Metacommunities of practice in the code-sharing commons

Adrian Mackenzie (Lancaster), Matthew Fuller (Goldsmiths), Andrew Goffey (Nottingham), Richard Mills (Cambridge), Stuart Sharples (Lancaster)

December 2014

Topics of the paper online data, analytical and presentational tool

· transmit best practice in use of such tools for social science; · show how the tools can be used to test social-science theories; and · suggest possible improvements/innovations in the tools to help integrate analytics and open-source data tools in general into the teaching/learning community

ideas for esrc dragons den paper

- relationality is the potential for singular effects of qualitative change to occur in excess || over or as a supplement to objective interactions.

– we didn't do that; but we could use this to identify organisations who we could work with - embedding findings in the field - computer scientists and statisticians as buddies - tensions around that; - newbies to the domain – qualitative and quantitative thing – voluntary naivete - how the api shapes what you can find out – our attempts to find stuff out; failure to address the infrastructures that produce data - do you need a lab? there is a need for combining critical perspectives and capacity to do analysis; how do scientists work with their devices to modify them; uncomfortable message for esrc — you have to get geeky; - what we couldn't do – are we competitors in the fields of github data challenges? william bunge using quantitative data to show inequality; – conjunctions; lack-adaisical solidarity; commons with potential forms of solidarity; - what we did with the namespace – messy data that works against the api – other examples of working against the api?

quotes to use

Overview

> A metacommunity is a set of interacting communities which are linked by the dispersal of multiple, potentially interacting species

(Leibold et al. 2004).

> One of the areas that being most dramatically shaken up by $N = All$ is the social sciences. They have lost their monopoly on making sense of empirical social data, as big-data analysis replaces the highly skilled survey specialists of the past. … More important, the need to sample disappears (Mayer-Schönberger and Cukier 2013, 30).

The 'Metacommunities of practice in the code-sharing commons' project was an $N = All$ style data analysis. By $N = All$, we mean that it sought to make use of all the publicly available data in the domain of research. The title of the project is a slightly loopy one, but relates to this $N = All$ approach in important ways. First of all, the domain of the research was software, code, and coding practices as seen in public software repositories. We focused almost solely on Github.com, the leading public code repository today. We could have done comparative work with other platforms such as GoogleCode or SourceForge, but working with all the data from Github was substantial enough. We did not sample code and coding practices on Github. We sought to analyse them in their entirety precisely because we wanted to look for evidence of the *commons* in code. Our guiding theme of 'meta-community' was meant to open up a different perspective on the flow of code across platforms, devices, apps and applications. This perspective would contrast with much of the existing social science that focuses right down on exemplary cases (often by doing fieldwork with particular communities of software developers such as Debian (Coleman 2012)), and diverge also from the extensive work done by computer scientists and software engineers who do look at the aggregate phenomena but tend to focus on issues of productivity, reliability or code quality on particular projects (often by using software repositories as their data source (Godfrey and Whitehead 2012)). In contrast with both the existing social science and the computer science, we were looking for transverse flows and connections between different parts of the seething pool of software development. More importantly, we were treating the repositories in aggregate as the form to be analysed, rather than seeing them as a site for sampling of independent, representative cases. The overarching goal was something like what , as Viktor Mayer-Schőnberger and Kenneth Cukier suggest in *Big Data*: 'using all the data makes it possible to spot connections and details that are otherwise cloaked in the vastness of the information' (Mayer-Schönberger and Cukier 2013, 27).

Research objectives and their practicalities

Our research objectives in order of priority were:

1. Using datasets generated from publicly accessible software code repositories and programmer question and answer (Q&A) sites, to explore

ways of tracking programming practices as they move through digital economies/network media cultures.

2. To develop an empirically rich and reusable database of evidence describing the extent and diversity of code sharing practices across a comprehensive range of open-source software projects during the period 2008-2012.
3. To explore and document how data analytic techniques including visual data exploration, statistical machine learning and predictive models can be re-purposed to focus on questions of practice that are normally seen as the province of qualitative case-studies.
4. To investigate whether analytical and methodological problems in the social sciences often framed in terms of the 'empirical crisis of sociology' (Savage 2009) or the dominance of social sciences by the 'general linear model of reality' (Abbott and Tsay 2000) can be addressed through a re-conceptualisations of what it means to track practices (Latour et al. 2012).
5. To demonstrate reproducible social scientific research in the form of tightly integrated 'virtual machine' distribution of data, analysis, code, text and visualizations.

