6 Code traffic: code repositories, crowds and urban life

Adrian Mackenzie

Introduction

> The initial step can be made through the venerable geographical act of mapping the expanding realm of machinekind, clearly part of the remaining *terra incognita*.
>
> (Horvath 1974: 188)

In what frame and at what levels of abstraction does the density and plurality of code in the city become legible or even enumerable? Writing in 2002, Nigel Thrift and Shaun French addressed a version of these questions: 'Is there any way of making a more general assessment of software in the city?' (Thrift and French 2002: 314). They sketched some possibilities, ranging from hegemony to haunting:

> It would be easy at this point to fall back on some familiar notions to describe software's grip on spaces like cities. One would be hegemony. But that notion suggests a purposeful project, whilst software consists of numerous projects cycling through and continually being rewritten in code. Another notion would be haunting. But again the notion is not quite the right one. Ghosts are ethereal presences, phantoms that are only half-there, which usually obtain their effects by stirring up emotions – of fear, angst, regret, and the like
>
> (Thrift and French 2002: 311–12)

In their review of different understandings of software in the city, they affirm something quite elementary: 'numerous projects [… are] continually being rewritten in code'. The rest of this chapter could be seen as an update on that observation from 2002. What has happened to the cycling through and rewriting of code?[1] Thrift and French go on to describe three geographies that were cycling – or as I would prefer to say, trafficking – code through cities: a geography of *writing code*, a geography of *power and control* and a geography of *indeterminancy*.

> The first of these geographies is the most obvious, the large and complex geography of the writing of software – of the production of lines of code – a geography that takes in many different locations and many different languages and which has been built up progressively since the invention of programming in the 1940s.
>
> (Thrift and French 2002: 323)

According to Thrift and French, the geography of software writing clusters around key places and regions: Silicon Valley, New York, London and a number of auxiliary software mass production zones (often concentrating on such tasks as consulting, testing and support) in countries like Ireland and India. China, Russia and Brazil are not mentioned. As we will see, this geographical centring still matters, but in a somewhat reconfigured form: code cycles through platforms that tremendously redistribute the production of lines of code.

A second, less concentrated, geography of power, conceived in Foucauldian terms as the conduct of conduct, or massive proliferation of corporeally practised rules, was also unfurling through software: 'In essence, we can say that it [software] consists of rules of conduct able to be applied to determinate situations,' (Thrift and French 2002: 325). Through this power geography, software increasingly interlinks rather than compartmentalises urban processes. In many ways, the app economy, the virtualisation of computing into the Cloud and containerised infrastructures, and the spectacular growth of social media platforms or, indeed, the Internet of sensor-equipped Things attests to an intensified application of rules to determinate situations. Again, coding itself, the process of specifying, configuring and propagating these rules of conduct, has not been exempt from the conduct of conduct. Coding has been powerfully recoded in recent years.

Thrift and French (2002: 328) envisaged a final more open and less localised geography in which abundance produces indeterminacy and lack of closure:

> The general profusion of software, its increasing complexity and consequent emergent properties, all count as means of producing playful idioms that have not been captured by dominant orders. Software's very indeterminacy and lack of closure provide a means of creating new kinds of order.
>
> (Thrift and French 2002: 328)

In this geography, 'profusion' and 'increasing complexity' generate less orderly or regulated idioms. These playful idioms are largely irreducible to the geographical centres of coding or power-generated control practices, and therefore take on singular forms, arise in unexpected locations and articulate non-representational processes. Thrift and French attributed this generative aspect of software to the phenomenality of code as a form of *traffic*:

> Software is more like a kind of traffic between beings, wherein one sees, so to speak, the effects of the relationship. What transpires becomes reified in actions, body stances, general anticipations. We would argue, then, that software is best thought of as a kind of absorption, an expectation of what will turn up in the everyday world.
>
> (Thrift and French 2002: 312)

The 'traffic between beings' they refer to here, the reification of 'general anticipations', the curiously contrasting descriptions of software as absorption *and* expectation, could be seen as implicitly urban. They concern 'traffic' in everyday worlds. In this third geography, the traffic between beings, we might see processes of composition that not only generate much code traffic, but assemble compositions of people and things that turn up new things.

