What is an important event? 175 million events and counting

Abstract

Even a moderate sized media platforms like Github.com, a 'social coding' site widely used by software developers, record millions of events. Such sites, with their millions of projects, participants, groups and organisations, can be understood as expressions of contemporary social life. But given such vast and complex forms of expression, how can social researchers get a feeling for what counts as an important event? Even if they have open access to the data, infrastructural, configurational, analytical, philosophical and financial problems make it hard to decide what counts as an important event amidst massive streaming data. Against the drive for immediacy and the allure of totality that characterises many engagements with data streams, the paper will describe our attempts to realise in numbers, graphics, code and language a feeling for important events on Github.

When coding becomes more social: Github as cultural form, flow and practice

GitHub.com is one relatively small social media platform. It is basically a code repository for software developers and programmers. The company, like so many other Web 2.0 startups, is based in San Francisco. It started in 2008, ostensibly has no managers, and has around 300 employees, many of whom work remotely (> 65%). In 2012, it attracted $US100 million in venture capital (Hardy, 2012), a significant investment for a relatively small company. In many ways, Github exemplifies contemporanry open source ethos and business model. It is a social media platform, it is massively open source and the Github ethods of no-managers only teams exemplifies the agile forms of startup busineess found in their thousands in New York, London, Berlin, Shanghai, Melbourne, and probably Copenhagen (see TechCrunch for a listing).

Obviously programmers have nearly always swapped and shared source code. They also almost always borrow, cite, invoke or otherwise draw on existing bodies of code, even if it is only in the form of copying patterns or architectures. Writing code is nearly always closely tied up with reading code. Despite its aura of monolithic algorithmic force, code is highly *intertextual*, and this intertextuality or intersecting of inscriptions is both powerful and problematic. Many of the problems around code concern unruly forms of change and sharing. The ease of changing code creates problems: it breaks software systems. Consequently, developers take great efforts to rectify writing and reading practices around code. Revision control systems or version control systems, programming methodologies such as object oriented languages or extreme programming, or disciplines such as software engineering, are just some of the ways in which the seething textuality of code has been channelled and ordered. Code repositories are also one way in which this intertextuality or dependencies between pieces of code have been managed.

There are quite a few other online source code repositories: bitbucket, sourceforge, code.google, gitorious and launchpad are a few of them, but Github is the biggest by almost any measure. So, it might be important to software developers and to the state of software. What difference does it make when coding shifts from something that a few people do together to a massively diverse social media platform? I don't know the answer to that question, and I think it is a hard question. Github itself has become a vast textual environment (Couldry, 2000, p. 83) in which flows of meaning, flows of practice and flows of text mingle in diverse geographies. These texts contain ideas, values, practices and concepts that define the contemporary social field. These modes of thought vary as they move across populations. While software texts themselves are not usually repositories of meaning, they are heavily concerned with the distribution of meanings, affects and practices.

There are good reasons to belief that this movement of meanings, affects and practices, as intensified and expanded in software systems, today constitutes large swathes of the power-laden

material surface of culture. Rather than individual code objects, on which much discussion of the power of software has focused, we might need to be examining these techniques of distribution more closely. This in a way akin recent calls from anthropologists such as Anna Tsing for studies of supply chains (Tsing, 2009) or cultural theorist Brett Neilson's focus on logistics as a form of power (Neilson, 2012), and especially to ask if, as the anthropologist Marilyn Strathern does, 'techniques of distribution do not just disseminate what has been created elsewhere, but have themselves a creative or productive potential' (Strathern, 2006, p. 16).

Three perspectives on Github repositories

How then would we make sense of what happens on Github itself? I'm going to sketch three perspective that might be helpful. (There are no doubt others). We might look at Github in terms of:

1. Highly valorised social processes: sharing

2. Technological modification of conditions of collective existence: recursive publics

3. How online may lend structure to the analytic objects of social research

In Github.com's own terms, the sprawling flow of texts on the platform attests to the power of 'social coding', and the *sharing* of code: 'GitHub is the best place to share code with friends, co-workers, classmates, and complete strangers. Over four million people use GitHub to build amazing things together' (https://github.com/about). Github showcases (https://github.com/showcases) this sharing and sociality. It encourages people to 'be social' (https://help.github.com/articles/be-social) by copying, watching, following and contributing code. But sharing is a complicated process, and not automatic. In a sense this paper is about the making of sharing.

Sharing has multiple meanings

I'm not going to enter debates around the digital commons and its importance here. But we should be aware first of all, as Nicholas John has recently suggested, that 'sharing' has become a new keyword in social media (John, 2013). The social has come to be heavily defined on contemporary social media platforms in terms of sharing practices. Sharing here has at least three major meanings. One concerns practices of distributing things by dividing them into parts. (This sense is cognate with words such as 'shear' and 'shard'). We might call this *dividing*. Another sense of sharing concerns the holding something in common – a commitment, a belief, an attitude or set of values – without dividing or otherwise changing it. I'm calling this *holding*. Finally, the act of sharing has come to mean a form of communication or imparting, as in 'share my feelings' or 'share my experience.' This sense of sharing is closely associated with being close or *near*. All three senses of sharing – *dividing, holding, nearing* – are closely tied together in contemporary understandings of culture or social life. Digital forms of sharing, and Github is typical of this, mix all three senses of sharing.

