerial processes (e.g., stakeholder involvement, stand-up meetings), with an overall emphasis on continuous testing (starting with test codes and extending to integration testing), communication and visibility of progress (Parson 2011). Most importantly, the introduction of agile methods helps produce software in much shorter iterations (weeks vs years, or multiple times a day), in greater simplicity (always do the simplest thing that could possibly work and don't design more than what you need right now), and continuous code reviews (Fox and Patterson 2013).

2.2 From Shrink-Wrap Software to Service-Oriented Architectures

Legend goes that in 2001, Jeff Bezos sent Amazon developers a memo demanding that "henceforth, all teams will expose their data and functionality through service interfaces and will communicate with other through these interfaces" (Yegge 2011). Bezos had a vision for the architecture of Amazon's offering as composed of services, internally and externally, a vision that has contributed to the company's leadership in services and the cloud.[6]

The primary technical benefits of the shift to service oriented architectures are the extensibility, integrability, and interoperability of software components in a distributed environment (Exposito and Diop 2014). In concert with web service standards, service oriented architectures make it easier for modularized service components to connect and cooperate within an enterprise or across the industry. The goal is to allow service operators to rapidly extend existing services, integrate additional service components (or create service-composites), and allow service components to be accessed through different channels, e.g., fixed and mobile devices (Newcomer and Lomow 2005). This way of architecting the offering as a set of microservices,[7] allows companies to adhere to the mantra of "doing one thing really really well" and relying on others for everything else (Google 2014, Carlson 2014). It contributes to the ability of businesses to rapidly respond to market and environmental changes—in other words, it offers business agility.

The move to services has varying impact on different software products and their users. Companies and organizations offering (information) goods and services through digital channels (shortly, curators) can now integrate themselves into the service environment, often through the mere addition of a few basic

[6] The memo ended with a clear indication of the gravity of non-compliance: "anyone who doesn't do this will be fired."

[7] More recently, the industry has started to use the term microservices for the components in SOA, signaling a trend to further decompose services into modular parts. For a discussion see e.g. Lewis and Fowler 2014.

lines of code,[8] outsourcing basic functionality like authentication, advertisement placement, or security to a third-party provider. Consequently, what to the end-user looks like a website offered by a single provider is often in reality a mix of a seamless Frankenstein and a Matryoshka doll concealing dozens of services.

For software that used to be offered through a shrink-wrap model, the implications for the users are just as significant. First, pay as you go access to service models replaces software ownership and licensing, creating a more attractive business model for producers. In shrink-wrap software, the binary of the application used to run under the control of the user, typically on the user's device. New versions would be released intermittently. Software vendors would have to make sure that their software matches the requirements of all permutations of user hardware. Updates and maintenance would be cumbersome. Users would typically have to manage them on their own.

In contrast, with services end-user data is to be secured by the service provider and the code that provides the functionality resides mainly on the server side. This allows for iterative feature development, and more control over the costs of maintenance, monitoring, support, and third-party service licensing.[9] Instead of waiting for the release of new versions, the users can benefit from continuous updates. This is a big advantage in an industry where as much as 60% of software costs are for maintenance, and of those costs 60% are for adding new functionality to legacy software (Armando and Patterson 2013). Services that support collaborative work (such as document sharing or processing) can now also better attend to the needs of end-users, even across organizational borders. The collaborative and social implications of the agile turn, under the heading Web 2.0 (O'Reilly 2005), have been the subject of significant privacy research and policy debates, while other structural characteristics have received significantly less attention.[10]

What used to be structured and marketed as shrink-wrap software, like Microsoft Word, has transformed into a service like Office 365 or Google Docs. A music player like RealPlayer now finds its counterpart in music streaming services like Spotify that not only play your favorite music, but offer recommendations, social network features, and intelligent playlists. Thus, different types of software users are pulled into service offerings replacing the software that used to run on their own hardware.

[8] Add SDK integration possibility. See e.g. Button for an example of a *service* to facilitate such integration: https://www.usebutton.com/developers.

[9] See e.g. John Vincent's argument against running a private version of a SaaS on customer premises http://blog.lusis.org/blog/2016/05/15/so-you-wanna-go-onprem-do-ya/

[10] O'Reilly's Web 2.0 explanation discussed many of the software production transformations, including lightweight development, the shift to services. With respect to privacy, he anticipates a "Free Data movement within the next decade" because of the centrality of user data collection and power struggles over its control in the industry (O'Reilly 2005).

Surely, this was not a painless transition for those in the software industry that had "a dearly held belief that installable applications can and should be treated as packaged product, to be sold to consumers at retail like a bottle of shampoo or a box of dried pasta" (Stutz 2003). And, providing some reassurance to its adherents, shrink-wrap software will continue to exist, for instance in operating systems or in the "clients" of the client-server model such as browsers or apps, as well as in safety and security critical settings. Yet, while these may still look and feel like software products, they are increasingly likely to operate under the control of producers and bundled with update, security, or performance services.

2.3 From PCs to the Cloud

Finally, the agile turn is made possible by and continues to shape, what is called cloud computing. At the basic level, Infrastructure as a Service (IaaS) cloud computing involves the economic and physical restructuring of computing resources (processing, databases and storage) into flexible, scalable utilities, available on demand (Blanchette 2015, Weinman 2015). Cloud computing also stands for a similar restructuring at the level of software production and software usage, encompassing the rise of Platform as a Service (PaaS) and Software as a Service (SaaS) offerings. Cloud computing offerings continue to develop and diversify rapidly in the various layers of development, storage and processing. Recent developments include Data as a Service (DaaS), including the possibility for services to integrate specialized data products (Pringle et al. 2014), machine learning and AI services (Hook 2016), and the rise of container models for service deployment (Cloud Native 2016).

Historically, the emergence of cloud computing reflects a return to the mainframe, after the rise and decline of the PC, which became dominant in the 1980s and 90s. In earlier days, computing hardware was very costly and the mainframe model in combination with time-sharing provided the most cost-efficient access to computing for predominantly large, organizational users. The invention of the PC in the 1970s and its subsequent mass adoption in the following decades turned the tides. People had access to and control over their own, personal computing resources to run software (or develop it) themselves. This, in combination with the rise of internet connectivity provided the basis of what Zittrain has called "the generative internet" (Zittrain 2008). Mobility and the rise of the service model have significantly changed the user device landscape. Today, users have less and less actual control over their devices as their data and software increasingly moves to servers in the cloud.

Cloud computing entails significantly more than the mere relative shift of hardware capacity from users to server farms. It involves the development of a variety of layers to manage these computing resources to the benefit of different users. Considering our focus on the *production* of digital functionality, it's worth noting that many significant developments related to these technical management layers first became common practice inside dominant Web-native service companies, such as Amazon or Google. Around 2004 and shortly after its adoption of the service oriented architecture paradigm, Amazon

realized that its internal solutions for the production and management of virtual machines could be the basis of an external offering as well (Black 2009). To phrase it differently, Amazon's cloud offerings emerged from internally oriented engineering innovations related to the efficient production of their services in a new production paradigm. Amazon's cloud services are leading in the industry (Knorr 2016).

