Articles from Computational Culture

Objects of Intense Feeling: The Case of the Twitter API

2013-09-15 15:09:48 andrew

Introduction

The past decade has seen a staggering rise of social media – online services that facilitate social interaction between its users. We live and breathe social media, as services like Facebook and Twitter have not only become household names, but something like actual households themselves – places people choose to live and socialize. Whilst much indeed has been said about these so-called social media, by and large, media researchers continue to treat these media as proxy for studying something else, whether this be privacy, self-presentation, interpersonal relations etc. Not only have social media become the theme of much media scholarship they increasingly also constitute the *tools* for this research. One particularly popular mode of research in this regard utilizes aesthetically-pleasing and often very seductive visualizations of social media networks, created from data collected using the application programming interfaces (APIs) provided by corporations like Google, Facebook and Twitter. While there is nothing wrong with using APIs to collect data, of course, researchers should be wary about letting any current obsessions with big data overshadow the fact that APIs are far from neutral tools.

It is exactly here, at the moment of apparent tool-blindness, that a software studies perspective becomes particularly important. With an emphasis on *software criticism*, rather then mere software *use*, bringing a software studies perspective to bear on data collection 'tools' like APIs, seems imperative if we want to understand what these 'tools' do and the politics and powers they entail, beyond helping to collect and provide access to the data and functionality contained by social media platforms. However, software criticism should not be taken as a term describing a critique of material determinants only, no matter how important these may be. Rather, and in the spirit of software studies as a field of inquiry that seeks to understand the mutable and contingent nature of software, advancing a critical understanding of the 'tools' themselves, implies much more than merely a critique of its material support. Given the expanded definition of software mobilized within software studies, as a 'neighborhood of relations', being concerned with the 'stuff of software' in the sense describes by Matthew Fuller, may quite easily mean a number of different things.[1] Only by contributing with theoretical and empirical accounts of specific stuff and their relations can we begin to get a better picture of the neighborhoods in question. This article represents one such contribution to the field, by exploring the 'stuff' of Twitter's API, its specificity as a *protocological* software object, in terms of how it does work in the world. In so doing, the aim is to contribute a better understanding of the 'platform politics' of social media, or indeed, the neighborhood of relations and negotiations entailed by APIs.

So far software studies has been particularly successful in exploring the materiality of software, through the study of its technical properties. However, less attention has been paid to how people experience various attributes of code. Taking up Kitchin and Dodge's call for the need to develop more detailed ethnographic studies of how developers produce and make sense of code, what is of interest in this article is how third-party developers view and understand the APIs that they are using, and how we may begin to understand the work that APIs perform?[2]

APIs in general, and web APIs in particular, are interesting to study for a number of reasons. First of all, APIs provide the condition of possibility for sharing content and data online. As protocological objects, APIs allow interested parties to access the data and functionality of popular online services, all in a very controlled manner. Protocol, as Alexander Galloway argues, is not merely a technical specification regulating how data can be exchanged on a network. Rather, protocol must be understood as a management style, a technique for governing the relations it contains.[3] Understood in this way, APIs not only participate in governing the transmission and exchange of information in networks. In doing so, APIs have 'politics', meaning that they can be seen as having 'powerful consequences for the social activities that happen with them, and in the worlds imagined by them'.[4] Moreover, Tarleton Gillespie writes, 'once we can see artifacts as crystallized forms of human labor, communication, and value, the importance of how they shape activity becomes clearer'.[5]

Starting from these premises, this article seeks to open up a line of inquiry into the specificity of APIs as protocological objects, asking not so much what APIs are, but of what they do. In doing so, this article

picks up on one of the core concerns for software studies, namely the question of agency – of the enactive powers and specific agencies of software. Sympathetic to the view that software has a 'variable ontology', the specificity of APIs cannot be determined technical features alone.[6] As Michael Callon emphasizes, 'the agents, their dimensions, and what they are and do, all depend on the morphology of the relations in which they are involved'.[7] However, studying the enacting powers of objects from a relational point of view is not without its problems. Mike Michael says, '"choices" have to be made as to what to include and exclude in its composition'.[8] Tracing associations and making decisions about which relations and which actors to include in the study of hybrid objects, is ultimately an analytical fabrication. The point is not to account for 'the empirical accuracy' of the hybrid, but rather to employ a 'heterogeneous perspectivism', with which one seeks to find evidence of ordering and disordering from the varying perspectives of the relevant agencies involved.[9]

This article thus reports on the following choices made in studying the enactive powers of the Twitter APIs. First, it offers an account of the specificity of APIs in terms of its *sociomateriality*, meaning that APIs are understood as historical contingent arrangements of social and material components that coalesce to produce new realities.[10] Next, the article provides a description and analysis of an empirical study of the Twitter API, based on interviews with third-party developers. What emerged from the interviews was the sense that what APIs are and the work they perform, to a certain extent, needs to be understood as discursively constructed through the collective qualifications of developers using the APIs. In the final part, this article offers a discussion of the enactive powers of the Twitter APIs, by drawing on Michael Serres' notion of the 'quasi-object' as one particularly useful way to account for the relationality at work in an utterly sociomaterial world. The notion of the quasi-object designates a way of framing objects as active participants in social relationships, rather than as passive end-points of human action. For as Serres writes, 'our relationships, social bonds, would be airy as clouds were there only contracts between subjects'.[11] One needs operators that mediate collective and individual individuation. These operators, or mediators, are what Serres calls quasi-objects, understood as semantically meaningful objects. This implies viewing APIs not merely as 'specifications and protocols that determine relations between software and software', but also in the sense of the quasi-object, as protocols that structure and exercise control over the specific social situations on which they are brought to bear.[12] Drawing on Roland Day's claim that quasi-objects are best understood as historical projections of power within organizational and epistemic structures, the argument is made that the kind of work that the Twitter APIs perform, needs to be situated within the platform politics of data exchange and transmission.[13] By looking at how the Twitter APIs are hinged historically and sociomaterially, this article contributes an understanding of the function of APIs as a mediatory object.