If sharing happens on Github at all – I'm not sure it does – then it should be visible at the level of practice. Github pivots on a single piece of software *git* and a series of seemingly trivial operations performed most basically by typing commands of form `git *do_something*'. Although they are somewhat more complicated than tweeting, the basic operations on git and github are simple. (There are many tutorials on git and Github.) A code repository is established using the command:

git init

Files are created or added to the repository:

git add some_files

Some changes are made to the files – adding new lines of text for instance to a text file and then added to the repository like so:

git commit files

The files are usually 'committed' to the repository along with a message. Prior to Github all of this might be happening a single laptop or in a small team of developers. Github adds a bit to git in order to make Git-Hub:

git add remote origin https://github.com/user/repository_name

The repository can be mirrored at github as long as you have joined Github. Any new commits can be sent to Github like this:

git push

And now anyone can copy the entire repository or just some branch of it to their machine using this line:

git clone https://github.com/user/repository_name

I am not going to say too much more about this seemingly small piece of software *git,* written by perhaps the very famous programmer Linus Torvalds, who is still the key developer on the Linux kernel project. Git embodies a set of values and propositions about coding and other practices that valorises distribution, de-centralisation, and a certain vision of the coherence of code as text. Unlike some other version control systems, or revision control systems, git does not discourage different versions of code. In some ways it encourages them, and it allows different versions to coexist without much problem. But here I want to point to the simplicity and flatness of these commands. Everything I've shown here can be done, apart from signing up to Github, can be done using these simple commands. While there are more sophisticated ways of using git and graphical user interfaces for using git, this simple command-driven approach matters a lot in git and github. It connotes a directness and transparency that matters a lot to many software developers. A fundamental tension in Github, and perhaps many others settings, arises when such direct practices of integration, merging, accumulating and distributing textual forms encounter the flows of interference and diffraction associated with many repositories coming together in one place. By now I hope it is evident that work on code texts has been de-centralised and distributed somewhat. In a public repository on Github, anyone can clone or 'pull' the repository. Not everyone can make changes to it. A stranger to repository can offer changes to a repository by cloning/forking it, and then creating a PullRequest that invites the repository owner to accept the suggested changes from the cloned repository. This process of forking-pullrequest-pull is often regarded as the epitome of the social life of code sharing on Github.

We should remember too that the constant invocation of 'sharing' by social media platforms barely covers over the ways in which sharing practices become data products sold to the advertising industry. This is the argument in (John, 2013). There is no advertising on Github. The product on sale here is not code, nor audiences as such. If anything, it is the growing promise of Github as a good way of writing and reading code that potentially generates business. Note that neither git itself or Github implies any commitment to sharing. It is a mechanism for coordinating distributed work on highly changeable textual assemblages. While distributed work might be done in the name of sharing, it might not. Nevertheless, the more the public sharing side of Github grows, the more this process of sharing becomes a common good for coders. For this reason, all repositories on Github created by non-paying members are public. You pay to not share on Github as in so many other places. Non-sharing, however, does feed on the sharing. Organisation that rely on coding work are more likely to use the collaborative, distributed practices of git and Github for their own software development in non-public settings. We have little idea of how much of Github is private: 'We don't share those numbers publicly- sorry! Hope you understand' wrote back Zach Holman of Github when I posed this question (http://www.quora.com/GitHub/How-many-private-repositories-are-there-on-GitHub).

Recursive publics

A second point of departure comes from existing work on free and open software. A huge amount of

work has been done on free and open software. I'm drawing on Chris Kelty's notion of a recursive public to a certain extent:

> A recursive public is a public that is vitally concerned with the material and practical maintenance and modification of the technical, legal, practical, and conceptual means of its own existence as a public; it is a collective independent of other forms of constituted power and is capable of speaking to existing forms of power through the production of actually existing alternatives. (Kelty, 2008, p. 3)

Kelty developed this notion as a way of engaging with certain groups such as Free Software, open access science, and today open data initiatives that concern themselves with the 'radical technological modifiability of their own terms of existence' (3). All of the software we can see on Github is free and open. Much of it is concerned with modifying its own terms of existence, although often in very mundane ways.

The core sharing component of Github are *repositories* or 'repos.' At the moment (March 2014), Github itself says there are over 11 million repositories on Github (https://github.com/about) . This number needs to be treated carefully . Some of these repositories are highly significant in the world of software for cultural and commercial reasons (e.g. the Linux kernel - https://github.com/torvalds/linux or the open source firmware for Android devices, CyanogenMode, https://github.com/CyanogenMod). Others have broader significance politically (e.g. the Obama administrations 'We the People' petitioning system is at https://github.com/WhiteHouse;) or in terms of contemporary media politics (e.g. Django and its connections to the growth of data journalism at *The Washington Post* https://github.com/django/django). Many have importance to particular web platforms or media (for instance, core components of Mozilla Foundation's Firefox web browser code https://github.com/mozilla/gecko-projects). Undoubtedly, many commercially important software projects rely on github and its repositories. For instance, heavily used pieces of information infrastructure are made and worked in Github (e.g. the noSQL database MongoDb https://github.com/mongodb/mongo; or the Apache Foundation's CouchDb https://github.com/apache/couchdb). Others are major repositories for scientific or engineering projects. Nearly all of the 3D printing software is hosted on Github (https://github.com/josefprusa/PrusaMendel). Important scientific and data wrangling packages such as NumPy (https://github.com/numpy/numpy) or visualization packages such as ggplot2 (https://github.com/hadley/ggplot2) can be found there. We also see the tools for configuring and managing large collections of servers or information infrastructures (e.g. https://github.com/puppetlabs/puppet). And of course, an enormous number of entertainment related projects such as Minecrafts mods (https://github.com/Bukkit/Bukkit).

Many Github repositories aren't actually software repositories. People use the repository mechanisms, and the 'source code management' systems of git to write and maintina blogs or websites rather than code repositories. That is, people use the git mechanism as a way of coordinating writing on websites or blogs. So many repositories are not about software. They are part of the blogosphere or Web. In addition, the popularity of Github as a way of storing and coordinating work by many people on shared texts has attracted other users who are not developing software: how-to guides, metadata on the Tate's art collection (https://github.com/tategallery/collection), the White House's open data policy (http://project-open-data.github.io/policy-memo), the US Pirate Party's efforts to transparently archive their own processes (http://uspirates.github.io/), the complete genome of Kenneth Reitz (https://github.com/kennethreitz/genome), collections of maps (https://github.com/djaiss/mapsicon), legal documents, recipes, books, and blogs are just some of the diversifying use-cases now found in repositories on Github. And I have not yet mentioned the huge variety of noisy, ephemeral, incomplete, incoherent, insubstantial, trivial, playful, spammy or virtual empty repositories to be found here. As a recent article in *The Atlantic* suggests, Github is increasingly of interest to non-programmers because the de-centralised, distributed and trackable collaborative processes it supports can be used for many kinds of collective documents: designs, legal documents, maps,

images, books, blogs or websites (Meyer, 2013).