More recently, a similar move can be observed in the proliferation of the container model for the production and management of service components in a cloud environment (Metz 2014). This container model involves a further advancement in the use of the cloud for production of digital functionality. It involves an abstraction away from the virtual machine and a focus on making the service component the dominant building block, both for development as well as for operations. In the words of the Cloud Native Computing Foundation (CNCF), that is spearheading the container model: "Cloud native applications are container-packaged, dynamically scheduled and microservices-oriented" (Fay 2015). The foundation includes the likes of Cisco, Google, Huawei, IBM, Red Hat, Intel, Docker and the Linux Foundation. Google's contribution involves the donation of open sourced container manager 'Kubernetes',[11] an open sourced solution derived from its internal solution called Borg (Metz 2015).

The agile turn has accelerated software production while transforming business operations. Clearly, this has great implications for different aspects of privacy governance. Many of the elements of the agile turn have been addressed by privacy researchers and policymakers in some way, but an integrated perspective on the implications of the agile turn for privacy governance has so far been missing. In the next sections, we develop three perspectives that allow us to look at the privacy implications of the agile turn and to start reflecting upon the ability of existing privacy governance frameworks to address some of the related challenges.

3. Modularity

The agile turn comes with an increase in modularity in the software as a service environment. The term modularity is used to describe the degree to which a given (complex) system can be broken apart into subunits (modules), which can be coupled in various ways (Baldwin 2015). As a design or architectural principle modularity refers to the "building of a complex product or process from smaller subsystems that can be designed independently yet function together as a whole" (Baldwin and Clark 1997). The concept of modularity and its application have been the subject of research in different engineering disciplines and industrial management (Dörbecker and Böhmann 2013). It is generally used to manage complexity of systems and to allow for independent implementation and reuse of system components (Clark et al. 2005) and is an important design and policy principle for the Internet (Van Schewick 2010; Yoo 2016). Modular design involves the mantra that the independence of system components is

[11] Kubernetes is derived from κυβερνήτης and is Greek for "helmsman" or "pilot".

optimized after which they are 'loosely coupled' (Van Schewick 2010). Its origins lay in the study of complex systems more generally (Simon 1962).

The implications of modularity in software and service offerings for privacy are varied and have not been systematically studied until now.[12] We observe an incentive for the pooling of data in the industry along the lines of specialized services offering basic functionality, creating what could be called a variety of functional data brokers. Second, the unbundling of service components leads to a situation in which users, when using one service, are pulled into a whole set of service relationships. Each of those relationships has its own (dynamic) privacy implications for end-users. This dynamic also lends urgency to the question of how privacy is addressed in the various business to business (B2B) arrangements between service components, internally and externally, and as a matter of engineering as well as policy. Finally, the resulting network of relationships between different services and users raises the question of who is the proper addressee for privacy norms in such an environment. Which industry players can and should take what responsibility for addressing privacy after the agile turn?

3.1 Service Use and Production in a Modular World

Before discussing some of the main implications of modularity for privacy, it is useful to further clarify the implications of modularity from the perspectives of the service curators, as well as the developers of software as service (while focusing on the privacy implications for end-users). Organizations of various kinds are pulled into using 'software as a service' delivered by third parties when structuring their offerings to their own end-users, thereby taking the role of 'service curators'. Second, the developers of software as a service may develop specific new functionality, while integrating existing software and services of third parties.

The integration of services by curators is well illustrated by the concept of the mashup, which was pioneered by services such as HousingMaps. HousingMaps was an early Google Maps mashup, created even before there was a Google Maps API.[13] It combined Craigslist apartment and housing listings on a Google Map, giving end-users a new interface for searching apartments that addressed a clear user need. The range of basic service components that is available for integration into the offering of companies and organizations[14] has matured significantly over the last decade. Many of these services have direct privacy implications for end-users. Typical service components for publishers, retailers and

[12] Pearson discusses the relevance of certain characteristics of modularity for privacy governance in her discussion of privacy and cloud computing but doesn't explicitly refer to the concept (Pearson 2008). Anthonysamy and Rashid state the emergence of novel challenges for privacy governance because of massive collection, processing and dissemination of information in hyper-connected settings (Anthonysamy and Rashid 2015).

[13] See http://www.housingmaps.com/.

[14] Here, we consider all companies and organizations that are offering (information) goods and services, connecting to end-users through digital channels.

other organizations include:[15] user analytics,[16] UX-capture[17], advertisement,[18] authentication,[19] captcha,[20] performance and (cyber)security, [21] maps and location,[22] search,[23] sales and customer relation management,[24] data as a service,[25] payment,[26] event organizing and ticketing,[27] stockage,[28] shipping,[29], reviews,[30] sharing and social functionality,[31] commenting[32] and embedded media.[33]

The strength and attraction of these third-party services is strongly linked to the fact that these services are structured so they can be offered across curators and domains, at so-called internet scale. For instance, authentication services can use intelligence gathered across websites to the benefit of smaller players unable to gather such information effectively (iOvation 2016). Even a curator with a small end-user count, can benefit from the knowledge gathered by a service operator that serves many many

[15] This is a non-exhaustive list meant to illustrate the argument. The question of what the current array of service components in different online service sectors looks like is the kind of future research that we think needs to happen and is likely to provide further insights into how privacy governance may be organized.
[16] Statcounter (https://statcounter.com/) or market leader Google Analytics (https://analytics.google.com/analytics/web/provision).
[17] Fullstory (https://www.fullstory.com) for user session replays and UX recorder (http://www.uxrecorder.com) for recording the users' face and audio during interaction.
[18] Revenue Hits (http://www.revenuehits.com/) or market leader Google AdSense (http://www.revenuehits.com/).
[19] See e.g. SwiftID by CapitalOne (2-factor authentication) https://developer.capitalone.com/products/swiftid/homepage/ OpenID (http://openid.net/) or Facebook Login (https://developers.facebook.com/docs/facebook-login).
[20] See e.g. sweetCaptcha (http://sweetcaptcha.com/) and market leader Google reCaptcha (https://www.google.com/recaptcha/intro/index.html).
[21] See e.g. CloudFlare (https://www.cloudflare.com/), Symantec's Web security, including Web filtering (https://www.symantec.com/en/uk/web-security-cloud/) or the free and open https as a service, Let's Encrypt (https://letsencrypt.org/).
[22] OpenStreetMap (https://www.openstreetmap.org/) or market leader Google (https://developers.google.com/maps/).
[23] See e.g. Google Custom Search (https://cse.google.com/cse/).
[24] See one of the earliest movers to the cloud, Salesforce (http://www.salesforce.com/).
[25] See e.g. Oracle Data Cloud (https://www.oracle.com/applications/customer-experience/data-cloud/index.html) or Acxiom's LiveRamp connect (http://www.acxiom.com/liveramp-connect/).
[26] See e.g. PayPal's Braintree v.zero SDK (https://developer.paypal.com/).
[27] See Eventbrite (https://developer.eventbrite.com/) or Ticketmaster (http://developer.ticketmaster.com/).
[28] See e.g. Fulfillment by Amazon (https://services.amazon.com/fulfillment-by-amazon/benefits.htm).
[29] See e.g. Deliver with Amazon (for delivery suppliers) (https://logistics.amazon.com/ and UPS Shipping AP (for delivery demand) (https://www.ups.com/content/us/en/bussol/browse/online_tools_shipping.html).
[30] See e.g. Feefo (https://www.feefo.com/web/en/us/).
[31] See e.g. AddThis (http://www.addthis.com/) and Facebook Sharing (https://developers.facebook.com/docs/plugins).
[32] See e.g. Facebook Comments (https://developers.facebook.com/docs/plugins/comments/) or Disqus (https://disqus.com/).
[33] See e.g. Google's YouTube (https://www.youtube.com/yt/dev/api-resources.html) and Soundcloud (https://developers.soundcloud.com/docs/api/sdks).