The specificity of APIs

The general concept of an application programming interface refers to standardized methods for allowing one software component to access the resources of another component. In relation to web-based APIs, the methods in question allow for programmatically accessing data and functionality via HTTP. Essentially APIs are interfaces that facilitate the controlled access to the functionality and data contained by a software service or program. But APIs, and web APIs in particular, are much more besides. The characteristics of APIs need to be understood as encompassing a variety of social and technical aspects that are 'caught up in webs of discursive and materials determinants'.[14] Among other things, web APIs encompass: a physicality in terms of the corporeal landscape of infrastructure and technology, through to the economic logics at work (i.e. business models, ownership, licencing of the APIs), functions and services (i.e. access to data), practices of users (i.e. forms of labor, play and collaboration), discursive formations (i.e. statements, knowledge, ideas), rules and norms (i.e. design principles, terms of service, technical standards), as well as social imaginaries and desires. As is apparent, APIs are complex phenomena. The question is not merely one of what APIs are, but where they come from, and what purpose they serve. What is needed, then, is to historicize APIs as part of the specific technological, economic, and socio-political constraints of the day. Let us thus take a slight detour into the history of software and the politics of 'openness', which doesn't just guide much of the material-discursive history of software development in general, but as it turns out, also plays a significant role in legitimizing APIs as the current business model of the social Web.

APIs emerged as an important software design principle to ensure *interoperability* between different systems, at a point in the history of software and computing, which coincided with the commercialization of software and rise of the personal computer during the 1980s. As software systems grew, modularity in design became the dominant principle for managing the complexity in systems design. IBM introduced the

first modular system, the System/360, in 1964. In general, modularity refers to the principle of breaking up a product into subsystems or modules, which can be recombined.[15] Modularity makes it possible to separate concerns, through the principle of 'information hiding'. First described by Parnas in 1972, this principle holds that software components need to hide implementation details from other components in order to manage complexity and to make it easier to maintain the system.[16] As Galloway points out, information hiding helps 'reduce the 'cognitive load' on the programmer by minimizing the amount of information required to understand any given portion of the system'.[17]

Indeed, hiding information is not just technically necessary, but also desirable. APIs are important instantiations of this principle, as they separate 'modules into public and private parts, so changes to the private part can be performed without impacting the public (the API itself) part, and therefore minimizing the dependencies between these two parts'.[18] We can see this principle at work in terms of how web APIs function. Basically, web APIs 'provide information to third-party applications through 'calls', a technique of retrieving data on a server in the background, without disrupting the display and function of a web page'.[19] These calls however, are usually limited, in order to prevent full access data and to keep API management under control. Situated in between codes belonging to interoperable systems, APIs ensure that changes in the underlying code will not affect the code written to interact with the core system. As such, APIs signify contracts of sorts, promises of stability and deliverance.

In the context of web services, what APIs promise to deliver, is above all data. While the past few decades have seen many ideological battles fought over the accessibility and relative openness of source code, today what matters most is access to the database. Among the first companies to offer a peek into their database was eBay in 2000, followed by Amazon with the launch of Amazon Web Services in 2002. The idea was quite simple and was reminiscent of long-standing efforts within free and open-source software development to outsource software development efforts to a potentially indefinite number of programmers. As Robert Bodle writes, 'far from a risky business strategy, opening APIs was considered a sustainable business move to encourage the growth of a supportive ecosystem of third party developers, which could increase the value of a platform or web service'.[20] The rhetoric surrounding the alleged (re)turn towards openness, emphasized the innovative potential of letting others find cleverer ways of using the data than the companies owning them were able to. For example, as Google said when they launched their Maps API in 2005: 'If you like Google Maps, but think you could do something better, now's your chance'.[21]

Today, most social media companies offer APIs, so that third-party developers can build new applications on top of their platform.[22] Not only do APIs offer a way for third-party developers to access parts of the data, but APIs have also become a useful way for these companies to extend their reach and growth across the Web. When considering the rhetorical moves deployed by most social media companies launching an API, it quickly becomes apparent that the notion of 'openness' plays an important role. However, as the reader well versed in the history of software and computing, knows, the signifier of openness is not unique to companies like Google or Twitter.

As much as APIs prevent access, they also enable access to other parts of the software, hiding, as much as they are revealing. The example of IBM illustrates, how software companies adopt different kinds of openness, for different reasons, at different points in time. More than anything, an APIs 'openness' is relational, meaning that it needs to be considered as a negotiation between the various actors involved.[23] Situated between platform providers and users, constant negotiations are forged over the degree and nature of access and usability of the data regulated by the API. Because of their nature as highly-controlled gateways to data, web APIs constitute important sites for the study of power relations. More specifically, by looking at the Twitter APIs as a case in point, this article aims to show how APIs are not simply intermediaries between API providers and API users. Rather, as mediatory objects, APIs help transform and shape any one side of this pair.

The Twitter APIs

As with all sociotechnical systems, APIs need to be understood as historical projections of power within specific organizational and epistemic structures. This article is concerned with the specific case of the Twitter APIs and its developer community. Twitter was launched in July 2006 as a micro-blogging service that prompted users to disclose their thoughts and activities in real time. Only two months after its launch, Twitter published an API and made it publicly available. This 'turn towards openness', of letting anybody with enough programming skills built new software using APIs, has in retrospect been heralded as one of the decisive factors for ensuring Twitter's growth and success. As Biz Stone, one of Twitter's three

founders, declared already one year after Twitter launched:

> The API has been arguably the most important, or maybe even inarguably, the most important thing we've done with Twitter. It has allowed us, first of all, to keep the service very simple and create a simple API so that developers can build on top of our infrastructure and come up with ideas that are way better than our ideas.[24]

As we can see, the philosophy of letting others come up with innovative ideas popularized by companies like Amazon, eBay, and Google, also became an important strategy for Twitter. Twitter's sparse beginnings as a web service quickly grew into an ecology of third-party applications. As ProgrammableWeb, a website specializing in mapping different web APIs, recapitulates:

> Twitter's growth can be attributed to the Twitter API, which allowed the company to be on every mobile platform before it had an internal team building mobile apps […] Even Twitter's search engine was built on Twitter's API. Originally called Summize, it was Twitter's first acquisition way back in 2008. The product, which was better than anything built internally at Twitter, is also available via the Twitter Search API.[25]

Almost from the very beginning, API usage generated more traffic than end-users did. In 2007, Twitter's API traffic amounted to ten times that of Twitter's main site, which had reportedly increased to twenty times the main site by 2009.[26] In 2010, ProgrammableWeb reported that a staggering 75% of Twitter's traffic came from third-party applications.[27]

Little by little and steadily over time, Twitter has acquired the most popular third-party apps and clients that developers were making, often integrating these apps into the Twitter core service. For example, as indicated by the above quote, instead of developing its own search engine, Twitter bought *Summize*, the most popular third-party search tool on the market at that time. Subsequently Twitter incorporated the search tool into its service while continuing to offer Summize's original search APIs, and even employed all five Summize software engineers as part of the Twitter team. Similar patterns of acquisition have followed suit, a strategy that now constitutes an important part of how Twitter develops its platform, while also making sure to fight off their strongest competition.