Given this diverse expression in repositories of what is happening in the world, of art, science, government, business, and politics, how do we make sense of what happens on Github? On the one hand, it seems an incredibly rich resource for understanding how software and the idea of writing code together refigures experience, space, time, cities, civil society or science. We can look at the distribution of topics, the relatively popularity of code constructs, devices or programming languages. We can map and explore the geography of coding, seeing for instance how waves of work are done between and across different locations: London – New York – San Francisco – Melbourne – Hong Kong- Bangalore – Berlin – Amsterdam, to name a few prominent locations. We could examine where repositories are located in relation to institutions, governments, corporations, media platforms, civil society organisations in health, medicine, science, entertainment, transport, etc. Some of this might be hard to do, but effectively we could see Github (and other platforms like it) as a goldfish bowls, as a miniature worlds whose components illustrate or depict a larger cosmos. On the other hand, a couple of problems come up in looking into these repositories as if they were fish in a bowl.

Data traffic as analytic social objects

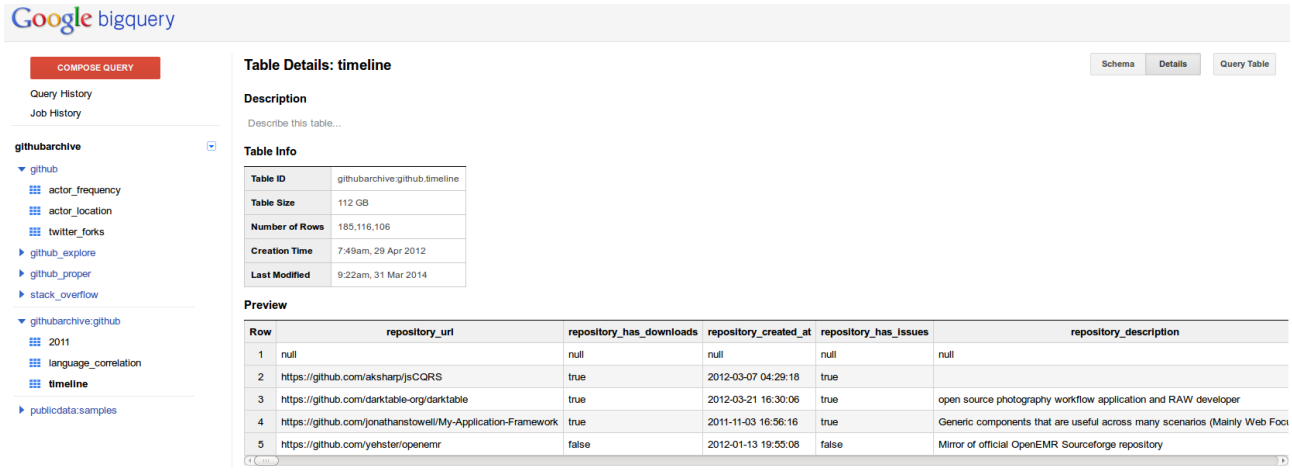
Finally, there is the question of how we know anything about what happens on these platforms. In contrast to the massive turn to data analytics as a direct line to the happening of the social, I'm following Noortje Marres and Esther Weltevrede suggestion about the need to focus on the devices through which data on platforms is produced. web scraping here:

> Scraping, we propose, makes it possible to render traffic between the object and process of social research analytically productive. It enables a form of 'real-time' social research, in which the formats and life cycles of online data may lend structure to the analytic objects and findings of social research. (Marres and Weltevrede, 2013, p. 3)

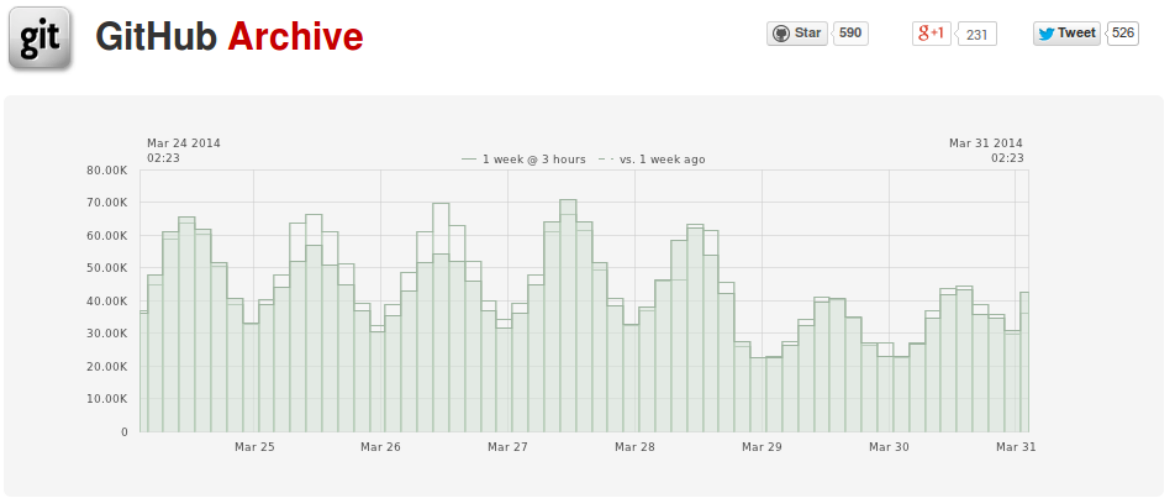
Marres and Weltevrede describe how to engage with online data as a more live or real-time social than the data itself might offer. I think this a really useful analytic addition to the much more flat data analytic approaches that we often see around social media platforms.