others. The emergence of third party advertising and tracking over the last two decades is just one example of modularization and its implications for publishers.[34] For small and medium-sized organizations, it is unlikely to be economical to consider in-house development of any of the functionality mentioned above. When in-house development is pursued successfully, it will often make sense to split off the result as a separate service offering. As a result, curators regularly default end-users into other services and the choices these third parties make with respect to privacy governance. In some cases, third party services may integrate services of their own, further decreasing control and oversight over the experience of privacy of end-users. The service environment's response to such complexity inherent to its logic is to do what it knows best: introduce a service. Ghostery, for example, helps curators (and end-users) regain some oversight. Its Marketing Cloud Management service helps curators manage third-party trackers on their websites.[35]

For the developers and operators of software structured as a service, something similar is going on as in the case of the mere curators. Many of the basic service functionality components mentioned above may well be desirable to integrate into a developed software as a service offering. For instance, a music streaming software service like Spotify may integrate sharing features and authentication solutions from third parties such as Facebook, Google or Twitter. It will likely build its own recommendation engine, but may use third party service for the actual streaming of music to mobile users or the analysis of potential fraudulent use. Moreover, those producing software as a service will use a range of developer tools with direct implications for user privacy, e.g. because of their access to usage data. More often than not, those tools themselves will be offered as a service[36] or be integrated into a Platform as a Service (PaaS) offering.[37] These services may sometimes be doubled[38] and replaced on demand depending on developers' needs. As we will discuss in Section 4 and 5 of this chapter, developer tools often continue to be in use after the service has been deployed to end-users.

3.2 Pooling of Data

A central ramification of modularity for privacy is the incentive towards the pooling of end-user data across services and domains by specialized service providers. Such pooling of data, in various modalities, can allow for the most cost-efficient provisioning of core functionality, but results in significant concentration of data in the hands of specialized service providers affecting large numbers of end-

[34] On the displacement of publishers, see e.g. Turow.
[35] See Evidon, Marketing Cloud Management, https://www.evidon.com/solutions/mcm/.The Ghostery brand, which was owned by Evidon, was sold to Cliqz GmbH in early 2017.
[36] Trello https://trello.com
[37] Platform as a service may maintain Software Development Kits (SDKs) or mediate the complete service cloud production cycle as in the case of Cloud Foundry https://www.cloudfoundry.org
[38] A software company may integrate multiple analytics services like Google Mobile Analytics https://www.google.com/analytics/mobile/ and Mixpanel https://mixpanel.com

users.[39] For instance, a recent empirical study found that Google is tracking end-users on 92 out of the 100 most popular websites, as well as on 923 of the top 1,000 websites (Altaweel et al. 2015). More specifically, the researchers found that Google Analytics were tracking visitors on 52; DoubleClick on 73; and YouTube on 19 sites of the top 100 sites (Altaweel et al. 2015).

The business-to-business arrangements between curators and service operators with respect to the use, storage, ownership and analysis of data and the treatment of end-users' privacy interests more generally are crucial to understand privacy governance after the agile turn. Research and data on such relationships, however, is currently minimal.[40] Clearly, increasing market concentration and seeming winner take all dynamics are at play in some of these service markets.[41] Such market power further decreases the leverage that curators have to demand privacy-friendly standards for the treatment of end-users, assuming that they may have some incentives to do so.

What choices do curators have that want to cater to the privacy of end-users? Services like Google data analytics, Facebook authentication and advertising networks are likely to present curators with take it or leave it options. Clearly, such take it or leave it options across different types of curators and end-users are unlikely taking account of the contextual nature of privacy norms. In certain domains, it is still unlikely that data is being pooled at the individualized level, partly, as such pooling of data may amount to a data privacy violation. Generally, however, what starts to emerge is a landscape of what could be called functional data brokers.

3.3 Bundled Relationships

As a direct result of modularity, end-users are increasingly confronted with bundles of service relationships when using digital functionality. The explosive rise of tracking of website visitors is best documented. Recent research established that by merely visiting the top 100 most popular sites, end-users collect a staggering 6,000 HTTP cookies in the process (Altaweel et al. 2015). The researchers conclude that a user who browses the most popular websites "must vet dozens, even hundreds of policies to understand the state of data collection online" (Altaweel et al 2015). Notably, such web privacy measurement studies tend to be limited to the study of tracking that is visible to the researchers

[39] As we discuss in Section 5 the usage of this data increasingly constitutes a core ingredient for developing these services and further strengthens this dynamic.

[40] Examples that come to mind are the Facebook-Datalogix deal that allows advertisers to evaluate the impact of their advertising on in-store sales on an individualized level (see https://www.facebook-studio.com/news/item/making-digital-brand-campaigns-better) or the Foursquare American Express deal that closes the loop between location check-ins and purchases (see http://www.nytimes.com/2011/06/23/technology/23locate.html).

[41] Many of the resulting services, such as Facebook authentication, amount to what economists would call 2-sided market platforms, serving both end-users and organizational users.

(Englehardt and Narayanan 2016). The exchange and collection of data through first and third-party services that are not visible to end-users is typically not accounted for. Advertising networks are known to exchange data amongst each other through ad exchanges, parallel to bidding on advertisement placement options. First parties may also end up having to track and share data about multiple devices belonging to an individual end-user, and precisely monitor their consumption of services on these devices (e.g., number of installations, deletions, amount of use), due to licensing of third party services.

Some of the services that curators default their users into are by now well known to end-users as they are operated by major internet companies that have a large and growing offering of services, such as Google or Facebook. While these internet companies may produce and organize their offering based on the modularity principle, this does not have to be the case for the way in which they negotiate privacy with end-users. For instance, Google consolidated its privacy policy across all its different services into one privacy policy which generally allows user data to be shared amongst the different service components operated by the company (Google 2012), resulting in regulatory action in Europe (Schechner and Ifrati 2012). The company does provide a host of specific privacy controls to end-users with respect to the different service components but in many cases only the use of data for particular features can be controlled. Thus, the benefits that modularity could offer in allowing for the negotiations around privacy with end-users on a more granular level generally do not seem to materialize. What such benefits are, and which privacy engineering solutions could be deployed to accrue these benefits, are questions that a more systematic study of the implications of modularity for privacy could help to answer.