More than a decade later, these geographies remain in play. Code work is concentrated in more or less the same places; and the coded conduct of conduct certainly continues. But when we think about code in terms of traffic, what has happened to it? What kinds of new order have eventuated?[2] As I will suggest, a complicated set of reorderings of code traffic have occurred. The centred geographies of code production have been somewhat decentred through a much more multilateral or networked flow of code. At the same time, the very arrangements that have dislocated the centres of software production have themselves become the platform for new platform-based centres or hubs for code. In turn, these centres attract and generate saturated streams of coding traffic that bring new indeterminacies, diverse distributions, encounters and adaptations into play.

We could give the code traffic a Deleuzean or a Tardean formulation. Deleuze and Guattari write:

> Assemblages are passional, they are compositions of desire. Desire has nothing to do with a natural or spontaneous determination; there is no desire but assembling, assembled desire. The rationality, the efficiency, of an assemblage does not exist without the passions the assemblages bring into play, without the desires that constitute it as much as it constitutes them.
>
> (Guattari and Deleuze 1988: 399)

Deleuze and Guattari's notion of assembling as passional composition or putting together suggests one way of thinking about what is generated in code traffic. The 'traffic between beings' moves through paths by processes that we might understand, drawing on crowd sociologists such as Gabriele Tarde and Robert E. Park, as *imitation*. For both Tarde, the microsociologist of crowds, and Park, the urban sociologist, imitation powerfully yet unpredictably effects repetition and invention, and generates new urban forms of various kinds (Borch 2005). In particular, Tarde speaks of a 'coadaptation of imitative fluxes, a cooperation, even in an individual brain, but always a

multitude of agents social and infinitesimal, and their ordinary ideas' (Tarde 1902: 270). While there is much to discuss here (for instance, Tarde's political conservatism and anachronism poses analytical problems), the 'coadaptation' of imitations between a multitude of infinitesimal agents, beneath and around individuals, suggests a way of thinking of what happens in code traffic. Understood either as passional compositions or coadapting imitative fluxes, we might come closer to a concrete understanding of how the transient coagulations and diffusions of code traffic may play out in control structures, in architectures, in matters of concern, etc. Examining patterns of imitation in code moves the emphasis away from code-shaping-cities to code-as-crowd. In the code-crowd, mutually shaping imitative fluxes run between people and machines in places.[3]

To treat *code traffic*, in terms of Tardean crowd sociology, as a coadaptation of imitative fluxes is not to deny the production and power geographies of code. It is not to say that code does not still act on cities, on space, on public and private practices. High-profile and much discussed changes taking place in computational platforms (mobile devices, virtualisation and containerisation in the Cloud, etc.) and in algorithmic processes (machine learning) intricately reorganise urban life. What transpires there is rapidly reified in actions, body stances, etc. But it might also be worth seeing how code has become a mixing process, reconfiguring the architectures, logistics and diffuse circulation of individuals in cities. We would, from this standpoint, no longer concentrate on following how software emanates from global production centres as hierarchical or supervisory control structures reorganising cities. Furthermore, we would no longer focus on isolated pieces of software, systems or applications but on the transverse flows that change how code itself moves and takes shape. We would apprehend coding itself as something closer to pedestrian and vehicle movements in a busy street, in which branching, merging, starting, turning and stopping compose transient multiplicities adapted to particular problems and situations. That is to say, we might attempt to see code as noisy and crowded, propagating aggregates in which juxtapositions, proximities and patterns of imitation multiply through each other.

`git` as code traffic: dangerous coagulation or regularised order?

The study of code traffic poses some empirical problems, but we can glimpse some of the traffic in code via increasingly aggregated code repositories. A huge number of code repositories (possibly around 50 million) are now hosted publicly online at a few code repository sites, such as GitHub.com, Bitbucket.com, code.google.com and SourceForge.net. Focusing on one of these repository hosting platforms – GitHub.com, allegedly the 'largest code repository on planet' – might be a way to see traffic in code, and to track how coadaptive, infinitesimal fluxes flow.[4] Code moves in and out of these repositories in many different ways. This movement is not logistically or hierarchically controlled. It constitutes a vast, complex reticular movement.