Twitter offers two different APIs, both based on the HTTP standard: the REST, and the streaming API. The REST API provides a way for developers to access and operate Twitter's core functionalities, using a list of standardized methods, including: to post tweets, access a user's followers, search for recent tweets (restricted to tweets within the last week), sending and retrieving messages.[28] The streaming API, on the other hand, pushes data to selected partners in near real-time. This does not mean that all the data stored by Twitter is freely available to anyone who is technically literate enough to make a few HTTP requests. The API documentation details what limitations apply for every single endpoint offered. Below, I outline the empirical study conducted with Twitter third-party developers, on their practices of working with APIs and Twitter APIs in particular.

The study

This study draws on data collected from a variety of different sources, including online interviews, forum discussions and publicly available documents about the Twitter APIs. Between August 2010 and July 2011, I interviewed twenty Twitter third-party developers in order to gain deeper insight into the ways in which software, in this case APIs, do work in the everyday life of developers. The interviews probed developers' uses and perception of the Twitter APIs, as well as their personal background and interest in programming more generally. Questions were also asked about their relationship to, and interactions with the various actors in the Twitter ecosystem, including the broader community of third-party developers, Twitter Inc., and end-users.

All the informants were recruited using the 'Twitter Development Talk' mailing list hosted by Google groups, which at the time of conducting this research, was the main online forum and support group for users of the Twitter API.[29] As many of the group members had had their contact details on display, a sample of addresses chosen at random was compiled, and invitations to participate in my study were sent out. The final sample consisted of two women and eighteen men, spanning an age range of forty years. In addition, the developers came from very different educational and professional backgrounds (i.e. high school students, a PhD in biochemistry, an American visitor to CERN), and their background in computer programming varied from hobby developers through to expert programmers.

Although at least double the number of people responded to my initial email invitation, the final sample of twenty respondents refers to those developers who replied to my questions in a more or less elaborate manner, or with whom I subsequently entered into several rounds of discussions, sometimes spanning several months of email exchanges. Some of the developers had only posted once on the mailing list when I contacted them to ask for their participation, while others had been quite active members of the Google group, sometimes posting up to several times a day.

All the interviews were carried out using email, as this turned out to be the medium of choice for most of my informants. The data collection was carried it out in two stages, with the initial round of interviews conducted during August and September 2010, and the second during May and June 2011. All names and identifying information have been changed to protect the privacy of the informants. The data was analyzed using an iterative coding strategy, whereby the transcripts (emails) were coded to look for common themes across the corpus.
In addition to the interviews, this study draws on entries from the discussions hosted on 'Twitter Development Talk', as well as information from publicly available documents pertaining to the Twitter APIs (i.e. Twitter terms-of-service agreements, the developers rules of the road for using the Twitter APIs, and industry blogs).

How developers view and understand APIs
In many ways, the developers who participated in this study agree that the impact of APIs can hardly be overstated. Not only has the introduction of APIs proved detrimental to the success and growth of platforms like Twitter, but they have also allowed for an entire new generation of web developers to emerge, as one of the developers pointed out. We could say that APIs have opened up the playing field of software development, in the sense that their presence has allowed for a much broader range of actors to use and repurposing the data and functionality of an existing service. Jack, the CEO of a major social media data reselling company, described how APIs have 'yielded the next wave of Internet related innovation':

> There's been a huge wave of "just open it up and see what happens" that we're just at the beginning of understanding. The implications of which will shudder some businesses, while allowing some to flourish. What the underlying API supports, or doesn't, indeed is defining social media as we know it. APIs define what we can build, policy-wise, as well as technically, and subsequently the products we build/use/consume, which in turn obviously affect culture and socialization in general.

Perhaps more than anything else, Jack's comment illustrates how APIs not merely have the power to regulate access to the content of databases and software functionalities, as they currently exist. Rather, APIs are also future-oriented. The kind of 'openness' invoked here, is not one of access or 'free', but of anticipation. 'Just open it up and see what happens' I would argue, signifies a certain kind of openness towards the future, where APIs are essentially deployed to ask developers to reimagine existing services and to transform them into new realities.[30] While APIs have indeed lowered the barrier to entry, third-party developers are not merely engaged in producing their own applications. Rather, they participate in producing the underlying Twitter service itself, as their work is systematically fed back into the underlying system. 'Twitter would have taken off without an API', says Carl, as 'it allows third party developers to do things with Twitter and tweets that Twitter as a company did not initially think of'. Nick, a biologist turned system admin, concurs in describing the alleged importance of the third-party ecology to the making of the Twitter platform: 'Third party developers were really pretty essential to Twitter's success'. Tony explains the business logic:

> Software companies have to develop a very small part of the software and they get millions of developers for free (almost) all around the world, creating new improvements to the original software. The companies can even incorporate those modifications into the software if they become important enough, and the cycle keeps going round! Basically they are taking, what is probably their biggest direct cost, programmers, and dilute that cost amongst a huge base of programmers that not only code for less but also provide the largest source of new ideas for their software. They are making sure they stay in the retail software game as it evolves.

What stands out here, is the ways in which APIs represent governmental technique for controlling innovation. More than just lines of code or business logic, APIs becomes a social enterprise. Moving in out of the computer, into mailing lists and discussion forums, APIs, just like any digital artifact, needs to be

understood as 'discursively constructed through the collective activities of relevant communities'.[31] For the Twitter third-party developers I interviewed, the Twitter Google Group concentrates much of this discursive and collective activity. 'Every developer community usually gathers in one or more places to interchange knowledge', says Jeremy, pointing out that this 'interchange is essential to the progress of programming'.

Lifeworld of programmers

On closer examination, APIs appear to become meaningful to different actors, in different contexts, and in different ways. How, and in what ways, APIs figure as part of the lived experiences of developers is historically hinged. Developers are not a uniform lot, but come at programming and web development from a variety of backgrounds and motivations.