We have used the term end-users to refer to a multiplicity of roles that individuals can have. These roles include the role of the consumer, shopper, employee, student, patient, voter, traveler, fans. The fact that the same services are integrated by curators along a wide variety of domains without differentiation in privacy governance is one of the reasons why privacy may be eroded, or perhaps better, flattened to notice and choice/consent negotiations. The case of curators in the sphere of government and education has received some discussion in the privacy literature. Soghoian, for instance, called attention to the exposure of White House website visitors to YouTube tracking in 2009, in ways that were problematic considering the rules relating to the use of cookies on federal agency Web sites (Soghoian 2009). The White House later reconfigured the YouTube integration to prevent tracking of users that were not watching YouTube videos on the site. In the education context, service providers have also been challenged to adopt more context-appropriate privacy practices. Google Apps for Education switched its Gmail related advertising for students off after a legal complaint about the scanning of emails by students and alumni from the University of California-Berkeley (Brown 2016). In Germany, the integration of Facebook like-buttons on shopping sites has been the subject of successful litigation (Bolton 2016). While these cases indicate that certain actors may use legal or other means to

demand respect for contextual privacy norms from service providers, this result may not be attainable for actors with insufficient economic, social or legal leverage.

3.4 Responsibility for Privacy in a Hyperconnected Service Environment

The modularization of services raises the question of who exactly is and should be responsible to ensure privacy as a matter of policy, law and principle. In the hyperconnected service environments that have emerged over the last decades, this question is non-trivial to answer and policy makers continue to struggle to find the right answers. What responsibility does and should a curator have with respect to the privacy governance of integrated services affecting its end-users? How realistic is it to allocate responsibility to curators, if they have no bargaining power over increasingly dominant service operators? Reversely, what role could the operators of services have in ensuring the use of their offerings respects privacy? And who could and should develop and implement which engineering solutions to address privacy issues in business to business relations?

In Europe, where a comprehensive data privacy framework exists to ensure the lawful, fair and transparent processing of personal data, regulators have struggled to allocate legal responsibility as a result of modularization. European data protection imposes its interlinked obligations on so-called data controllers, i.e. the entity that determines the purposes and means of a particular personal data processing operation. In addition, the EU legal framework entails the concept of processors, which are entities that process personal data under the instruction and authority of data controllers, but do not process personal data for their own purposes.

The Article 29 Working Party, i.e. the EU-level body of independent data privacy regulators, has signaled the willingness of European regulators to attribute significant responsibility to curators.[42] For instance, in the case of behavioral advertising, it has asserted that "publishers should be aware that by entering into contracts with ad networks with the consequence that personal data of their visitors are available to ad network providers, they take some responsibility towards their visitors" (Article 29 Working Party 2010b). While it remains somewhat vague about their precise responsibility, including the extent to which publishers become (joint) data controllers for the processing of personal data by third party advertising networks operating on their website, it has concluded that such responsibility exists at least for the initial processing of visitor data in the case of redirects (Article 29 Working Party 2010b). More broadly, one of the criteria the Article 29 Working Party has developed for establishing who is the controller, is whether there is control over the processing of personal data as a result of 'implicit competence' (Article 29 Working Party 2010a). An educational institution's implicit competence over

[42] Cloud computing services, including IaaS subcontracting, has led to a similar (ongoing) discussion of controller and processor responsibilities under EU data protection law. Notably, the GDPR contains additional obligations for processors.

the processing of (its) student data or the employer's implicit competence over the processing of its employee's data (by third party services) could be seen as examples of this. In practice, this principle would imply that end-user facing curators have significant responsibility over the data privacy governance by service operators they have integrated. While EU regulators have taken a similar stance over the obligations of curators in service integration decisions, we still see little evidence that they have made much progress in enforcing this principle.

4. Temporality

In addition to modularization, the agile turn comes with a number of changes in the temporal relationships between end-users and the service developers and operators. First, in shrink-wrap software the transaction between developers and end-users is limited to a short moment at the point of sale (or download). With services, the transaction gets prolonged throughout the use, and, when it comes to privacy, sometimes beyond that. Second, service users are subject to a continuously evolving relationship with a cascade of curated services -- an end-user facing service may integrate or remove third party services, which similarly may integrate or remove third parties. Finally, any such service is optimized over time to capture relevant user activities and interactions through the management of a dynamic feature space.[43] Consequently, the distinction between the production and use phase of digital functionality that was inherent to shrink-wrap software is blurred significantly.

The blurring of the distinction between the production phase and use in combination with the acceleration in the dynamic production of services has serious implications for privacy governance. Privacy governance still implicitly and predominantly relies on this distinction, from the era of the waterfall model and its temporal underpinnings. To explore these implications for privacy governance, we juxtapose some of the temporal assumptions that seem to underlie the agile turn. This helps to surface certain gaps in current privacy theory and data protection regimes that need to be addressed in research as well as practice.

4.1 Temporality and Privacy

Privacy theories and models tend to consider spatial aspects of privacy more explicitly than temporal ones. Specifically, visibility, location, intimacy and information flow (from A to B in an abstract space) are prioritized, while temporal aspects are often only addressed incidentally or implicitly. It is not that these aspects are separable. Every spatial description has a temporal aspect: when we discuss being in public or private space, time is always implied. At the level of privacy theory, even if contextual integrity "is not conditioned on dimensions of time, location, and so forth" (Nissenbaum 2009), it could still be seen as a

[43] For our discussion of capture, see Section 5.

theory that requires looking back in time with the intent of identifying the relevant social norms for the appropriate flow of information, that can inform the design of future socio-technical systems. This complex time construction that is inherent to contextual integrity nicely illustrates that time is not linear but a complex concept that may provide a rich lens through which to consider what is happening in the increasingly dynamic privacy relationship between end-users and services.

Data privacy regimes, such as the FIPPS or EU data protection can at times be more explicit with respect to temporal aspects of the different principles. Data retention and the so-called right to be forgotten are clearly expected to regulate how long into the future captured data can be projected. Yet, data protection principles often assume the use of the planned and up-front design development and the associated long production phases of shrink-wrap software. For example, it is common to hear that privacy by design is to be applied from the very beginning of software design and not as an afterthought when the system is ready for deployment. However, "the beginning" of digital functionality that is offered as a bundle of services is hard to establish, and even if it could be established, not the only moment at which privacy by design is required. In addition, it seems likely that "the end" of a service, a service component, or the removal of a feature may be just as relevant for privacy.

Notice and choice/consent and purpose limitation all assume (for their effectiveness) that the functionality on offer can be stabilized enough to present to the users and that relevant changes to the functionality are rare enough to make a renegotiation of consent feasible. The introduction of frequent updates and dynamic permissions that prompt users during the use of a service show that for the current software ecosystem, the consent of a user may be under continuous re-evaluation. Absent a substantial stability in the service, these principles can easily lose meaning in establishing privacy protection and lead to symbolic industry practices (Van Hoboken 2016). As a result of 'big data', many have already given up on the possibility to continue to give meaning to these principles (Cate and Mayer-Schönberger 2012). The widely heard calls for the focus on the regulation of the use of personal data instead of the regulation of collection could be seen as a response to the temporal dimension of privacy, although these proposals may concede too much to be taken as serious proposals to address privacy (Nissenbaum 2016), as opposed to throwing in the towel.