Much of the flow of source code passes through a single piece of software called `git`. 'Git', an English word for a person who acts foolishly or annoyingly (often in a crowd), was chosen by Linus Torvalds in 2005 as the name for a new concurrent versioning system for source code. In 2002, Torvald's work on GNU/Linux epitomised for many people the emergent power of open source collaboration – 'crowdsourcing' – on the Internet to build things outside the geographies of the software industry. (Even then, as Thrift and French observed, Linux was a quite centralised hierarchically and industry-supported software project.) In turn, `git`, a revision control system for code, allows incremental changes to code made by many people working a common software project to merge with or diverge from each other. Today, `git` is probably the most widely used revision control system for code (followed by the interestingly named `subversion`).

What should we make of the control of revisions to code in terms of code traffic? In practice, a whole series of converging and diverging movements of cloning, forking, pulling, pushing, requesting, branching and merging pass through `git`. Often used on the command-line, commands such as `git clone`, `git commit`, `git push`, `git stash`, `git pull`, `git branch` and `git merge` suggest some of the elementary movements that generate code traffic in and out of nodes in `git`-connected bodies of code. For instance, some `git` commands replicate whole bodies of code, while others simply add, remove or alter small bits of code. These scale-variations, I would suggest, matter to the flows of imitation that occur. Movements of code can take place very incrementally, as small bits of code move around, or on a large scale, as whole bodies of code travel between different bits of software. As a contemporary site of coding, `git` merits much more empirical description than this, but for the present purposes `git` instantiates a much more distributed and granular cycling and rewriting traffic in code. These broadened and differentiated movements, while certainly not unique or unprecedented in the history of inscriptive techniques, record-keeping systems or archives, have become chained and interlinked in recent coding cultures in ways that generate the rapid transformations associated with many millions of code projects. Different bodies of code clone and branch off from each other, and occasionally, but not necessarily, merge again. The geography of coding would become less aligned to individual developers in specific times places and times, and more open to a range of different styles of code movement, ranging from an uncontrollably fluxing miasma of microscale projects through to vast hierarchical code architectures (such as the `linux` kernel). They in turn modify the mode of movement of code itself by removing some of the checkpoints, barriers and thresholds between production and deployment, between design and use.

Torvalds did not reckon with social media. Since late 2007, GitHub has provided a hosting platform for many `git` repositories, and now attracts the largest share of code traffic of any code repository. From the perspective of the `git` software, GitHub is just another remote code repository. But given

that many, in fact around 13 million, local `git` repositories have GitHub.com as their remote repository, then GitHub becomes a hub for `git`. The network and code traffic that now runs through GitHub is on the scale of a medium-size social media platform. That is, with 13 million repositories, 6 million developers, and around 250 million events in the public event stream (at the end of 2014), GitHub itself is a kind of code terminus, whose functioning, architecture and machinery symptomatically enunciate contemporary code traffic more generally. In relying on `git`, GitHub.com promotes a radically decentred patterning of code traffic. But this `git`-like mobility of code stands in tension with GitHub's own existence as a platform, a platform for 'social coding'.

Making Github into a `git` terminus

Several hundred GitHub staff ('hubbernauts') are scattered across a dozen or so countries, a somewhat dispersed geography for a relatively specialised software company that started in 2007 in San Francisco. GitHub work life is also putatively non-hierarchical, with no management hierarchy, only a `git`-like structure of fluid teams working on projects.[5] In principle, GitHub itself therefore makes itself into something like a terminus for code traffic, and is, itself, constructed and maintained via just such traffic. The geography of work on GitHub is still very much centred in San Francisco (followed by Portland, Chicago, Seattle, Boulder, etc.), despite claims of decentralisation.[6] As Thrift and French (2002) argued, geographical localities still matter greatly to code. But this largely localised traffic in and around GitHub is perhaps less important than the kind of movement of code traffic it seeks to operationalise. The hubness or centrality of GitHub is something to be produced or made via a combination of geography, rules of conduct and new forms of order flowing through code.