The Google group discussion thread *Introduce yourself*, gives a unique glimpse into the norms of the community and developers' motivations for using the APIs.[32] Counting 99 individual replies, many of the developers make a distinction between their professional and personal lives. Utterances like 'Engineer by day, Twitter hacker by night', or 'Making apps for fun after work' is common parlance amongst the discussants on the Twitter development forum. Because many of the developers seem to have other day jobs, often completely unrelated to using the Twitter APIs, they are left to tinker with the APIs in their spare time. While one discussant says he thinks the APIs 'are a super fun way to get into coding', others articulate the Californian ideology[33] by hoping to make the 'next killer app', and to finally 'make a living' from their love of programming.

Despite their differences, most of the developers I interviewed considered themselves as having extensive programming skills. In addition the majority of developers stressed the fact that they were completely self-taught, and had often started programming from an early age: 'It started when I was very young, like 10 years old. I started programming these Lego robots to pick up small objects from the floor and then it just emerged from there', says Daniel, who at the time of our interview attended high school in Denmark. Alex, a very active list member, concurs in describing his passion for programming:

> From an early age I was interested in computing and the way things worked. Computing just seemed like the next step to me. I started programming with BASIC on an old C64, then moved on to programming macros and VB programs at school. By the time I hit college I was already fluent in a couple of programming languages and started to get into web-based technologies. From there I learned about HTML, CSS, Javascript, PHP and the likes.

These discourses of programming as something vocational, evolving from a youthful desire to play with technology, do not operate as a functional whole. Indeed, the love for programming, widely expressed in both the interviews and the developer forums, often seems to go hand in hand with an 'entrepreneurial mindset' – a system of belief that values innovation and business opportunity.

Enterprise culture

The notion of 'entrepreneurism' stands at the centre of what cultural analyst Paul du Gay has described as 'enterprise culture'.[34] Emerging from certain managerial discourses during the 1990s, enterprise culture as du Gay sees it, promotes values of self-reliance, personal responsibility (for instance for acquiring the right skills, or for one's own failures and successes, etc.), as well as the desire for personal success and risk-taking. Indeed, while many third-party developers use their spare time to tinker with the Twitter APIs for fun, the emotional investments connected to these forms of 'work as play'[35], becomes a key resource in an economy geared towards perpetual innovation. By reinforcing concepts of pleasure and desire in discourses surrounding the Twitter API, pleasure is turned into a 'disciplinary technology'.[36] As Alice Marwick has shown, in her ethnographic account of startups in the San Francisco area, 'the tech scene is rife with a powerful mythology of entrepreneurship which places a high value on innovation and competiveness, and frames technology work as a meritocracy where the smartest and hardest-working people should be rewarded with immense wealth'.[37] Echoing this powerful myth of entrepreneurship, Brad who has worked for one of the most successful third-party apps in the Twitter ecology recounts his success:

> I've been working with PHP for 6 years now, but I am completely self-taught. Most people don't realize it, but all you have to do is pick up a programming book, start reading it, then dive in and get your hands dirty writing some code. I guarantee you it is the best and fastest way to learn any language. That's what I did, and now I'm working for one of the top 100 sites on the

> Internet. The “Web 2.0” era has really proved that anyone with determination and patience can make it big, as long as you have the drive and the creativity.

In many ways, Brad reinforces the values of self-reliance and personal responsibility characteristic of enterprise culture in his account, suggesting that all that is needed to make it big is hard work and determination. Programming is about ‘getting one’s hands dirty’ as Brad says, about the tedious and repetitive work that is required at times in order to become a good programmer. While the developers I talked to, for the most part seemed to agree that the Twitter APIs were ‘simple’ to learn and use, they also spoke about the patience and resilience required in getting the app right, ‘the best way to learn and make it mine is to be patient and discover how every bit of code works’, as Eric put it.

The time it takes to learn and acquire all the necessary skills, however, is more often than not, a luxury not everybody can afford. In the fast paced and ever-changing world of technology, being a programmer requires an ongoing effort to keep up with the latest developments. Just as software requires care, maintenance, updates, revision, and work, so do the skills involved. Tony describes his ‘career’ in programming as a bumpy ride requiring constant care and refinement in order not to ‘loose track of it all’. After leaving programming for a few years to get a MBA, says Tony:

> When I got a job again and got around to dabbling in asp, asp.net was king of asp and it was too complicated. PHP seemed like a good alternative but I got lazy to start all over again. I kinda gave up on the whole programming thing and then iPhone came along and tickled my curiosity to get into actual native language programming, which was always there in the back of my mind. Bought into the program, got a Mac and started reading up on it all. I believe APIs are a great thing because they have let people with ideas, come one step closer to being able to put those ideas to work by not having to know too much about programming.

While the success of any good working API lies in its stability, APIs too, are prone to change. The changing nature of APIs lie not so much in the technicalities of the function calls per se, but in terms of the constantly updated terms of service and developers rules of the road.

Risky territory

Web APIs do not just represent a welcomed opportunity for developers to reimagine existing services. Rather, for many of the people I interviewed, the Twitter APIs are perceived as risky territory. In recent years, Twitter has enforced increased restrictions and contractual limitations on the kinds of content that developers are able to access, in many cases designing explicit guidelines on how that data can in fact be used. Indeed, these enforcements have had the power to ‘shudder some businesses, while allowing some to flourish’, as Jack pointed out in his description of APIs in the opening quote. Jacob tells me:

> ‘I’m no longer interested in contributing anything to Twitter’s API. Their hostile stance toward developers like me has been very discouraging, not to mention costly – they killed my business; it has cost me many thousands of dollars.

At the time of conducting the last round of interviews in 2011, one particular controversy around contractual limitations stood out. On March 11, 2011, platform manager Ryan Sarver posted a message on the Twitter Developer Talk list, urging developers to stop making new Twitter clients. Sarver’s message: too many developers were confusing the user experience of Twitter, by simply producing replicas of Twitter itself .[38] The solution: Setting up guidelines and rules for the design of new services, including the ways in which tweets were to be presented in a consistent way across all third-party applications. Developers immediately expressed a huge distain towards what they essentially felt as a hypocritical move from Twitter. Once utterly dependent on developer efforts, Twitter was now perceived to obstruct the very same people that had helped build Twitter in the first place. In the discussion thread following Sarver’s announcement, one developer sarcastically remarked: ‘Wow. Thanks for getting so many people interested in Twitter. Now get lost. This is appalling’. Another forum discussant provided an apt summary of what seemed to be at stake for the developer community: ‘All third party Twitter developers, no matter what they make, are now walking on eggshells, constantly at risk of offending Twitter’s ideas of how users should interact with Twitter’. Yet another developer’s reaction emphasized the potential risk that API changes have for software developers: ‘You’ve just scared the bejesus out of me because I don’t know if I’m suddenly verboten or not. Five months of work shot to hell?’[39] Of the people I interviewed, many expressed a similar sense of betrayal at the face of Twitter’s efforts to impinge on their programming practices. Says Jacob, who had been working on a Twitter client for the past six month at the time of Sarver’s announcement:

> They have changed the API to make it more difficult to build a 3rd party client, they've directly asked us to stop building clients (see Ryan Sarver's talks over the past six months), excluded us from using new API (the new iOS API is great, but cannot access DMs, so is useless to build a client app). The one thing they haven't done is just cut us off. To be honest, I'm really not sure why.