Some privacy scholars have explicitly discussed temporal aspects of privacy and design. Palen and Dourish build on Altman's theory of privacy management to argue that people inform their privacy practices in the present based on past experiences as well as in anticipation of future actions (Palen and Dourish, 2003). In contrast, Agre's theory of capture tries to understand privacy in relation to the practice of producing socio-technical systems. More specifically, the 'grammar of action' model claims that the reorganization of existing activity is inherent to the development and operation of socio-technical systems (Agre 1994, 11). This means that the design of functionality builds on past activities, but also implies that observation of end-users allows for the regulation of future activities in ways that

may infringe upon end-users' autonomy. The introduction of the Facebook like button, its dissemination across the web, and its recent diversification into "Facebook reactions" can be seen in this light as one instance of the evolution of regulation of "self-expression" over time (Kant, 2015).

4.2 A Dynamic Environment of Services and Features

The temporal dimension plays out in relation to privacy in digital functionality at multiple levels thanks to the three shifts in software production. An agile service operator can shape the temporal relationship with its users by successfully leveraging the increased control developers have in curating and designing the server-side functionality. Thanks to the high degree of modularity, which allows for the loose coupling of service components, service curators can integrate, switch or remove services throughout deployment. Further flexibility and scalability is achieved by relying on the shift to cloud computing and the available services for development and programming, in addition to dynamic functionality and associated business development.

Specifically, agile programming practices allow developers across services to continuously tweak, remove, or add new features using "build-measure-learn feedback loops".[44] Weekly sprints, scrums and daily standup meetings are the rituals of this accelerated production of features that is unleashed into the world.[45] This includes experimental features, minimum viable products, alpha releases, and may be best captured by the term 'perpetual beta', which stands for a never ending development phase (O'Reilly 2005). Minor changes to existing features happen daily, while major changes can be introduced every two weeks to two months.[46] For example, Microsoft Bing boasts that they are "deploying thousands of services 20 times per week, with 600 engineers contributing to the codebase", "pushing over 4000 individual changes per week, where each code change submission goes through a test pass containing over 20,000 tests".[47] For smaller companies the numbers are of course smaller, but the time intervals of daily tweaking, weekly releases and quarterly feature overhauls are common.

[44] "Build-measure-learn feedback loops" is a term coined by lean methodology proponents who claim that "[t]he fundamental activity of a startup is to turn ideas into products, measure how customers respond, and then learn whether to pivot or persevere". See http://theleanstartup.com/principles/.

[45] Research as well as the interviews we conducted in preparation of this paper show that there are great differences between start-ups and large companies in how agile methods are applied. Often there is a divergence between the ideals of the manifestos for agile programming and actual practice, which we do not discuss further. We are also brushing over the challenges of applying agile methods in distributed teams, where knowledge, economic, and racial hierarchies exist between those developers located in Silicon Valley or its competitor sites in the Global North and those predominantly located in the Global South.

[46] While we make it sound like rapid feature release is an easy feat, in reality it is a complex management challenge and may result in problems such as overproduction (see e.g. https://www.oreilly.com/ideas/overproduction-in-theory-and-practice) and feature drifts.

[47] See http://stories.visualstudio.com/bing-continuous-delivery/.

Yet, the constant evolution of the feature space is not just a compulsive programming activity but a way to situate the business in the marketplace. Companies big and small will opt for new features to gain advantage over or to catch up with competitors. Features provide the distinctive flavors that distinguish one service from another. Venture Capitalists may evaluate investments based on feature portfolios.[48] Given these influence factors, dominant companies and investors in a given market can play an important role in feature design trends across the industry.

In practice, the management of the introduction, change or removal of features is mainly the responsibility of product managers who keep an eye out on competitors or talk with clients to drive decisions on new features. Depending on their standing in the company, user experience (UX) engineers or sales teams may also ask for new features. The demand for features may also originate from the developers themselves. For example, due to accelerated production, engineering teams may incur what is coined *technical debt*: quality of engineering suffers due to speedy production and code hacks to release features which may lead to increasing costs, system brittleness and reduced rates of innovation.[49] Features may be redesigned to pay off technical debt accrued from rapid feature release. In total, features provide refined and flexible units for defining and evaluating business ambitions in sync with development activities.

c. Changes of the feature space and autonomy

For end-users, the result of this feature inflation means that they may interact with features that look the same but do different things, interact with multiple features that do the same thing, and in some cases, may see their favorite features simply disappear. On the surface, using a service implies agreeing to change in functionality across time. Under the hood, it implies that the relationship between a user and services is constantly reconfigured by a line-up of service providers, their curatorial choices in services, as well as agile programming practices.

This continuous reconfiguration is barely communicated in privacy policies and terms of use and is generally not open to negotiation. In this constellation, informed consent runs into its temporal limits. Even if the service bundle remains stable, changes to features means that the information captured by the services is easily likely to be repurposed. This raises the question whether and under what conditions changes to features require re-establishing informed consent.

So, what happens to the relationship between end-users and services once the service is no longer in use? In general, data privacy regimes are very concerned with the governance of information collection

[48] And their effectiveness and scalability in terms of capturing unique user data.
[49] Technical debt may include the intensification of interdependencies that break modularity, in which case teams may start talking about concepts like “changing anything, changes everything” (Sculley et al. 2014).

and processing activities, but they are silent as to what happens when a functionality is removed or a service relationship ends. Depending on the functionality, removal of features may impact end-users' ability to complete tasks, organize their work with and in relation to others, or access related data. In contrast, with every removal of a service from a curator's bundle, the relationship between the user and the removed service becomes ambiguous.

Finally, the conflation of production and use may have its advantages, too. If a feature is seen as a privacy disaster, it can be easily redesigned, or simply removed. At the same time, the ability to silently change features also means that service providers are more susceptible, for example, to requests from government agencies to design features or break privacy functionality for surveillance purposes, as was the case for Lavabit (Van Hoboken and Rubinstein 2014). We also heard many examples of service providers being subject to the interests of intellectual property holders. Such developments are not independent of these temporal shifts inherent to the production of services. The increased ability of service providers to fiddle with functionality at the request of powerful third parties, together with their ability to use features to optimize user activities to business goals illustrates that services are likely to amplify potential infringements upon the autonomy of users.

5. Capture

The agile turn takes the capture model to its logical extreme. Capture is an alternative privacy concept developed by Agre in the 1990s to highlight the conditions of privacy in relation to the construction of socio-technical information systems (Agre 1994). Capture involves the reorganization of everyday human activities through improvements of their legibility and evaluation for economic purposes. Specifically, it implies the development and imposition of 'grammars of action' -- specifications of possible activities enabled by systems and that can be mixed and matched by users -- that, when put into use, can come to reconfigure everyday activities while subjecting them to commodification and economic incentives. In contrast to surveillance models of privacy that focus on how technological advances intrude upon private space and that derive from histories of government surveillance, the capture model has its roots in automation, industrial management (including Taylorism) and systems engineering.

In this section, we question the reasonableness of treating information flows as the central concern to privacy, and as something that can be discussed independent of the design and production of functionality. Leveraging capture, service providers may accumulate power through their ability to reorganize and optimize user activities, leading to a host of other problems. We argue that discussions of privacy need to consider capture through service and feature production in a way that recognizes their impact on user activities, autonomy, labor, and markets.