In the course of the project, which ran for around 14 months overall, we did not meet all of these ambitious objectives. We did however address all of these objectives in various ways, ranging from using large public API (Application Programmer Interface) datastreams, using big data analytic tools such as Google BigQuery, and working with various analytic techniques coming from social network analysis, and statistical learning. As well, we worked extensively with the reproducible research tools and platforms in order to test how and to what extent reproducible research can be conducted using tools available in the wild rather than constructed as a bespoke or purpose-built platform for data analysis. As usual, we altered and reoriented some objectives, and curtailed others in order to keep the research in movement through its different phases.

Using public datasets

The metacommunities concept, with its focus on dispersion, on multiple 'species' (this could be understood in various ways in our domain), and on potential interactions, suggested a need to work with the data in aggregate. While we ended up focusing almost all of our efforts on the data that relates to the Github software repositories, we also made some of public data dumps of the popular Q&A site StackOverflow.com to help identify important processes of interaction between developers.

We faced some fairly fundamental practical choices in working with the Github data. We could harvest data from Github's extensive APIs (see https://api.github.com/):

```
  {
  "current_user_url"
  "current_user_authorizations_html_url"
  "authorizations_url"
  "code_search_url"
  "emails_url"
  "emojis_url"
  "events_url"
  "feeds_url"
  "following_url"
  "gists_url"
  "hub_url"
  "issue_search_url"
  "issues_url"
  "keys_url"
  "notifications_url"
  "organization_repositories_url"
  "organization_url"
  "public_gists_url"
  "rate_limit_url"
  "repository_url"
  "repository_search_url"
  "current_user_repositories_url"
  "starred_url"
  "starred_gists_url"
  "team_url"
  "user_url"
  "user_organizations_url"
  "user_repositories_url"
  "user_search_url"
}
```

Each item in this list designates a specific facet of the Github platform data. The `events_url` is probably the most general one since it offers almost live access to everything that people do on Github. But as this list shows, we were facing several dozen different access points to what was happening on Github. The data available through the APIs varies in volume, and rate-limits apply to many of the API points of access, but in general we were faced with an abundance of data, of varying duration. The problem was more one of being spoiled for choice.

Even given this abundance of data, several other major viable alternatives presented themselves. This may not be typical of all social media data, but in the Github case, Github API data from the `events` API has been archived in a public archive http://www.githubarchive.org by a Google employee named Ilya Grigorik for the last year or two (Grigorik 2012). This archive was pub-

lished after we submitted our grant application, but as soon as we discovered it in early 2013, we seriously considered using it because it does not suffer from the rate-limits that Github's own APIs impose. Initially, GithubArchive only reached back to early 2012, but now it extends back to February 2011, so offers a timeline datastream of every public event on Github for the last three years. This dataset can almost be wrangled on a single machine. It totals around 179 Gb compressed, and consists of a file for every hour since early 2011. As is often the case in software cultures, the code that runs GithubArchive can itself found on Github as a `git` repository: https://github.com/igrigorik/githubarchive.org.

A quick glance at that repository shows that GithubArchive.org feeds directly into another major Github data resource, the GoogleBigQuery data analytics platform, which hosts the full GithubArchive dataset at GithubArchive.timeline. On GoogleBigQuery the formatting of the event data is quite different. The data is stored in a single huge table, with almost 200 columns and, at the time of writing, around 280 million rows. This massive table can be queried using an SQL (Structured Query Language) dialect.

While we did use the Github APIs, and wrote scripts that allows us to move back past the February 2011 limits of the GithubArchive data, having the full `events` timeline in both the compressed forms of hourly files from GithubArchive and in the GoogleBigQuery table made any efforts to construct our own dataset from scratch seem somewhat futile. But working with either the GithubArchive or the GoogleBigQuery data brings its own problems. GithubArchive is in around 30,000 files that need to be read or loaded into memory in order to work with them. Even using GoogleCompute platform instances as we did, processing those files takes takes many hours. We did process all the files several times, in particular in pursuit of a full description of the terrain covered by the 12 million or so code repositories on Github, but the result of the processing were new database summaries of the Github data (for instance, we build a Redis database to give us quick access to the main attributes of all the repositories, and all the user attributes whenever we needed it). The problem with these full dataset traversals is that each time they are run, they produce a new summary or synoptic dataset, but a slight change or variation in the information needed for a particular analysis means running all the scripts again. This is not a good workflow.

Most of our daily analyses were constructed then using GoogleBigQuery. As mentioned above, the advantage of this platform is that it allows SQL-style queries to be run against a large, albeit not quite $N = All$ Github dataset. On the one hand, these queries can be done interactively through a website interface (especially useful for incrementally constructing queries), but on the other hand, the nature of the GithubArchive data stored on GoogleBigQuery means that queries are quite difficult to construct. They are often highly intricate, nested queries.