If GitHub itself has been produced using `git`, does it epitomise the kind of code traffic I am attributing to `git`? In some ways it does. If we look at the code generated by the staff listed as the GitHub team (233 at the time of writing), there is some evidence of their `git`-like activity as they write the code that constructs, augments, reshapes or expands GitHub as a platform. Figure 6.1 offers a view of what happens as the several hundred hubbernauts work on GitHub as a platform.[7] It is important to note that only some of the GitHub code appears here. The anchoring repository `github/github`, the repository that contains the code for whatever GitHub becomes, remains private.

The events shown in Figure 6.1 roughly summarise some of the code traffic associated with GitHub's own code during the last few years. We see this traffic in the form of events, social coded repackagings of rawer or somewhat more elementary `git` movements. The mix of different event types still, however, suggests something of the `git` flow of code. While PushEvents embody code writing and ForkEvents imply copying (and hence imitation),

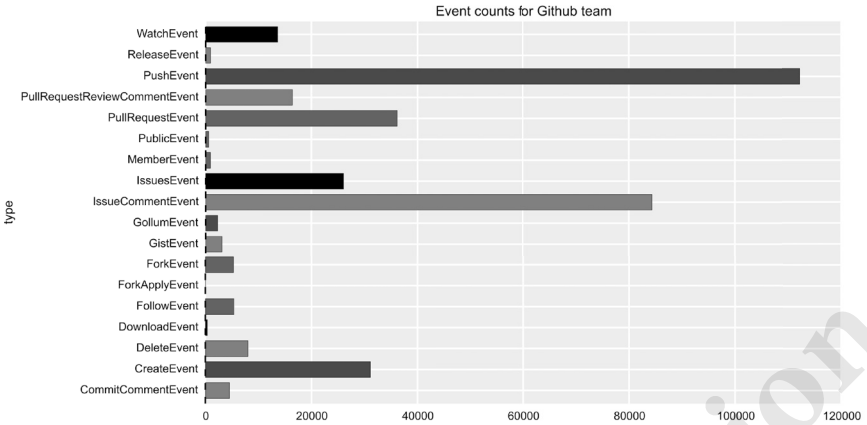

Figure 6.1 Event counts for the GitHub team

many other events, including IssueCommentEvents, PullRequestEvents, IssueEvents and WatchEvents, point to different kinds of traffic in code. In some ways, the variety of event categories shown in the figure already highlights some of the analytical instabilities in tracking code traffic. The events mix `git`-related practices, such as pushing or forking, with GitHub-specific devices such as pull-request or follow that seek to socialise or 'hubify' the movements of code. 'Social' events outweigh technical events, and disentangling what belongs to `git` from what GitHub has added becomes difficult (although not impossible since GitHub must preserve all `git` movements even as it recodes them as 'social'). GitHub as a terminus for code traffic rechannels and reshapes much of the traffic that passes through it. To the ongoing capillary flow of code it adds all the social media-style baggage of followers and watchers, comments, liking and tagging. This socialisation of code traffic, as we see next, generates its own kinds of traffic.

GitHub as collective meta-`git`

How would a social media platform affect the traffic between beings or the cycling–rewriting practices of coding? Hubbernauts such as `defunkt`, `mojombo` or `technoweenie` have thousands of followers and hundreds of repos (i.e., repositories) to their names. While the GitHub team has participated in some 17,000 public repositories in some way (including just watching), they work much more heavily on several hundred of these (see Figure 6.2).

The top repositories display some important features of GitHub as a platform seeking to social reorganise the flow of code traffic. Many of these repositories relate to the building out of GitHub as a massive `git` instance, a kind of meta-`git`. It is hardly surprising that hubbernauts contribute to the `git` project itself or reimplementations of `git` such as `libgit2`, `objective-git`, `rugged`, `hub` (a 'wrapper' for `git`) or `many-branch-repo`. All of these