What is shared: events

So if something happens in relation to this shift to sharing, what is it? While Github does not share its non-sharing numbers publically, Github does distribute huge amount of data in the form of 'timeline' data and in the form of Application Programmer Interface (API) data . Like many social media platforms, these data data feeds are not meant for social scientists as such. They are meant to be used by software developers to create new services, applications or apps that will bring more people and traffic to the social platform. This data is often close to realtime, although limited in various ways (see (Uprichard, 2012) for an analysis of this). But something more is going on with the Github data. Since 2012, a massive archive of events on github has been updated hourly at both GithubArchive.org and on Google's BigQuery cloud analytics platform (https://bigquery.cloud.google.com/table/githubarchive:github.timeline).



At the time of writing (March 2014), there were around 180 million events in the GithubArchive. While the BigQuery dataset gives no description of the data other than a basic summary of how much is there, GithubArchive.org does:



> Open-source developers all over the world are working on millions of projects: writing code & documentation, fixing & submitting bugs, and so forth. GitHub Archive is a project to **record** the public GitHub timeline, **archive it**, and **make it easily accessible** for further analysis (Grigorik, 2012)

writes Ilya Grigorik, a 'Developer Advocate @ Google, working on everything web performance related: protocols, standards, browser performance' according to his Twitter feed. Much more detail on how to access and use this data can be found on Grigorik's in one of Grigorik's own Github repositories (https://github.com/igrigorik/githubarchive.org/tree/master/bigquery). For instance, the approximately 200 columns in the dataset are described in the 'schema.js' document there (https://github.com/igrigorik/githubarchive.org/blob/master/bigquery/schema.js). Each row of the dataset is one timestamped event. So, to take an almost random example, the query

'SELECT created_at, type, actor, repository_name, url FROM [githubarchive:github.timeline] LIMIT 50'

yields:

Query Results 11:19am, 31 Mar 2014

Row	created_at	type	actor	repository_name	url
1	2014-03-15 09:04:43	PushEvent	vvakame	DefinitelyTyped	https://github.com/borisyankov/DefinitelyTyped/compare/3d3832de3e...93a2313f62
2	2014-03-15 09:04:42	PushEvent	gizmomogwai	plist	https://github.com/gizmomogwai/plist/compare/12eb82d283...83ad8b5f26
3	2014-03-15 09:04:48	IssueCommentEvent	ben-lin	node.inflection	https://github.com/dreamerslab/node.inflection/issues/16#issuecomment-37721104
4	2014-03-15 09:04:48	WatchEvent	cbmd	mongofill	https://github.com/koubas/mongofill
5	2014-03-15 09:04:48	WatchEvent	svett	molokai	https://github.com/tomasr/molokai
6	2014-03-15 09:04:47	GollumEvent	swordray	wiki	https://github.com/ruby-china/wiki/wiki/RubyGems
7	2014-03-15 09:04:46	PushEvent	elfet	purephp	https://github.com/elfet/purephp/compare/dc24b70f00...c94c524e76
8	2014-03-15 09:04:46	PullRequestEvent	nikkypx	rets_data	https://github.com/arcticleo/rets_data/pull/2
9	2014-03-15 09:04:46	WatchEvent	mjaneczek	google-authenticator	https://github.com/jaredonline/google-authenticator
10	2014-03-15 09:04:53	PushEvent	micmath	Rye	https://github.com/micmath/Rye/compare/69c02b1aea...7885ea9e36
11	2014-03-15 09:04:53	PushEvent	fitret	daigon	https://github.com/fitret/daigon/compare/8897fa9404...77e5f48276
12	2014-03-15 09:04:53	WatchEvent	joeyates	spork	https://github.com/sporkrb/spork
13	2014-03-15 09:04:53	CreateEvent	grcnva	mcs-chatboard	https://github.com/grcnva/mcs-chatboard
14	2014-03-15 09:04:51	PushEvent	albatrossen	bungeebouncer	https://github.com/albatrossen/bungeebouncer/compare/57a3771590...7794ffb531
15	2014-03-15 09:04:51	PushEvent	catalinstanciu	ChessEngine-Xboard-cpp	https://github.com/catalinstanciu/ChessEngine-Xboard-cpp/compare/9bedce1744...8b98d2ff45

First < Prev Rows 1-15 of 50 Next > Last

I don't think this is a meaningful query, but we can see events occurring second by second. The events of various types – PushEvent, IssueCommentEvent, WatchEvent, PullRequestEvent, CreateEvent, GollumEvent – performed by various actors with mostly unpronounceable names feed into repositories with often quite generic names – 'wiki', 'google-authenticator', 'mcs-chatboard', 'plist', but also odd sounding repositories such as 'bungeebouncer' (prevents impersonators in chats sessions connected to Minecraft servers connected together using Bungeecord software) or 'spork' ('Tim Harper‘s implementation of test server (similar to the script/spec_server provided by rspec-rails), except rather than using the Rails constant unloading to reload your files, it forks a copy of the server each time you run your tests' http://spork.rubyforge.org/).

I want to turn to this data, and what we might make of it a bit later, but we should note here that a 'Developer Advocate' at Google makes available a huge amount of data about what developers are doing on Github using a cloud analytics platform like BigQuery. This somewhat self-referential or recursive conjunction of practice, recording and and analysis is a typical one today. Does this availability, this sharing of the data about the practices of sharing matter?