What upsets Jacob the most, is the uneven treatment he sees in the ways in which Twitter regulate their APIs, as the changes they inflict only ever seem to have the most severe consequences for third-party developers. For example, says Jacob, Twitter's in-house clients are allowed to use a far greater range of functionalities than do third-party applications, often features and functions that are simpler to use for the end-users. He emphasizes an ongoing battle for authentication requirements that may have huge impacts for the developers when changed. It's a double hit, says Jacob, 'because not only does the engineering cost a huge amount, but Twitter steals a ton of your users who are frustrated by the process: higher costs, fewer customers. Ouch'. Whether this a calculated business move by Twitter or not, the APIs function as important regulatory instruments that, indeed, define what can be built, policy-wise, as well as technically, and subsequently the products that are allowed to exist.

Rethinking APIs as quasi-objects

An API is never a neutral tool. Rather, an API is the result of complex sociomaterial negotiations and practices that embodies specific values. While APIs do nothing by themselves, they are not objective in any way. How then, can we make sense of the kind of agency imbued by APIs? In what follows, I want to offer an understanding of the power of APIs by drawing on Michael Serres' concept of the 'quasi-object' as a particularly useful analytical device. By thinking of the Twitter APIs in terms of a 'quasi-object', we can begin to trace the instabilities and negotiations between the various actors involved. For Serres, the notion of the 'quasi-object' designates a way to account for the construction of collective relations and specific situations of social life, without having to have recourse to social bonds for an explanation of sociality.[40] Instead, Serres seeks a materialist account of sociality, by arguing for the co-constitutive relation between objects and subjects (or rather quasi-objects and quasi-subjects). The quasi-object on this account, organizes the collective.

Serres uses the example of the ball game, to illustrate the productive forces of the ball – as a quasi-object. On the one hand, the ball acts as a catalyst for the game; without it there would be no game. On the other hand, the ball alone does not make the game. Rather, as Roland Day argues, quasi-objects are both conditioned by and condition a 'shared landscape of meaning'.[41] The landscape for meaning in a game, says Day, includes the rules and goals of the game, the institutional and economic forces acting upon the players, even the physical grounds upon which the game is played.[42] The ball, when passed between the players, is what established their relations and positions in a process of reciprocal redefinition. As Serres suggests: 'Around the ball, the team fluctuates quick as a flame, around it, through it, it keeps a nucleus of organization'.[43] The quasi-object, then, brings the players together in constantly shifting configurations. However, 'quasi-objects are not "quasi" simply because they, as objects, function across ontological types or series (institutions, organic bodies, machines, and so on)', Day holds.[44] Rather, they are '"quasi" because they are representations of social desires that utilize objects in order to bring about goals of social organization'.[45] Insofar, as the object cannot be separated from its social function, the quasi-object is less a thing Serres argues, and more like a contract that holds relations together and helps structure new realities.[46] The Twitter APIs, for their part, are not just something like contracts; they are almost entirely contractual, and almost nothing like a thing. Just like any other protocol, APIs are conduits for governance. Following Galloway's claim that protocol 'outline the playing field for what can happen, and where', the question becomes how we can begin to understand the playing field outlined by the Twitter APIs?[47]

APIs are not simple intermediaries, or entities that only transfer information. Like Serres' ball, the Twitter APIs assume the position of a mediatory object – entities that 'transform, translate, distort and modify meaning'.[48] As a mediatory object, the APIs help modify and structure both Twitter and the API users. Given that not all aspects of the object 'matter' at any given point in time, the question is how specific features of the API become significant for the programmers and the work that they are involved in?[49] As we have seen from Jacob's case described above, it is not the entirety of the API that matters for him, but rather certain aspects given his specific situation. Different features, matter for different actors, at different times. While changes to the authentication systems may not matter for someone using the API to display tweets on his or her personal website, it has huge consequences for someone like Jacob, whose whole

business depends on the ability to access the direct message feature.

While not meant as a functional analogue to APIs, I believe Serres' explication of the quasi-object in his example of the ball game serves as a helpful conceptual device for an analysis of the enactive powers of APIs. If we follow this line of reasoning, the Twitter APIs help stabilize some relations while destabilizing others. Indeed, as Jack suggested in the opening quote, what the underlying API does or does not support, defines social media as we know it. Who is being affected, and in what ways, depends on the positions of the actors involved. According to Steven Brown, 'the relationship between the players is defined by how they position themselves with regard to ball'.[50] In other words, some players stand a better chance of winning, while others are at risk of losing. How these positions are established in the first place depends on how well subjects – or rather quasi-subject, since they change their own ontological status as they enter into relation with the quasi-object – subsume to the logic of the quasi-object.[51]

Most obvious is perhaps the absolute need to conform to the categories presented as to what counts as *proper* innovation, at any given time. Every developer who wants to use the API to access and repurpose the data needs to conform to a set of rules. The rules essentially describe 'what type of innovation is permitted with the content and information shared on Twitter'.[52] Moreover it says that 'Twitter may update or modify the Twitter API, Rules, and other terms and conditions, including the Display Guidelines, from time to time'. This, as we have seen, constitutes a risky territory. Not only do these rules grant Twitter the power to shut down any application that does not comply, but because of the changing nature of the contract, developers are asked to continuously stay alert. As the social media blog *Mashable* already alluded in 2009, 'creators of third party applications are always at risk that their efforts will simply be erased by some unpredictable move on the part of the company that controls the API', asking whether it simply 'has become too dangerous to build an application on an API you can't control?[53] Just as API providers expect developers to adhere to the rules of the road and the terms of service, developers expect the APIs to remain stable. Broken contracts, as in changed APIs, often imply significant extra cost for third-party programmers. In some cases, changed APIs means complete waste of time, in worst case the loss of a business, of ones livelihood. As with ball game, there is always the risk of being tackled. Says Serres, 'with the ball, we are all possible victims; we all expose ourselves to this danger and we escape it'.[54]

As the case of Twitter prohibiting certain kinds of software development discussed earlier shows, being tackled by the quasi-object might be more real than not. From Twitter's perspective the playing field appeared to have gotten out of hand; the rules of the game were not as apparent anymore, and too many players had begun playing the game on their own terms. The rules needed to be straightened out, and it needed to be made clear who had the power to define the rules and conventions of the playing field. Of course, my intention is not to assign blame to any one party for something that arguably is 'part of the game'. That is, third-party developers are often well aware of the risks involved in making their applications reliant on the Twitter platform.