It should be noted too that these data analytic tools are changing rapidly. When we first started working with GoogleBigQuery, we were writing quite a lot of

Python code to set up a query, execute and then reshape the data it returned into forms that we could work with. A typical set up query in June 2013 began something like this:

```
def query_table(query, max_rows=1000000, timeout=1.0):

    """ Returns the results of query on the githubarchive
    args:
    ----------------
    query: a BigQuery SQL query
    max_rows: maximum number of rows to return
    timeout: how long to wait for results before
    checking for actual data in the rows

    returns:
    ---------------------------
    results_df: a pandas.DataFrame of the results
    """

    try:
        bigquery_service = setup_bigquery()
        job_collection = bigquery_service.jobs()

        # put limit on query if it doesn't have one
        if query.lower().find('limit') == -1:
        query = query.replace(';', ' ') + '   LIMIT %d' % max_rows + ';'

        print 'executing query:'
        print query
        query_data = {'query':query}

       query_reply = job_collection.query(projectId=PROJECT_NUMBER,
                                    body=query_data).execute()

        job_reference = query_reply['jobReference']

        while (not query_reply['jobComplete']):
            print 'Job not yet complete...'
            query_reply = job_collection.getQueryResults(
                            projectId=job_reference['projectId'],
                             jobId=job_reference['jobId'],
                             timeoutMs=timeout).execute()

        results_df = pn.DataFrame()
```

By January 2014, the query, including all the setup (not shown in the code

vignette above) could be written in one line of `R` something like this:

```
query_exec(project="metacommunities",  query=sql)
```

This contrast is probably commonly experienced by researchers working with data analytics tools and commercial data sources. The churn of tools and infrastructures can make even quite recent work, and indeed investments in infrastructure somewhat redundant. Much of the code we wrote to wrangle GithubArchive and GoogleBigQuery in the first half of the project is already somewhat beside the point in terms of ongoing use. At the time, it was the only way to work with the data, but because so many people are working on similar problems, the tools changes quickly.

% latex table generated in R 3.1.2 by xtable 1.7-4 package % Wed Dec 17 13:54:55 2014

	data types
boolean	7
integer	41
string	151

Table 1: Columns in GithubArchive on GoogleBigQuery

As Table **??** shows, 75% of the fields in the GoogleBigQuery dataset are string, around 20% are numerical count data, and the remainder are boolean or True/False values. But for any given event, many of these fields are likely to be empty because the GoogleBigQuery has been constructed to accommodate every conceivable event type without consideration of the distribution of events. Having all the attributes of every different event type stored in a single table, and trying to avoid moving things in and out of GoogleBigQuery too much leads to quite complicated SQL queries. The tremendous advantage of the GoogleBigQuery is freedom from the need to think very much about platform computing capacity. But adapting ourselves to the constraint to build densely nested queries that extract everything we want from a single table of GoogleBigQuery was quite hard.[1]

To both give an idea of the distribution of event types, and the relative distribution of events using GoogleBigQuery is relatively straightforward. A query to count show the frequency of events on Github is:

[1] As mentioned above, GoogleBigQuery can be accessed through a web interface, which is good for testing queries, and programmatically using scripts written in `R` or `Python`. We used scripts to run full queries. On one occasion, a `Python` script ran up a query charge of $US2400 overnight. We were shocked and dismayed at the cost. After verious discussions with the GoogleBigQuery engineering team, we received a credit for 75% of the charge. We also had quite an interesting and length discussion with Google Marketing team in California about some problems in using GoogleBigQuery. Soon after this event (October 2013), Google announced a massive reduction in the price of using GoogleCompute.

```
SELECT type, count(type) as event_count FROM [githubarchive:github.timeline]
group by type order by event_count desc
```

% latex table generated in R 3.1.2 by xtable 1.7-4 package % Wed Dec 17 13:54:55 2014

	type
1	CommitCommentEvent
2	CreateEvent
3	DeleteEvent
4	DeploymentEvent
5	DeploymentStatusEvent
6	DownloadEvent
7	FollowEvent
8	ForkEvent
9	ForkApplyEvent
10	GistEvent
11	GollumEvent
12	IssueCommentEvent
13	IssuesEvent
14	MemberEvent
15	MembershipEvent
16	PageBuildEvent
17	PublicEvent
18	PullRequestEvent
19	PullRequestReviewCommentEvent
20	PushEvent
21	ReleaseEvent
22	RepositoryEvent
23	StatusEvent
24	TeamAddEvent
25	WatchEvent