Strathern suggests that the value of revealing or sharing cultural products depends on moments of stoppage. She writes that there needs to be 'alterity to structure the moment of validation. .. the producers of creations need witnesses to confirm their value' (Strathern, 2006, p. 23). The githubarchive and google bigquery timeline data flows might be the forms of alterity that Githubbers need to confirm their value. The archive and the timeline might be forms of alterity that structure the moment of validation to put it more technically. That is, the availability of this data might be part of the way that certain kinds of optimistic promises and fantasies of a good life (a life without managers?) are potentialised and distributed. The Github data competitions that ensued in the wake of the 'sharing' of this data suggest that something of this kind might be taking place.

Sharing is not automatic

But even if this sharing is done for the purposes of promising, can we repurpose the act of promising in order to reconstruct the practices of the promise? Can we follow the growth of expectations and hopes and promises in the timeline itself?

Signing on to BigQuery.cloud.google.com

We signed on to BigQuery to access the timeline. The timeline is around 112Gb, and is, as I mentioned already, updated hourly. Our intention was to track the movement of coding practices between different software projects to gain some sense of the deep sharing in the sense of holding in common being done today. Given the sheer diversity of projects, their sprawling geography, the sheer numbers of developers, platforms and software libraries, as well as the increasingly tangled

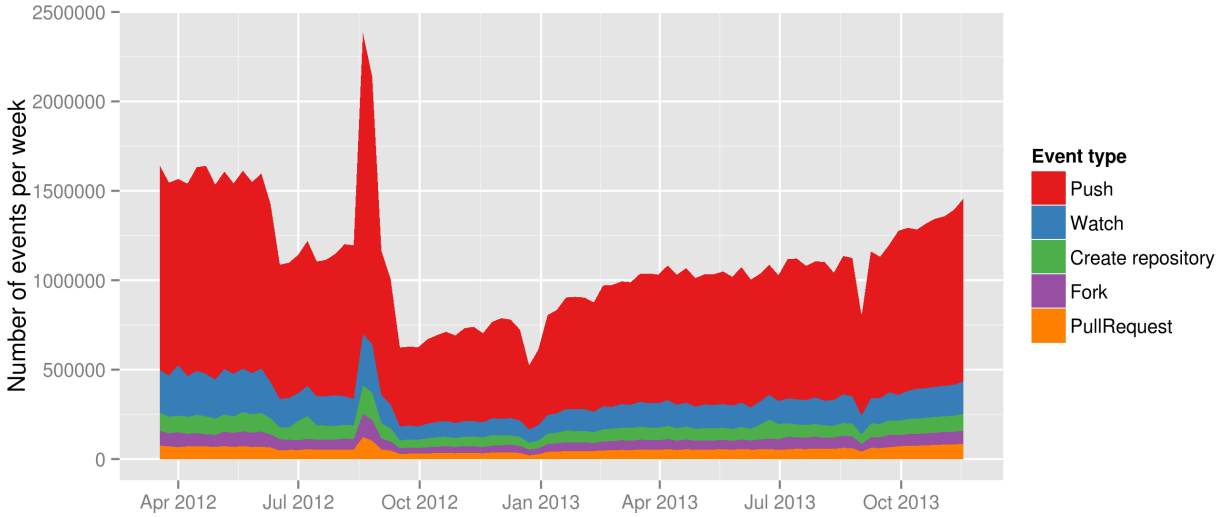
digital economies and cultures, could this dataset provide a way of doing something like what Bruno Latour has recently asked: ‘let the agents produce a dynamics and collect the traces that their actions leave as they unfold so as to produce a rich data set’ (Latour et al., 2012) He asks: ‘is it possible to do justice to … common experience by shifting from prediction and simulation to description and data mining?' The common experience he refers to here is precisely that which is shared not in the sense of divided but held in common.

We should not BigQuery is not the only way this data is shared. We could have downloaded it all from githubarchive.org and stored it in our own databases. A command like this:

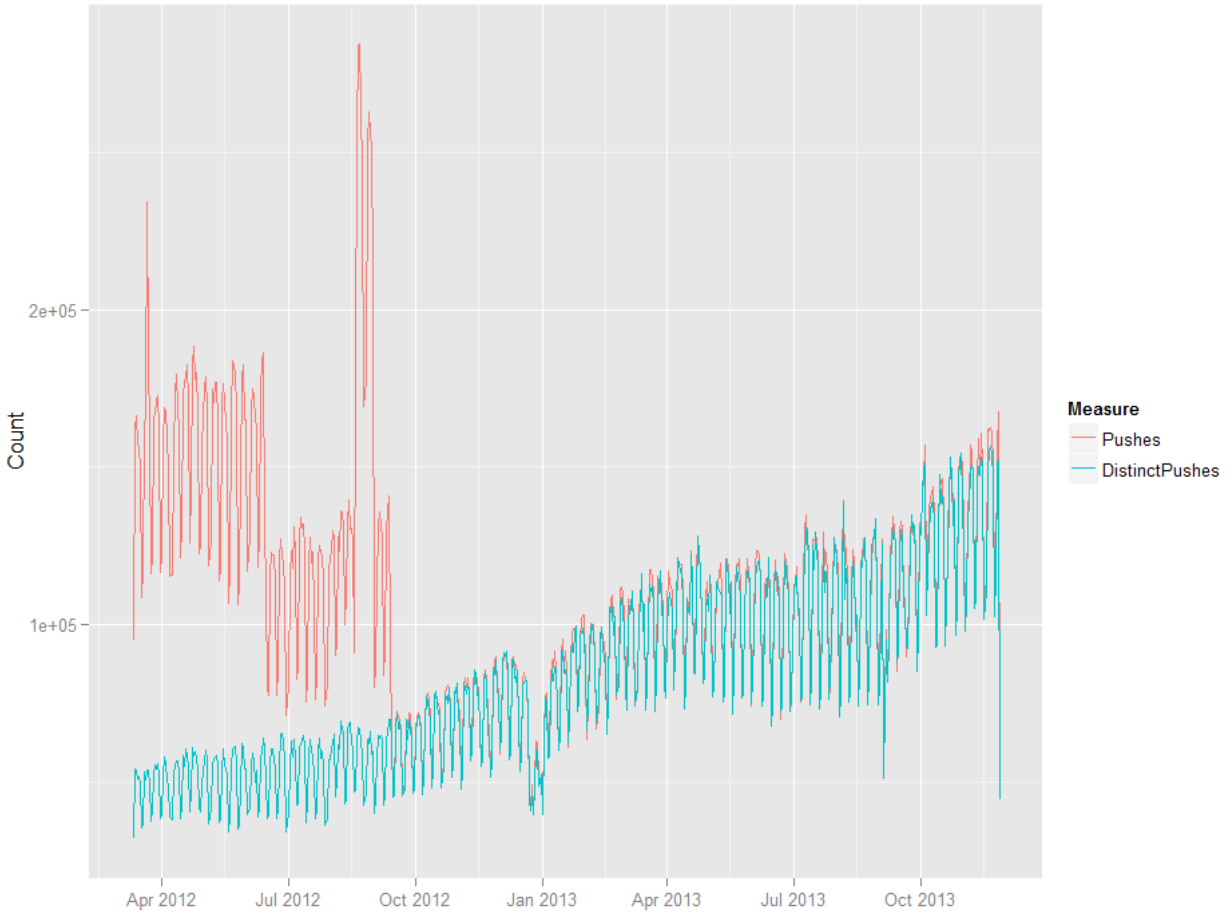
```
wget http://data.githubarchive.org/2011-04-{01..30}-{0..23}.json.gz
```