Incidents and controversies like the Ryan Sarver message, further suggest a need for the quasi-subject to conform to the logic of cognitive capitalism. Consistent with enterprise culture, third-party developers have to assume flexible subject positions, and they have to be willing to take on the risk that this kind of work entails. For someone like Jacob, the implications might simply mean losing the game, because of the ways in which the quasi-object moves in unidentified ways. While APIs have become the basic building block of the social Web, cases like the Twitter third-party client prohibition, show that 'sometimes applications might be building their services on a foundation of sand'.[55]

Here, it is important to not see quasi-objects as detached from the societal contexts in which they are imbued. Rather, following Day in his emphasis on the historical dimensions of the quasi-object, APIs need to be situated in terms of how they 'relate to existing social structures and in how they embody and anticipate the future through the socio-material practices that they allow or disallow'.[56] Thus, we need to acknowledge that the entrepreneurial self is not just produced through the object, but is part of a much larger performative incitement to meritocracy and individualism characteristic of neoliberal discourse.

Not only does the function of APIs depend on how well users conform to its logics and rules in the present, APIs are also fundamentally geared towards the future. The traces that quasi-objects make, Day argues, are 'not simply of its past or even of its present' but of the ways in which they 'organize and model the future'.[57] What is interesting about the specific case of APIs is that the meanings and uses 'we

foresee for them in the future' is deliberately indeterminate and left open for interpretation. Paradoxically, this *potentiality* or openness towards the future, is highly controlled. APIs, as we have seen, determine what kinds of applications that can be built, both technically and policy-wise. However, demarcating a possibility space and set of potentials, APIs are exactly the promises they purport to be. Not only do APIs invite developers to reimagine their services, they are deliberately constructed around an ethos of participation. As such, web APIs constitute key techniques for governing developers into paths of desired productivity. According to Paolo Virno, it is exactly the potential to produce, which constitutes real labour-power.[58] Production in such autonomist accounts however, is not necessarily linked to making concrete objects, but aimed at harnessing and regulating the *capacity* of the field, that is, of ensuring that there exists a pool of yet to be actualized potential. A potential, that in actuality, is to be actualized in an anticipated way.

Conclusion

The aim of this exploratory article has been to draw attention to some of the shifting configurations of players, positions, referees, legislators, desires and motivation that are caught up in the webs of discursive and materials determinants of web APIs. While APIs open up interesting new possibilities for software studies and social media research, in terms of data collection, they are currently at risk of being treated as just another convenient software tool. As it stands, APIs are for the most part simply *used*, not critically scrutinized as powerful managers of contingent relations and flows of information and communicative exchange. However, the Twitter API have the power to gather multiple actors into communities of practice and discursive regimes of vested interests, revealing that APIs are never only neutral objects. Rather, the notion of the quasi-object attests to the status of API as a software object that actively produces the conditions described above. The history and practice of working with the Twitter APIs suggests that APIs are mediatory objects that have enactive powers, structuring both the Twitter platform and its users. By providing an API, the company is able to harness the capacity of the field, of letting third-party developers come up with ideas that they would not have been able to. Commonly, APIs are described as gateways to the data trove of web companies. From the perspective of platform owners however, APIs constitute gateways to a yet to be actualized pool of imagination. In this sense, the work performed by the APIs is of rhetorical nature, asking developers to engage in practices of reimagination and anticipation.

The work of anticipation however, is not just confined to the rhetorical function of APIs. As we have seen, APIs are anticipatory in their very operational logic. What their service function allows for, and the types of methods offered, certainly determines what kinds of applications that can be built, at the types of data that can be used. Moreover, as protocological objects, the power of the Twitter APIs stems from their governing capacity to define what counts as *proper* innovation. Importantly, the governmental power of the quasi-object depends on how well the quasi-subject subsumes to the to the logic of the quasi-object – a logic that is always already historically hinged. In the case of the Twitter API, developers are asked to conform to certain forms of social organization and desired models of production within the epistemic structures of cognitive capitalism. If we were to leave matters there, the analytic usefulness of Serres' notion of the quasi-object would only be partial. What is important here is the co-constitutive nature of the quasi-object. Using the notion of quasi-object as a rigorous analytic device, allows us to understand how the Twitter API is both conditioned by and condition a shared landscape of meaning. As we have seen, this landscape of meaning is sociomaterially constituted. As a consequence, the practices and functions of APIs cannot usefully be understood by considering its technical determinants only. Besides the infrastructural specificity of HTTP, the morphology of relations involved in the Twitter APIs include the specific business models underlying the APIs, the rules, norms and terms of services guiding the use of APIs, and the various desires and social imaginaries accompanying the practices of third-party developers and APIs providers. It is up to the researcher to dive into these temporarily fixed arrangements to try and understand what actors are in fact enlisted, and involved in the various forms of contestation at a specific point in time, for what purpose and for what organizational goals. Whilst contributing one such glimpse into the specific currents of Twitter's platform politics between 2010 – 2011, more broadly, this study suggests that APIs can be seen as quasi-objects of intense feeling – semantically meaningful entities that are invested with various forms of contestation and identification, desires and disappointments.[59] APIs do not merely give rise to new realities, enlisting different actors to engage in different types of work, they also regulate the playing field of what can happen where and when, what can be built technically, culturally and policy-wise. It is therefore important that we start paying more attention to APIs as powerful governing techniques of the current social Web.