Table 2: Event types on Github Events API

```
library(bigrquery)
event_query = 'SELECT type, count(type) as event_count FROM [githubarchive:github.ti
event_counts = query_exec(project="metacommunities",  query=event_query)
```

```
## Auto-refreshing stale OAuth token.
```

```
## Warning: cannot open compressed file '.httr-oauth', probable reason
## 'Permission denied'
```

```
## Error: cannot open the connection
```

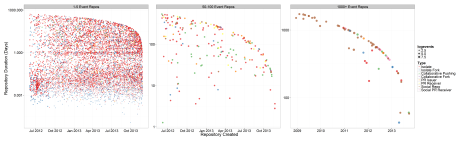

Figure 1: Github repository durations by event count

As we will see, the complexity of practices on Github means that there are more than 20 different event types in the Github event data. On the one hand, this is really helpful and interesting in terms of our aim to analyse the diversity and commonalities of coding practice. On the other hand, it poses some real analytic problems that we struggled with throughout the project. Event types occur in a huge range of patterns. They are dominated by PushEvents, but the event total includes many other other events types, and this suggests that code repositories are much more than just repositories. They are sites of production, sociality and community, not just places where people store code.

Standing back a little from this detail, the important point here is we were easily able to meet our first objective of 'using datasets generated from publicly accessible code repositories' almost by the accident of GithubArchive and GoogleBigQuery choosing to treat Github data as something worth publishing in bulk.

Database on diversity of practices

Our second objective to develop an empirically rich and reusable database of evidence describing the extent and diversity of code sharing practices across a comprehensive range of software projects. This was a slightly misconceived objective in some ways. Evidence of the diversity and commonality of code sharing practice was at the centre of our interest as we analysed what has been happening in the last few years on Github. We did struggle with the pre-2011 history of the platform in terms of data analysis. Like many social media platforms – Github is a social media platform in key respects – Github has constantly changed since its launch in late 2007. Unlike many platforms, in which working with everything that happens in even one month is difficult (for instance, all Tweets for a month), it is possible to work with whole years of Github activity at a time, especially with GoogleBigQuery. This, however, does not mean that we could easily make a database of diversity.

The problem of constructing such a database can be detected in Figure 1. below. The main obstacle in doing that quickly became obvious as we began to work with the data. Although the existence of 12-13 million repositories on Github suggests that there are great variety of different things happening on Github, when you count those things, some familiar patterns begin to appear. The availability of all the Github data means that making a database of the data

Learning and modelling diversity of practices

Tracking practices in the field of devices

Reproducible research

Conclusion

- how Github differs from much more complicated twitter-based analysis
- why Github matters as a social research topic

- ## References

Abbott, A., and A. Tsay. 2000. "Sequence Analysis and Optimal Matching Methods in Sociology: Review and Prospect: Sequence Analysis." *Sociological Methods & Research* 29 (1): 3–33.

Coleman, Gabriella. 2012. *Coding Freedom: the Ethics and Aesthetics of Hacking*. Princeton N.J.: Princeton University Press.

Godfrey, Michael, and Jim Whitehead. 2012. "Introduction to the Special Issue on Software Repository Mining in 2009." *Empirical Software Engineering* 17 (4-5) (August): 345–347. doi:10.1007/s10664-011-9188-2.

Grigorik, Ilya. 2012. "GitHub Archive." http://www.githubarchive.org/.

Latour, Bruno, Pablo Jensen, Tomasso Venturini, S. Grauwin, and D. Boullier. 2012. "The Whole Is Always Smaller Than Its Parts. How Digital Navigation May Modify Social Theory." *British Journal of Sociology* 63 (4): 590–615.

Leibold, M. A., M. Holyoak, N. Mouquet, P. Amarasekare, J. M. Chase, M. F. Hoopes, R. D. Holt, et al. 2004. "The Metacommunity Concept: a Framework for Multi-Scale Community Ecology." *Ecology Letters* 7 (7): 601–613. doi:10.1111/j.1461-0248.2004.00608.x. http://onlinelibrary.wiley.com.ezproxy.lancs.ac.uk/doi/10.1111/j.1461-0248.2004.00608.x/abstract.

Mayer-Schönberger, Viktor, and Kenneth Cukier. 2013. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Eamon Dolan/Houghton Mifflin Harcourt. http://books.google.co.uk/books?hl=en/&lr=/&id=uy4lh-WEhhIC/&oi=fnd/&pg=PP1/&dq=schonberger+big+data/&ots=Jrk7hiJVHT/&sig=QVKugcrFF4Jq5eO7xd8exEEG_Hk.

Savage, Mike. 2009. "Contemporary Sociology and the Challenge of Descriptive Assemblage." *European Journal of Social Theory* 12 (1) (February 1): 155–174. doi:10.1177/1368431008099650. http://est.sagepub.com/cgi/content/abstract/12/1/155.