brings down all the hourly data files for 1-30 April 2011. But in the spirit of contemporary data analytics and its increasingly cloud-centric infrastructures, we decided to use Google's BigQuery system just as competitors in the Github data competition had.

But the power of highend data analytics comes at a cost. For instance, if we just trusted what is in the timeline dataset on BigQuery, we might think that something amazing was happening on Github in 2012.



Not only was there more activity on Github in April 2012 than 2014, there was a massive spike in activity during September 2012. This is completely counter to normal trends in growth. But a quick look at the data around September shows there is heavy duplication of events recorded on the timeline. Something happened here in the pipeline between Github and githubarchive.org and the bigquery.cloud.google.com.



There are couple of possible responses to this spike. We could say the data is very messy data and needs to be cleaned up. Yes, that is possible, but it is quite hard to actually eliminate duplicate events in the timeline dataset. Another response is to see this dirtiness as part of the sharing, and therefore not something to be cleaned away, but to be explored carefully. Here I am suggesting that we should see datasets like this and databases more generally not as as inert objects but as open, processual and emergent forms with many of the attributes and propensities of human informants that we might more typically work with in ethnographic research. Just as it can be hard to get to know human informants and to establish good dialogues with them, databases and datasets need to be treated less automatically if we want to make sense of what is happening in them.

At a time when sharing was the modus operandi of media platforms in the way it is today, the anthropologist Ulf Hannerz more than twenty years ago wrote that 'there is nothing automatic about cultural sharing,' (Hannerz, 1992, p. 44) . Hannerz suggests that there are real complexities in the way that things are shared. His argument remains important, even as sharing is somewhat automated through devices such as git and platforms like Github. If Github.com is widely regarded as the exemplar of code sharing today, the ways in which that sharing is accomplished might also need to be understood through the processes of *non-sharing* on which it relies.

The process of tracking sharing and non-sharing on the various scales and intricate patterns of practice of complex contemporary culture is not at all straightforward. Patterns of sharing and non-sharing, it turns out, entwine with problems of scale in both contemporary culture and in its analysis. The case I draw on here – Github.com – is interesting precisely because what goes on there – open source software development – is both typical of the vast and somewhat incoherent sharing that makes contemporary culture, and plays an important role in reshaping them (to the extent that software ever does this). It exemplies the gamut of software-related practices. More broadly, I am asking how *distributed revision control* – this is what git does as an inscriptive device

– might help us think about contemporary experience and the power-geometries of sharing.

Mirrors and forks

Many major repositories are clones or mirrors of code held elsewhere. As it became more popular, Github also became a place to put things made elsewhere. We need to decide in looking at a given repository whether we are seeing something made elsewhere and put on Github to advertise and publicise it, or seeing something that exists primarily on Github. That is, is Github being used as a download site, a cleaned-up version of Pirate Bay or MegaUpload, a place that is used to *mirror elsewhere*? Second, even if we decide that the repository is indigeneous to Github, that is, it grew there, further questions arise. Many repositories are derivatives of other repositories. Many repositories are *forks:* repositories that copy or mirror other repositories and then vary them slightly, reconfiguring certain aspects. This process of forking repositories is a key event in the social life of Github platform. It is something like 'following' on Twitter or 'liking' on Facebook, but with potentially much more open-ended consequences. A fork can become more important than the original repository it copied. Forks are sometimes re-incorporated into the original or parent repositories that came from. Forks and mirroring complicate the flow of code on github. Do forks increase the importance of a repository or do they dilute it? That question remains open.

Organising, watching and starring

Finally, to add to the complexity, Github adds many layers of social organisation that pile on top of the repositories. People do not only fork or work on their own repositories. They watch other repositories for changes. Watching activity fluctuates greatly. It swings in response to external news, and many other social media dynamics.