5.1 The Ingredients of Capture

Agre described how capture is produced based on software engineering practices in the beginning of the 90s, well before the agile turn. He defines capture in five (potentially iterative) phases. In phase one, someone *analyzes* human activities of interest and their fundamental units. For example, a requirements engineer may study a bank to identify relevant activities which may include opening an account, and making and auditing transactions. Next the developer team *articulates* this ontology in a technical system with which end-users will be able to put those units of activity together into possible *grammars of action*. When eventually the bank deploys this system, these designed units of action are *imposed* onto the users and employees of the bank who inevitably have to reorganize their activities (or resist to do so) in a way compatible with the grammars of action inherent to the system. The bank system can now maintain a running parse of all activities that can be *instrumented* through their system. Once trackable, records of the reorganized activity of the bank customers and employees can be stored, inspected, audited, merged and employed as the basis of optimization, performance measurement, and quality assurance (*elaboration*).

In comparison to the 1990s, the way capture functions has transformed in technical and economic terms, due to the modularization and the new temporalities of software and service production. It is realized through an assemblage of features, services and multilayered testing in which the end-users are being tracked and enlisted to test capture's efficiency. Specifically, every feature is a real-time probe into the continuous reorganization of users' activities. End-user activity becomes a key ingredient in managing the system. Or, as Tim O'Reilly asserts: "users must be treated as co-developers" (O'Reilly 2005). The mantra "release early and release often" translates to an imperative to closely monitor how features are used to decide whether and how they will remain part of a service. As O'Reilly further highlights, "if users don't adopt [features], we take them down. If they like them, we roll them out to the entire site" (O'Reilly 2005).

This refashioning of the user into a co-developer and evaluator is worthy of greater attention. When features are offered to millions of users, data analysts state that user anecdotes are not a reliable source of intelligence for the developers. Instead, captured data are assumed to reflect users' desires, interests, constraints and opinions. Thus, user (inter)actions as captured by the system can easily be conflated with user intentions. As a result, people are made complicit in the capturing and re-organization of their own behavior and in the "rapid development of features that are able to identify, sequence, reorder and transform human activities" (Agre 1993). Again, this also illustrates that captured data is not a mere "byproduct of the digital mediation of otherwise naturally occurring activities. The data are, at least in part, evidence of the purposeful design of the system that 'happens' to generate them" (Barocas 2012).

5.2 The Merging of Digital Functionality and Data

The tight feedback loops between features and captured data means that they may eventually melt into each other. In order to capture the success of a feature, developers may need to develop new metrics. For example, sometimes developers may want to know if the user has "considered" the contents of a new feature, say a new information box, before taking an action. Researchers have shown that eye-mouse coordination patterns are reliable enough to provide just such an indicator (Rodden et al. 2008). So, now the developer may integrate an analytics service that captures mouse movements to evaluate the use of the information box. Once the service is integrated, mouse movements may be used in the evaluation of other features, too.

Beyond evaluation, captured data becomes a key-ingredient in producing services and their features. For example, traditionally authentication would be a binary decision problem, i.e., matching user credentials to an authorization database. After the agile turn, it can be turned into a classification problem based on user keystrokes, browser information, and time of login. Mouse movement data, for example, can be used for continuous authentication. This example illustrates that user and behavior analytics and A/B testing that were essential for the evaluation of data products like recommender systems, search ranking, and voice recognition, are increasingly incorporated into the core functionality. Finally, all the captured data can support business agility. Data analysts in small startups may use captured data and behavioral data to analyze user churn or compute pricing, as well as for evaluating future programming efforts.

5.3 Grammars of Action, Flexible Ontologies and Testing

Agility demands that capture operates recursively and in a multi-layered fashion. Specifically, service operators need to keep track of their users and their activities. They need to keep track of their service components. But they also need to keep track of how well they are keeping track of their users and service components. Each service that a feature resides in is a specialized unit that is loosely coupled into a dynamic ontology of market and organizational activities. Rapid feature release and loose coupling of services (and components) simplifies programming and management but creates complex interdependencies that need to be orchestrated to produce a seamless and adaptive environment for user tracking in support of the grammars of action that are imposed on the users.

Mastering capture involves an elaborate system of automation and testing. New software may be released dozens of times a day, where each release may be subject to 1000s of automated tests[50]. This

[50] In a popular blog post from 2009, IMVU developer Timothy Fitz writes: "We have around 15k test cases, and they're run around 70 times a day." (Fitz, 2009). How well such testing is integrated across the industry, e.g., in companies of different sizes and maturity, are a topic of future research.

shifts the problem of managing interdependencies and user tracking to the management of tests. Specifically, while service and feature optimization is realized through A/B testing, user and behavioral analytics, continuous software delivery is guaranteed through a host of internal tests that track interfaces, dependencies and users' response to changes.[51] Capturing user behavior and tracking the different service components bleeds into capturing the behavior of the operator's capture models. In conclusion, with the agile turn, the mastery of keeping track of tracking becomes central to the production of digital functionality.

5.4 Limitations of Privacy as Information Flow or Accumulation

If every form of digital functionality has the potential to be transformed into a data intensive machine learning product, the application of principles like data minimization, purpose specification, and policies that enforce some sort of control over data on the side of end-users is going to be challenging. At the same time, much of service capture creates prime opportunities to apply privacy technologies like differential or pan privacy. Application of these techniques could protect users from individual harms due to re-identification of data. Furthermore, the interviews taught us that metrics for user behavior in services are still in the developing phase, meaning this field is open for greater study, theorizing and regulatory intervention. Policy makers could pay attention to services that provide detailed user analytics, including intrusive techniques that facilitate the replay of individual users' gestures and video of their environment.[52] Computer scientists can leverage their techniques to provide automated privacy support that allow users to evaluate which information flows they want to engage in and how they can control these when they use services. However, all of these measures fall short of addressing the privacy implications of the capture model.

The conflation of user intentions with observations of their actions raises a fundamental question of what it means to consider users as stakeholders into the governance of services. Just because users are tracked very intensely, doesn't mean they can express themselves. Studies that purport the "privacy paradox", that users say they want their privacy but then act otherwise, are symptomatic of the same ideology that suggests that actions are intentions. It is a challenge for privacy research to express the

[51] Developers are often encouraged to write their unit tests before writing their code. Further tests include integration testing, system testing, acceptance testing, and regression testing. The last one refers to making sure that a new release is compatible with past versions (Humble and Farley, 2010). The automation of testing and management of testing results has become a science in itself (Boyapati et al. 2002).

[52] FullStory captures every detail of your customer's experience on your site or web app. http://www.fullstory.com; Jaco, "We Collect Everything. Jaco records and plays exactly what your users are seeing. No matter how complex your application is." http://getjaco.com; Mixpanel, "Mixpanel gives you the ability to easily measure what people are doing in your app on iOS, Android, and web." http://mixpanel.com.

fundamental damage of such positions and the practice of user capture to people's autonomy and human dignity (Rouvroy and Poullet 2009, Cohen 2012).