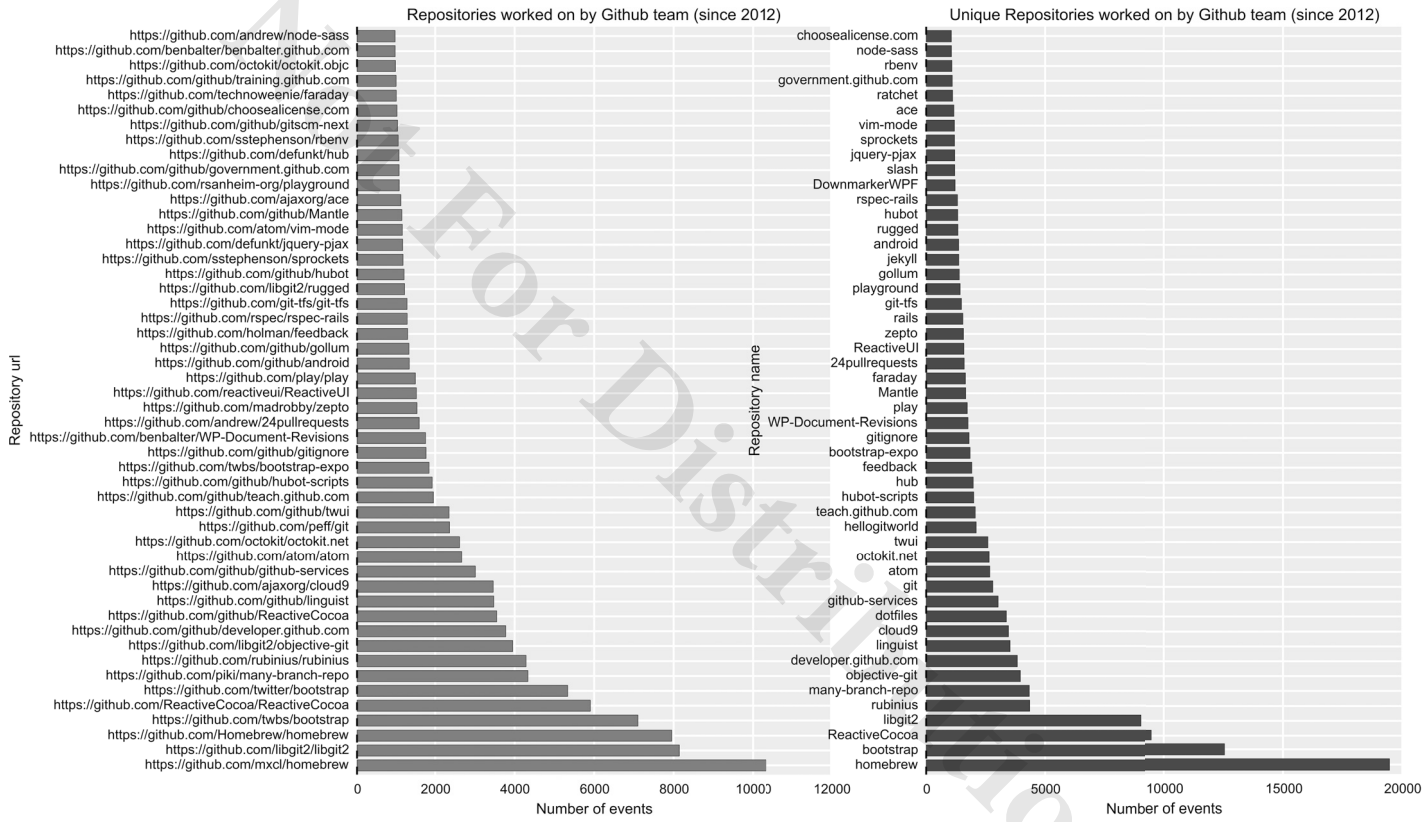

Figure 6.2 Repositories worked on by the GitHub team

projects are imitations, reimplementations or variations on `git`. Hubbernauts also make contributions to repositories that augment or build out GitHub, ranging from documentation websites, such as https://developer.github.com or https://training.github.com, websites that promote or showcase GitHub (https://government.github.com/) or its features (`24pullrequests`), repositories for testing GitHub (`hellogitworld`), repositories that act as question-and-answer sites about GitHub (feedback), repositories that contain code for accessing the GitHub APIs (http://octokit.net) or link GitHub with other Web services (`github-services`) or other code development platforms (`git-tfs`).

Certain elements of the platform generate more code traffic than others. For instance, `libgit2` attracts a large number of imitative events (cloning or forking) because the `git` functionality of committing, branching, merging, forking, cloning, etc., underpins nearly all of the other flows of imitation associated with the growth of 13.2 million repositories and several hundred million events. Typically of such entities, `libgit2` migrates throughout the software development ecosystem so that 'bindings' to `libgit2` have been made in almost any programming language imaginable, from Delphi to Lua, as well as to database back-ends such as `MySQL`, `redis`, `memcache` or the ubiquitous `sqlite`.