Bibliography

Ammirati, Sean. “Twitter’s Open Platform Advantage.” http://www.readwriteweb.com/archives/twitter_open_platform_advantage.php.
Ananny, Mike. “Press-Public Collaboration as Infrastructure: Tracing News Organizations and Programming Publics in Application Programming Interfaces.” *American Behavioral Scientist* (2012).
Baldwin, Carliss Y. and Clark, Kim B. *Design rules: The power of modularity* Cambridge, Mass.: MIT Press, 2000.
Barbrook, Richard and Andy Cameron. “The Californian ideology.” *Science as Culture* 6, no. 1 (1996): 44-72.
Bodle, Robert. “Regimes of Sharing: Open Apis, Interoperability, and Facebook.” *Information Communication & Society* 14, no. 3 (2011): 320-37.
Brown, D. Steven. “Parasite Logic.” *Journal of Organizational Change Management* 17, no. 4 (2004): 383-95.
Callon, Michael. “Actor-network theory: The market test”. In John Law and John Hassard (Eds.) *Actor network theory and after* Oxford, UK: Blackwell, 1999.
Campbell-Kelly, Martin, and Daniel D. Garcia-Swartz. “Pragmatism, Not Ideology: Historical Perspectives on Ibm’s Adoption of Open-Source Software.” *Information Economics and Policy* 21, no. 3 (2009): 229-44.
Cramer, Florian, and Matthew Fuller. “Interface.” In *Software Studies: A Lexicon*, edited by Matthew Fuller. Cambridge, Mass.: MIT Press, 2008.
Day, Roland E. *The modern invention of information: Discourse, history, and power*. Carbondale, IL: Southern Illinois University Press, 2001.
de Souza, Cleidson R. B., and David F. Redmiles. “On the Roles of Apis in the Coordination of Collaborative Software Development.” *Computer Supported Cooperative Work-the Journal of Collaborative Computing* 18, no. 5-6 (2009): 445-75.
Du Gay, Paul. *Consumption and Identity at Work*. London: Sage, 1996.
DuVander, Adam. “Twitter Reveals: 75 % of Pure Traffic Is Via Api (3 Billion Calls Per Day).” In *ProgrammableWeb*, 2010.
———. “5000 Apis: Facebook, Google and Twitter Are Changing the Web.” In *ProgrammableWeb*, 2012.
Ekbia, Hamid, R. “Digital Artifacts as Quasi-Objects: Qualification, Mediation, and Materiality”. *Journal of the American Society for Information Science and Technology* 60, no.12 (2009): 2554–2566
Galloway, Alexander R. *Protocol: How Control Exists after Decentralization*. Cambridge, Mass.: MIT Press, 2004.
———. “Language Wants to Be Overlooked: On Software and Ideology.” *Journal of Visual Culture* 5, no. 3 (2006): 315-31.
Gehl, Robert W, and Sarah Bell. “Heterogeneous Software Engineering: Garmisch 1968, Microsoft Vista, and a Methodology for Software Studies”. *Computational Culture: A journal of software studies* 2, 2012.
Gill, Rosalind, and Andy Pratt. “In the Social Factory? Immaterial Labour, Precariousness and Cultural Work.” *Theory Culture & Society* 25, no. 7-8 (2008): 1-30.
Gillespie, Tarleton. ”The stories that tools tell”. In Caldwell, John, and Anna Everett, (Eds.) *New Media: Theses on Convergence Media and Digital Reproduction*. Routledge, 2003.
Higginbotham, Stacey. “Are Apis the New Black?”: http://gigaom.com/2011/03/16/are-apis-the-new-black.
Hultman, Johan. “Through the Protocol: Culture, Magic and GIS in the Creation of Regional Attractiveness”. *Tourism Geographies: An International Journal of Tourism Space, Place and Environment* 9, no. 3 (2007): 318-336.
Kitchin, Rob, and Martin Dodge. *Code/Space: Software and Everyday Life*. Cambridge, Mass.: MIT Press, 2011.
Latour, Bruno. *Reassembling the Social: An introduction to actor-network theory*, Oxford: Oxford University Press, 2005.
Lennon, Andrew. “A Conversation with Twitter Co-Founder Jack Dorsey.” http://www.thedailyanchor.com/2009/02/12/a-conversation-with-twitter-co-founder-jack-dorsey.
Leonardi, Paul M. “Materiality, Sociomateriality, and Socio-Technical Systems: What do these terms mean? How are they different? Do we need them?” In Paul M. Leonardi, Bonnie A. Nardi, and Jannis Kallinikos (Eds.) *Materiality and Organizing: Social Interaction in a Technological World*. Oxford: Oxford University Press, 2012.
Mackenzie, Adrian. *Cutting Code: Software and Sociality*. New York: Peter Lang, 2006.
Marwick, Alice. “Status Update: Celebrity, Publicity and Self-Branding in Web 2.0.” New York University, 2010.
Michael, Mike. “On making data social: Heterogeneity in sociological practice”. *Qualitative Research* 4, no. 1 (2004): 5-23.
Musser, John. ” Twitter Api Trafic Is 10x Twitter’s Site ” http://blog.programmableweb.com/2007/09/10/twitter-api-traffic-is-10x-twitters-site/.

Parnas, David L. “On the Criteria to Be Used in Decomposing Systems into Modules”. *Communications of the ACM* 15, no. 12 (1972): 1053-1058.
Schroeder, Stan. “Twitter Api Gets Rate Limit; Will It Hurt App Growth?”. In *Mashable*, 2009.
Serres, Michel. *The Parasite*. Baltimore: The John Hopkins University Press, 1982.
———. *Genesis*. Ann Arbor: University of Michigan Press, 1995.
Suchman, Lucy. “Organizing alignment: a case of bridge-building”. *Organization* 7, no.2 (2000): 311-27.
Virno, Paolo. *A Grammar of the Multitude: For an Analysis of Contemporary Forms of Life*. Los Angeles, Mass.: Semiotext[e], 2004.
Yoo, Youngjin. “Digital Materiality and the Emergence of an Evolutionary Science of the Artificial”. In Paul M. Leonardi, Bonnie A. Nardi, and Jannis Kallinikos (Eds.) *Materiality and Organizing: Social Interaction in a Technological World*. Oxford: Oxford University Press, 2012.