Finally, we are doubtful that constraints on information flows, derived from contextual values or regulatory frameworks, will be able to fully address the implications of services for the reorganization and optimization of everyday activities. First, this would require the development of a framework for evaluating the agile production practices using appropriate information flows, which will be a considerable challenge. Data privacy regimes rarely attend to the conditions of production, nor do they easily address the implication of users being enlisted as labor in the production of services through the process of capture. The experimentation with user populations in the process of feature production requires a debate on the conditions under which these practices remain legitimate under the disguise of digital innovation. All in all, given how fundamental services have become to people's everyday lives, their work activities, their education and their enjoyment of (health)care, the agile turn raises the question whether other, or complementary regulatory approaches are called for. Rather than turning them into questions of information flows and data privacy, consumer protection, software regulation or the treatment of certain services as new types of utility providers may be better suited to address these deeper challenges associated with the agile turn.

6. Conclusion

In exploring questions of privacy governance in light of the agile turn, much of this chapter's attention has been directed to framing its consequences and the best ways to start exploring these in more depth. We conclude that modularity, temporality and capture are central to understanding the way in which digital functionality comes into the world and affect privacy and the conditions for its governance. This is the case from a regulatory as well as technical and organizational perspective.

The modularity in production presents us with a fine paradox of sorts. There is significant independence of basic service components but at the higher level and the way in which the world presents itself to end-users, interdependency is king. Service curators are key in production and default end-users into bundles of service relationships. Production has become increasingly dynamic and puts stress on the temporal dimensions of privacy. It also drives the capture of user activities and data to a new level. End-users are often depicted as mere consumers, while the capture of their activity and the data derived from it has become an essential ingredient in service production.

Much of privacy scholarship has decoupled the consideration of information flows from the dynamic construction of functionality for users, not engaging in depth in the latter. Not only should privacy scholarship pay more attention to how current-day software is produced, it should also more centrally

focus on what functionality ends up being offered and how it reorganizes user activities. The study of algorithms as a lens is likely to fall short of addressing the complexities of the production of different types of digital functionality for similar reasons. We believe that our argument for studying the conditions of production is more broadly applicable, especially considering the collapse of the distinction between consumption and production we discuss in this chapter.

Finally, this chapter forcefully raises the question, what are the kind of privacy solutions that are most likely to work after the agile turn. Which aspects of production should be embraced as they allow for effective privacy governance and which aspects should be constrained since they are incompatible? Or should the focus simply be placed on creating alternatives to centralized data-driven service production? From the industry itself, we expect the further emergence of 'privacy as a service' offerings in which specialized services, internally and externally, help curators and operators comply with data privacy regulations and the strategic challenges inherent in intrusive and data-intensive offerings. Perhaps the production of privacy-friendly alternatives should be stimulated for essential types of functionality. However, the economics of service modularity pose a significant challenge for independent privacy applications. Privacy is not the "doing one thing really well" kind of problem.

References

Agre, Philip E. "Surveillance and capture: Two models of privacy." *The Information Society* 10, no. 2 (1994): 101-127.

Altaweel, Ibrahim, Nathan Good, and Chris Jay Hoofnagle. "Web Privacy Census." *Technology Science (2015)*. Available at http://techscience.org/a/2015121502/.

Anthonysamy, Pauline, and Awais Rashid. "Software engineering for privacy in-the-large." In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2, pp. 947-948. IEEE, 2015.

Article 29 Working Party (2010a). "Opinion 1/2010 on the concepts of 'controller' and 'processor", Brussels, 2010. Available at http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2010/wp169_en.pdf

Article 29 Working Party (2010b). "Opinion 2/2010 on online behavioural advertising", Brussels, 2010. Available at http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2010/wp171_en.pdf

Article 29 Working Party. "Opinion 05/2012 on Cloud Computing", Brussels, 2012. Available at http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2012/wp196_en.pdf.

Baldwin, Carliss Y. "Modularity and Organizations." In: *International Encyclopedia of the Social & Behavioral Sciences*. Elsevier, 2nd ed. (2015): 718–723.

Bamberger, Kenneth A. and Deirdre K. Mulligan. *Privacy on the Ground Driving Corporate Behavior in the United States and Europe*. MIT Press, 2015.

Barocas, Solon. 'Big data are made by (and not just a resource for) social science and policy-making' (Abstract), *Internet, Politics, Policy 2012 Conference*, Oxford University (2012). Available at http://ipp.oii.ox.ac.uk/2012/programme-2012/track-c-data-methods/panel-1c-what-is-big-data/solon-barocas-big-data-are-made-by-and

Black, Benjamin. "EC2 Origins". Jan 25, 2009. Available at http://blog.b3k.us/2009/01/25/ec2-origins.html.

Blanchette, Jean-François (2015). "Introduction. Computing's Infrastructural Moment". In: Christopher S. Yoo & Jean-François Blanchette (eds.), *Regulating the Cloud. Policy for Computing Infrastructure*, The MIT Press. Cambridge, MA, p. 1-21.

Bolton, Doug. "Facebook 'Like' button may be against the law, German court rules." *The Independent*, Mar 10, 2016. Available at http://www.independent.co.uk/life-style/gadgets-and-tech/news/facebook-like-button-illegal-data-protection-germany-court-a6923026.html

Bonneau, Joseph, Edward W. Felten, Prateek Mittal, and Arvind Narayanan. "Privacy concerns of implicit secondary factors for web authentication." In *SOUPS Workshop on "Who are you*. 2014.

Boyapati, Chandrasekhar, Sarfraz Khurshid, and Darko Marinov. "Korat: Automated testing based on Java predicates." In *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 4, pp. 123-133. ACM, 2002.

Brown, Emma. "UC-Berkeley students sue Google, alleging their emails were illegally scanned." *The Washington Post*, Feb 1st, 2016. Available at https://www.washingtonpost.com/news/grade-point/wp/2016/02/01/uc-berkeley-students-sue-google-alleging-their-emails-were-illegally-scanned/

Carlson, Lucas. "4 ways Docker fundamentally changes application development." *InfoWorld*, Sept 18, 2014. Available at http://www.infoworld.com/article/2607128/application-development/4-ways-docker-fundamentally-changes-application-development.html.

Cohen, Julie E. "What privacy is for." *Harv. L. Rev.* 126 (2012): 1904.

Cate, Fred H., and Viktor Mayer-Schönberger. "Notice and consent in a world of Big Data." *International Data Privacy Law* 3, no. 2 (2013): 67-73.

Clark, David D., John Wroclawski, Karen R. Sollins, and Robert Braden. "Tussle in cyberspace: defining tomorrow's internet." In *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 347-356. ACM, 2002.

Cloud Native Computing Foundation (2016). Available at https://cncf.io/.

Dörbecker, Regine, and Tilo Böhmann. "The Concept and Effects of Service Modularity--A Literature Review." In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pp. 1357-1366. IEEE, 2013.

Douglass, Bruce P. *Agile Systems Engineering*. Morgan Kaufmann, 2015.

Englehardt, Steven & Arvind Narayanan. "Online tracking: A 1-million-site measurement and analysis". In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, (2016): 1388-1401.

Erickson, Seth and Christopher M. Kelty. "The Durability of Software", In Irina Kaldrack and Martina Leeker eds., *There is no Software, there are just Services*, Meson Press, 2015.

Estler, Hans-Christian, Martin Nordio, Carlo A. Furia, Bertrand Meyer, and Johannes Schneider. "Agile vs. structured distributed software development: A case study." *Empirical Software Engineering* 19, no. 5 (2014): 1197-1224.