Hubbernaut code traffic heavily configures the practices of writing, copying, viewing and packaging code at various levels, and in disparate facets. Heavily trafficked repositories, such as `cloud9`, `dotfiles`, `gitignore`, `vimmode`, `rbenv`, `ace` and `atom`, figure prominently, and attest to a constant retooling and reconfiguring of coding at the level of hand and eye movements. Arranging packages and libraries of code so that they are ready to hand is another major activity. The most active repository, `homebrew`, manages and updates software packages for MacOS computers, popular with coders. There are also a number of repositories that concern how people sit down and keep coding while writing code for GitHub. `Play` is a music server: 'We have employees all over the world, but `Play` lets us all listen to the same music as if we were all in the office together. This has actually made a big impact on our culture,' (https://github.com/github/play). Similarly, `Hubot` and `Hubot-scripts` are part of a software robot system that the GitHub team use extensively to maintain the GitHub platform, run the many online chatrooms they use as they work with each other, continuously deploy the changes they make on the master branch of `github/github` onto their production servers (the servers that actually run the code that makes up GitHub) and send updates to various social media platforms. One Hubot runs the whole of GitHub. Finally, many repositories listed here concern the infrastructure of GitHub as a software platform. Some are language-specific environments heavily used at GitHub, such as `rubinius`, an implementation of the Ruby programming language. Some, such as `rails`, are libraries that provide much of the dynamic infrastructure that holds GitHub as a collection of servers and databases together as a platform. Other repositories,

such as `choosealicence` or `linguist`, implement GitHub features concerned with licensing or tagging repositories by programming language. Since GitHub repositories can also function as webpages, blogs or wikis, other repositories, such as `jekyll` and `gollum`, provide code for that. Other heavily used repositories, such as `bootstrap`, `zepto` and `twui`, provide elements of the graphic layout, colours, styles, fonts and icons that comprise the visual appearance and interactive features of GitHub pages (and I return to this kind of software later).

The fabric of GitHub as a platform is constructed and connected along many edges. It folds in many different elements on various scales, ranging from almost microscopically perceptual configurations, such as editor settings, through to large infrastructural developments, such as `elasticsearch`. GitHub comprises programming language implementations, server infrastructures, deployment mechanisms, Web-frontend frameworks, various social media formats (wiki, blog), Javascript page and graphic elements, as well as the code versioning system of `git`, robots that automate chatrooms and servers that streams music to developers at different places and times. All of these together recursively construct the platform, with varying degrees of coherence and visibility, ranging from the vital `github/github` repository through to the many public-facing repositories that hubbernauts either release to the world or participate in publicly. The broader point here is that code traffic in and around GitHub is itself constantly transformed, modified and intensified by the flows of imitation that hubbernauts themselves semi-consciously generate as they assemble the platform. Music, robots, editors, libraries, databases and webpages intersect with each other as the small-crowd teams of GitHub work and, in the object of their mimetic immersion, with GitHub itself. The flow of elements comprising GitHub is, I suggest, typical of the ways in which code traffic concatenates things today. The mixing of different things in code traffic generates the forms, the changes in scale, the new modes of distribution, partitioning or localisations associated with contemporary life. The platform GitHub, in this case, itself a platform in principle oriented to the multiplication of code traffic, is a deeply passional assemblage, not a rationally ordered technology.

The panoply of code repositories frequented by the GitHub staff suggests something more general about the flow of code traffic. The recursive work of hubbernauts on GitHub can be seen as projected over the three urban-code geographies described by Thrift and French. We can now see that the urban centring of software writing still largely applies, but subject to some significant transformations in how it relates to such settings. The `git` version control processes open centres to more distributed and mutable collectives. The widely scattered geography of the GitHub team suggests something of this. The second geography of power and control (conduct of conduct) no doubt runs through much of what hubbernauts make (and here we could think, for instance, of the social coding of `git` traffic as events, or farther afield, the many DMCA takedown notices posted at https://github.com/github/dmca,

or the recent denial-of-service attacks on GitHub attributed to the Chinese government, or the repo `choosealicence`, which seeks to regulate flows of code according to legal licence). This power geography, however, does not completely supplant the geography of indeterminacy or the 'new forms of order' that GitHub itself as an assemblage exemplifies, even as it builds out the GitHub platform as a hub for widely dispersed and somewhat vagabond flows of code.