1. See Fuller, *Software Studies: A Lexicon*; Mackenzie, *Cutting Code*. ↩
2. Kitchin and Dodge, *Code/Space: Software and Everyday Life*, 247-251. ↩
3. Galloway, *Protocol*. ↩
4. Gillespie, “The Stories Digital Tools Tell”, 2. ↩
5. ibid., 3. ↩
6. Mackenzie, *Cutting Code*, 96. The variable ontology of software means that ‘questions of when and where it is social or technical, material or semiotic cannot be conclusively answered’. ↩
7. Callon, “Actor-network theory: The market test”, 186. ↩
8. Michael, “On making data social: Heterogeneity in sociological practice”, 10. ↩
9. ibid. ↩
10. Suchman, “Organizing alignment: a case of bridge-building”, 316. ↩
11. Serres, *Genesis*, 87. ↩
12. See Cramer and Fuller, “Interface”, 149. ↩
13. Day, *The modern invention of information: Discourse, history, and power*, 84. ↩
14. Gehl and Bell, “Heterogeneous Software Engineering: Garmisch 1968, Microsoft Vista, and a Methodology for Software Studies”. ↩
15. See for instance Baldwin and Clark, *Design rules: The power of modularity*. ↩
16. Parnas, “On the criteria to be used in decomposing systems into modules”. ↩
17. Galloway, “Language wants to be overlooked”. ↩
18. De Souza and Redmiles, “On the roles of APIs in the coordination of collaborative software development”, 449. ↩
19. Bodle, “Regimes of Sharing”, 322. ↩
20. ibid., 325. ↩
21. Taylor, “The world is your JavaScript-enabled oyster”. ↩
22. For instance, the photo-sharing platform Flickr has one API, Youtube has two, and Facebook is currently in the process of centralizing all API activity through their Graph API. ↩
23. Ananny, “Press-Public Collaboration as Infrastructure: Tracing News Organizations and Programming Publics in Application Programming Interfaces”, 2. ↩
24. Ammirati, “Twitter’s open platform advantage. ↩
25. DuVander, “5000 APIs: Facebook, Google and Twitter are changing the Web”. ↩
26. Musser, “Twitter API trafic is 10x Twitter’s site”; Lennon, “A conversation with Twitter co-founder Jack Dorsey”. ↩
27. DuVander, “Twitter reveals: 75 % of pure traffic is via API (3 billion calls per day)”. ↩
28. Technically, web-based APIs usually conform to one of two consistent architectural frameworks for implementation, REST or SOAP. Simple Object Access Protocol (SOAP), developed by Microsoft in 1998, is one of the most widely-used frameworks for building web services. While SOAP is a W3C-recommended web architecture standard, Representational State Transfer (REST) is not considered a standard so much as a style for designing networked applications by using simple HTTP. REST has to a large degree replaced SOAP as a standard for communication with other Web services over networks, which can mostly be explained by its simpler and more implementation-friendly nature. REST accepts many different message formats in contrast to SOAP, which is entirely bound up to XML as a standard message format. According to ProgrammableWeb, as of February 22th, 2012, 68 % of all APIs now use the REST protocol, compared to 23 % using SOAP. See: http://www.programmableweb.com/apis ↩
29. The group was established in March 2007 to provide a place where API users could gather to discuss issues related to the Twitter APIs. The discussions on the mailinglist mainly focused on technical issues, or debates about Twitter’s terms of service and their developer rules. As of August 2011 the group counted 12237 members, with an average of almost 700 messages posted per

month during the first half of 2011. At most, the group had 2240 entries a month (August 2009), and steadily became somewhat less active from October 2010 onwards. In July 2011 the Google group was closed down, superseded by Twitter's own efforts to establish a community forum at dev.twitter.com – Twitter's own developer site. This study only refers to documents from the now archived Google group. ↩
30. Yoo, "Digital Materiality and the Emergence of an Evolutionary Science of the Artificial", 144-145. ↩
31. Ekbia, "Digital Artifacts as Quasi-Objects: Qualification, Mediation, and Materiality", 2564. ↩
32. See thread called 'Introduce Yourself!': http://groups.google.com/group/twitter-development-talk/browse_thread/thread/d6bd8f0a9b242717 (last accessed February 22, 2012). ↩
33. Barbrook and Cameron, "The Californian ideology. ↩
34. Du Gay, *Consumption and identity at work*, 56. ↩
35. Gill and Pratt, "In the Social Factory?", 15. ↩
36. ibid. ↩
37. Marwick, *Status Update: Celebrity, Publicity, and Self-Branding in Web 2.0*,6. ↩
38. Ryan Sarver's post was titled 'Consistency and ecosystem opportunities' and was posted March 11, 2011. The message quickly turned into a discussion thread where it received 92 replies within the following three weeks. See: https://groups.google.com/forum/?fromgroups=#!topic/twitter-development-talk/yCzVnHqHlWo ↩
39. ibid. ↩
40. Serres, *Parasite*. ↩
41. Day, *The modern invention of information: Discourse, history, and power*, 81. ↩
42. ibid. ↩
43. Serres, Genesis, 87. ↩
44. Day, *The modern invention of information: Discourse, history, and power*, 75. ↩
45. ibid. ↩
46. Serres, Genesis, 88. ↩
47. Galloway, Protocol,167. ↩
48. Latour, *Reassembling the Social*, 39. ↩
49. For instance, as Leonardi exemplifies, the rubber coating on the handle of a hammer may not matter in one's ability to hammer a nail, but it may suddenly matter a great deal if one's hands are wet. See, page 30. ↩
50. Brown, "The Parasitic Logic", 394. ↩
51. Hultman, "Through the Protocol: Culture, Magic and GIS in the Creation of Regional Attractiveness", 327. ↩
52. See Developers Rules of the Road: https://dev.twitter.com/terms/api-terms ↩
53. Schroeder, "Twitter API Gets Rate Limit; Will It Hurt App Growth?" ↩
54. Serres, *Parasite*, 227. ↩
55. Higginbotham, "Are APIs the new black?" ↩
56. Ekbia, "Digital Artifacts as Quasi-Objects: Qualification, Mediation, and Materiality", 2656. ↩
57. Day, *The modern invention of information: Discourse, history, and power*, 75. ↩
58. Virno, *A grammar of the multitude: for an analysis of contemporary forms of life*,81. ↩
59. I borrow the expression 'objects of intense feeling' from Adrian Mackenzie, *Cutting Code*, 71. Writing about Linux, Mackenzie asks how the operating system became an object of intense feeling for so many of the developers involved. ↩

Series Navigation

<< The Algorithmization of the HyperlinkOn Creativity and Calculation: Attempts at and Rejections of Formal Definitions of Creativity >>