In addition, repositories on Github do not stand in isolation. They are grouped around other actors on various scales. For instance, there are around 80,000 organizations on Github, alongside the roughly hundreds of thousands of individual 'actors' . Like the repositories themselves, the organisations can be indigenous to or growing on Github, or can be seen as colonising or immigrating. While existing organisations sometimes have repositories, these repositories may not be very significant organisationally or not even a coherent expression. For instance, at first glance the organisation BBC has only has 6 repos (https://github.com/BBC) and they are not updated very often. But BBC has many different organisations on the platform: BBCNews, BBCFrameworks (https://github.com/bbc-frameworks), BBC R&D (https://github.com/bbcrd), BBC Data (https://github.com/BBC-Data) BBC Future Media or BBC World Wide, etc. repos are distributed more widely on Github. Large organisations like the BBC are present in complicated ways on Github. Sometimes their repos contain work specific to their activities, but very often their repos are clones or forks of repositories outside the organisation. This plurality of relations suggests that organisations inhabit the platform in order to leverage the work of the github metacommunity. (Something similar holds for *The New York Times*). Or to take another example, the social media platform Facebook has hundreds of repository on Github. In many ways this is strange because Facebook as a social media platform would normally not put part of itself on another platform. But the way that is developers coordinate their work Github, and the profile of their many repositories there can tell us something about the organisation of Facebook itself. A part of Facebook and perhaps an important part is on Github.

The experience of stream

In short, Github as a social media platform for software development constructs massive feedback loop where assessments, adjustments, and re-alignments of software-related activities mingle and meet, are negotiated, proliferate and change. Rather than github simply being a mirrored expression of the contemporary world and its saturation by software, it becomes very diffracted, a kind of fly-eye multiple perspective. How should we make sense of this budding and burgeoning character?

I will introduce a rather philosophical point here and then exemplify it in relation to Github. In contrast to many treatments of social media platforms as either sites of ethnographic study or as data sets to be analysed in terms of categories, clusters or correlations, I would like to suggest here that we should treat them both as forms of experience flowing connected with generalised inscriptive technologies. Many science and technology studies scholars are somewhat uneasy about the notion of experience. For good intellectual reasons, they have often preferred to work with conceptual frameworks that owe more to semiotics and structuralism than to phemenological experience. But the notion of experience I draw on here is broadly aligned with the 19th century American philosopher William James who wrote that

> experience as a whole is a process in time, whereby innumerable particular terms lapse and are superseded by others that follow upon them by transitions which, whether disjunctive or conjunctive in content, are themselves experiences, and must in general be accounted at least as real as the terms which they relate. (James, 1996, p. 62)

James' notion of experience, as people like Bruno Latour, Isabelle Stengers, Brian Massumi and Anna Munster have suggested, offers some ways of starting from parts rather than presumed wholes, in terms of belief rather than knowledge, in terms of events rather than things, and with a particular emphasis on networks of relations. What I'd like to take from James' notion of experience here is this emphasis on experience itself as only somewhat coherent. Any coherence it has depends on accepting transitions as real as the 'terms which they relate'. Transitions can be continuous or somewhat discontinuous, but in in whatever way they occur, the transitions themselves are experiences. Starting from this principle of symmetry between terms and relations, James defines a form of empiricism focuses on transitions:

> 'radical empiricism takes conjunctive relations at their face value, holding them to be as real as the terms united by them. The world it represents as a collection, some parts of which are conjunctively and others disjunctively related. Two parts, themselves disjoined, may nevertheless hang together by intermediaries with which they are severally connected, and the whole world may hang together similarly, inasmuch as some path of conjunctive translation by which to pass from one of its parts to another may always be discernible. Such determinately various hanging-together may be called concatenated union ((James, 1996, p. 107)

The description of the world here is curiously incoherent since it is a 'collection of parts' in various relation to each other, sorted by degrees of coherence that can be experienced and known through paths of 'conjunctive translation.' (This bears strong resemblances to some Actor-Network theory formulations). Even if on Github, these paths between parts are largely inscriptive translations, they still concatenate in various ways. Experience is the names for the process of translation in which paths are travelled. There is no travelling to see the world in this account, but seeing, reading, discerning as a travelling that makes the world.

Let us see whether James' notion of experience and the associated radical empiricism helps us traverse and see Github differently. What I have shown you of Github so far depends on nothing more than web pages that anyone can see by looking at Github.com. But most of what happens on Github is not the result of people pointing and clicking in Web browsers. On the contrary, Github is mainly fed by flows of inscriptions generated by git clients and fed into Github databases. These feeds of code and other text are complicated in ways that it would be interesting to follow in more detail. What happens if we try to expose ourselves to those flows of events that happen to take to the form of textual revisions?

The timeline: pushes, pulls, forks, and so on

A distributed revision control system such as *git* generates many records as people make changes to code and *commit* those changes to repositories. Repositories are read and written in overlapping patterns on Github at a high speed. Every hour, thousands of events occur. Between 10-11am, 10 January 2013, the following events occurred:

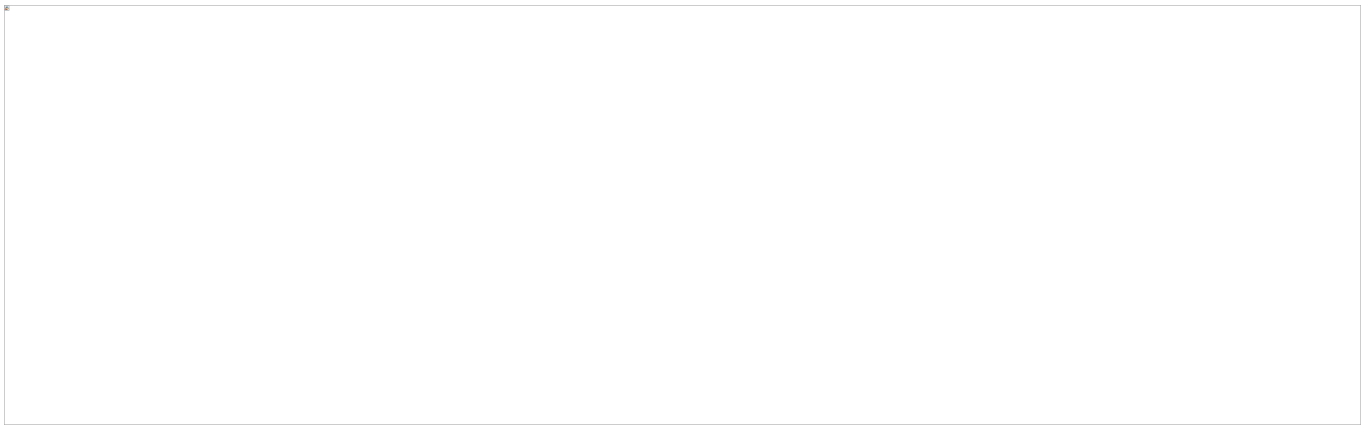
PushEvent	4248
IssueCommentEvent	1104
CreateEvent	935
IssuesEvent	770
WatchEvent	758
PullRequestEvent	382
ForkEvent	328
FollowEvent	193
GollumEvent	153
PullRequestReviewCommentEvent	119
CommitCommentEvent	86
DeleteEvent	83
MemberEvent	70
GistEvent	30
PublicEvent	2
DownloadEvent	1