Exposito, Ernesto and Code Diop. *Smart SOA Platforms in Cloud Computing Architectures*, John Wiley & Sons, 2014.

Fay, Joe. "Assembly of tech giants convene to define future of computing". *The Register*, Dec 18, 2015. Available at http://www.theregister.co.uk/2015/12/18/cloud_native_computer_cloud_native/.

Fitz, Timothy. "Continuous deployment at IMVU: Doing the impossible fifty times a day." *Blog Post, February* 10 (2009). Available at https://www.timothyfitz.com Last visited: May 16, 2016

Fuchs, Christian. "Class and Exploitation on the Internet." *Digital Labor. The Internet as Playground and Factory* (2013): 211-224.

Google. "Updating our privacy policies and terms of service". *Official Blog*. Jan 24, 2012. Available at https://googleblog.blogspot.com/2012/01/updating-our-privacy-policies-and-terms.html.

Google. "Ten Things We Know To Be True." (2016). Available at https://www.google.com/about/company/philosophy/

Gürses, Seda, and Jose M. del Alamo. "Privacy engineering: Shaping an emerging field of research and practice." *IEEE Security & Privacy* 14, no. 2 (2016): 40-46.

Helmond, Anne. "The Platformization of the Web: Making Web Data Platform Ready." *Social Media+ Society* 1.2, 2015.

Humble, Jez, and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

Fox, Armando and David Patterson. *Engineering Software as a Service: An Agile Approach Using Cloud Computing*, Strawberry Canyon LLC, 2nd ed. Edition, 2013.

Hook, Leslie. “Cloud computing titans battle for AI supremacy”, *Financial Times*, Dec 5th, 2016. Available at https://www.ft.com/content/bca4876c-b7c9-11e6-961e-a1acd97f622d.

iOvation. “iOvation Launches Passwordless Security for Consumer-Facing Websites.” Feb 29th, 2016. Available at https://www.iovation.com/news/press-releases/iovation-launches-passwordless-security-for-consumer-facing-websites.

Jamieson, Jack. "Many (to platform) to many: Web 2.0 application infrastructures." *First Monday* 21, no. 6 (2016). doi:10.5210/fm.v21i6.6792.

Kaldrack, Irina and Martina Leeker. “Introduction.” In: Irina Kaldrack and Martina Leeker eds., *There is no Software, there are just Services,* Meson Press, 2015.

Kant, Tanya. "FCJ-180 ‘Spotify has added an event to your past’:(re) writing the self through Facebook’s autoposting apps." *The Fibreculture Journal* 25 2015: Apps and Affect (2015).

Knorr, Eric. “2016: The year we see the real cloud leaders emerge”, *InfoWorld*, Jan 4th, 2016. Available at http://www.infoworld.com/article/3018046/cloud-computing/2016-the-year-we-see-the-real-cloud-leaders-emerge.html.

Fowler, Martin, and James Lewis. "Microservices a definition of this new architectural term." (2014) Available at http://martinfowler.com/articles/microservices.html.

Mahoney, Michael S. "Finding a history for software engineering." *IEEE Annals of the History of Computing* 26, no. 1 (2004): 8-19.

Neubert, Christoph. “‘The Tail on the Hardware Dog’: Historical Articulations of Computing Machinery, Software, and Services”, In Irina Kaldrack and Martina Leeker eds., *There is no Software, there are just Services*, Meson Press (2015).

Newcomer, Eric, and Greg Lomow. *Understanding SOA with Web services*. Addison-Wesley, 2005.

Nissenbaum, Helen. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press, 2009.

Nissenbaum, Helen. "Must Privacy Give Way to Use Regulation?" *Lecture at the Watson Institute, Brown University, March* 15 (2016): 2016. Available at http://watson.brown.edu/events/2016/helen-nissenbaum-must-privacy-give-way-use-regulation.

O'Reilly, Tim. "What is web 2.0: Design patterns and business models for the next generation of software. Retrieved March 2006." (2005). Available at http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html.

Palen, Leysia, and Paul Dourish. "Unpacking privacy for a networked world." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 129-136. ACM, 2003.

Parson, D. "Agile software development methodology, an ontological analysis." (2011).

Pearson, Siani. "Taking account of privacy when designing cloud computing services." In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pp. 44-52. IEEE Computer Society, 2009.

Pringle, Tom, T. Baer, and G. Brown. "Data-as-a-service: the Next Step in the As-a-service Journey." In *Oracle Open World Conference*, 2014.

Rodden, Kerry, Xin Fu, Anne Aula, and Ian Spiro. "Eye-mouse coordination patterns on web search results pages." In *CHI'08 extended abstracts on Human factors in computing systems*, pp. 2997-3002. ACM, 2008.

Roy, Jacques, (2014). *The Power of Now: Real-Time Analytics and IBM Infosphere Streams*. McGraw Hill Education, 2015.

Rouvroy, Antoinette, and Yves Poullet. "The right to informational self-determination and the value of self-development: Reassessing the importance of privacy for democracy." *Reinventing data protection?* (2009): 45-76.

Schechner, Sam and Amir Efrati. "EU Privacy Watchdogs Blast Google's Data Protection". *The Wall Street Journal*, Updated Oct. 16, 2012.

Scholz, Trebor, ed. *Digital labor: The Internet as playground and factory*. Routledge, 2012.

Sculley, D., Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. "Machine Learning: The High-Interest Credit Card of Technical Debt." (2014).

Shilton, Katie, and Daniel Greene. "Because privacy: defining and legitimating privacy in ios development." *IConference 2016 Proceedings* (2016).

Simon, Herbert A. "The Architecture of Complexity." *Proceedings of the American Philosophical Society* 106, no. 6 (1962): 467-482.

Stutz, David. "The Failure of Shrinkwrap Software." (2003). Available at http://www.synthesist.net/writing/failureofshrinkwrap.html.

Van Hoboken, Joris and Ira Rubinstein. "Privacy and Security in the Cloud: Some Realism About Technical Solutions to Transnational Surveillance in the Post-Snowden Era." *Maine Law Review* 66 (2013): 488.

Van Hoboken, Joris. "From Collection to Use in Privacy Regulation? A Forward Looking Comparison of European and U.S. Frameworks for Personal Data Processing." *Exploring the Boundaries of Big Data* (2016): 231.

Xu, Lai, and Sjaak Brinkkemper. "Concepts of product software: Paving the road for urgently needed research." In *the First International Workshop on Philosophical Foundations of Information Systems Engineering (PHISE'05)*, pp. 523-528. FEUP Press, 2005.

Yegge, Steve. "Stevey's Google Platforms Rant." (2011). Available at https://gist.github.com/chitchcock/1281611.

Yoo, Christopher S., and Jean-François Blanchette, eds. *Regulating the Cloud: Policy for Computing Infrastructure*. MIT Press, 2015.

Ziewitz, Malte. "Governing Algorithms Myth, Mess, and Methods." *Science, Technology & Human Values* 41.1 (2016): 3-16.

Zittrain, Jonathan. *The Future of the Internet And How to Stop It*. Yale University Press, 2008.