The events are mainly PushEvents where people put code into repositories. But many of them are meta-events, such as making comments on code, creating new repositories, making copies of existing repositories or starting to watch repositories. So we have these events occurring in continuously varying quantities:

These events can be read from the Github api https://api.github.com/events very readily, along with many other types of attributes associated with users, organizations, repositories, the code itself, etc. Some of this data is limited. For instance, the publically timeline data only really goes back a few minutes. But developers have so much interest in this data that is it is archived on its own site http://www.githubarchive.org/ and also loaded every hour onto Google's cloud computing service called BigQuery https://bigquery.cloud.google.com/table/githubarchive:github.timeline. Other data on the github.com API, for instance, data on git 'commits' to repositories https://api.github.com/repos/torvalds/linux/commits is completely unlimited, and has to be, otherwise the git mechanism would begin to break down.

This tension between two different modes of data is symptomatic of the tensions in sharing and non-sharing often encountered today. The timeline data is almost impossible to follow or keep hold of without turning to large scale infrastructures such as BigQuery. That brings new complications in

practically dealing with the data. In terms of the radical empiricist account of experience I'm pursuing here, it suggests that we should expect events to hang together by themselves, but only by virtue of intermediaries.



Conclusion

Github is interesting I have been suggesting because it combines a couple of different symptomatic features. Given its concern with practices of making and distributing software, the ways of making and working taking shape act as models or exemplars of sharing and coordinated work. Given also the way in which software imbues our worlds with structures and relations of control and interaction, the pattern of projects found there matters. It tells us something about what is happening and often about to happen in the world. The vastness of Github is like a monadic view on the whole. As Latour, Jensen, Venturini and ?? (Latour et al., 2012) suggest, we can build social theories by following the chains of links that compose the fabric of the social. Database or dataset navigation suggests a different kind of social theory. I am somewhat sceptical about the smoothness of this navigation.

I have been suggesting too Github is interesting too because as a social media platform for software development, it lends itself – somewhat – to the technological modifiability of its conditions of existence. While Github seeks to designate what it does under the 'social' injunction of 'share,' the distributed work that goes on there is very little like the holding in common and the nearness we might associate with sharing. The real sharing on Github resides much more at the level of the code, where shared dependencies heavily different repositories, sometimes linking them together, but aslo extending well beyond Github itself to the many other libraries and repositories where software developers get things they want to use.

Finally, and perhaps most important, I have been seeking to demonstrate and corroborate something of what Marres and Weltevrede write when they say 'render traffic between the object and process of social research analytically productive' (Marres and Weltevrede, 2013, p. 3). The particular forms of traffic I have focused on concern how this traffic moves.

References

Couldry, N., 2000. Inside culture : reimagining the method of cultural studies. SAGE, London.

Grigorik, I., 2012. GitHub Archive [WWW Document]. URL http://www.githubarchive.org/ (accessed 3.31.14).

Hannerz, U., 1992. Cultural complexity : studies in the social organization of meaning. Columbia University Press, New York ; Oxford.
Hardy, Q., 2012. Dreams of “Open” Everything [WWW Document]. Bits Blog. URL http://bits.blogs.nytimes.com/2012/12/28/github-has-big-dreams-for-open-source-software-and-more/ (accessed 3.27.14).
James, W., 1996. Essays in radical empiricism. University of Nebraska Press, Lincoln.
John, N.A., 2013. Sharing and Web 2.0: The emergence of a keyword. New Media Soc. 15, 167–182. doi:10.1177/1461444812450684
Kelty, C.M., 2008. Two bits : the cultural significance of free software. Duke University Press, Durham.
Latour, B., Jensen, P., Venturini, T., Grauwin, S., Boullier, D., 2012. The Whole is Always Smaller than its Parts. How Digital Navigation May Modify Social Theory. Br. J. Sociol. 63, 590–615.
Marres, N., Weltevrede, E., 2013. SCRAPING THE SOCIAL? Issues in live social research. J. Cult. Econ. 1–23.
Meyer, R., 2013. Github, Object of Nerd Love, Makes Play for Non-Programmers [WWW Document]. The Atlantic. URL http://www.theatlantic.com/technology/archive/2013/08/github-object-of-nerd-love-makes-play-for-non-programmers/278971/ (accessed 2.13.14).
Neilson, B., 2012. Five theses on understanding logistics as power. Distinktion Scand. J. Soc. Theory 13, 322–339.
Strathern, M., 2006. Imagined collectivities and multiple authorship, in: Ghosh, R.A. (Ed.), Code: Collaborative Ownership and the Digital Economy. MIT, Cambridge, MA, pp. 13–28.
Tsing, A., 2009. Supply chains and the human condition. Rethink. Marx. 21, 148–176.
Uprichard, E., 2012. Being stuck in (live) time: the sticky sociological imagination. Sociol. Rev. 60, 124–138. doi:10.1111/j.1467-954X.2012.002120.x