Github as convolutional process

The 'social coding' apparatus that is the visible face of GitHub – watchers, followers, stars, showcases, search facilities, along with their attempts to render code flow more like public spaces (see the 'showcases' at http://github.com/explore for examples of this public-making) – does not completely corral or control the imitative fluxes on GitHub. The primary `git` fluxes are more networked than the social media apparatus that GitHub assembles because they are not dependent on the formats and facades supplied by the GitHub platform, but instead mix people and things together. In other words, these *convolutional* fluxes animate the more crowd-like aspects of code traffic, and they criss-cross geographies, cities, feeding into and overloading GitHub geography in key respects.

How would we characterise something of these relatively pre-social imitative fluxes in the GitHub traffic?[8] Some analytical purchase on the convolution of people and things in code traffic can be gained by looking at the full names of repositories on GitHub. Note that these names are not coded by GitHub. Repository names on GitHub are lightly formatted. The first part of a repository name refers to a person or organisation ('mojombo', 'wycats', etc.) and the second to the specific code repository (`grit`, `merb-core`). Both parts of the full repository name are interesting, but the name as whole points to an intersection between groups and people and things, between coders and code. The constant variations of these names bear the traces of the complicated enmeshing of people and things, between coders and code that looms large in code traffic.

For present purposes, however, the second part of the name is more symptomatic since it refers more directly to the code. A simple comparison between repositories on GitHub in early 2007 and in 2014 suggests something of the tension between the efforts to socially code the fluxes and code traffic more generally. Early repositories on GitHub suggest a rather orderly and sensible traffic in beings. Table 6.1 (column 1), which shows the first 20 repositories on GitHub, dating from 2007–2008, lists repositories created and worked on mainly by GitHub hubbernauts themselves and a few others. These are the repositories that somewhat recursively host the code from which GitHub was being made. In the early days of GitHub, these names are largely comprehensible in terms of the GitHub platform itself. Repository names like `grit`, a Ruby-language version of `git`, `git-wiki` or `merb-core` (a Ruby

Table 6.1 20 early and recent repositories on Github.com

2007	*2014*
mojombo/grit	2m1tsu3/practice
wycats/merb-core	tylerdmace/ledomme
rubinius/rubinius	yehiaelghaly/xssya
mojombo/god	istvan-antal/commandjs
vanpelt/jsawesome	JohnKrigbjorn/ObjectOne
wycats/jspec	chenx/Ci35_1
defunkt/exception_logger	mohsenbezanj/AI_Project
defunkt/ambition	Dineshkarthik/blogengine
labria/restful-authentication	sapanbhuta/Sapari
technoweenie/ restful-authentication	gwoodroof/chat-gwoodroof
technoweenie/attachment_fu	tryuichi/Hello-World
topfunky/bong	prateek0020/NepTravelMate
anotherjesse/s3	discoverfly/discoverfly. github.io
anotherjesse/taboo	evan-007/ng-wikiful
mojombo/glowstick	sanemat/zipcode-jp
wycats/merb-more	donreamey/PJKiller
macournoyer/thin	discoverfly/discover
jamesgolick/resource_controller	jkkorean/MIUI-KK
defunkt/cache_fu	rtofacks1/ionic-app
bmizerany/sinatra	jkkorean/MIUI-JB

Web-development framework) nearly all relate to various aspects of GitHub as a platform under development. Other platforms and concerns are already present (`sinatra`) but they are somewhat marginal to the work of developing a platform to host `git` repositories.

A similar list from seven years later in 2014 reveals many complications (see column 2 of Table 6.1). The names of coders have become increasingly unrecognisable, even allowing for the internationalisation of GitHub use that quickly developed between 2007 and 2014. The coder names in GitHub become more and more like random patterns of key presses, as if coders have become less individual, more crowd-like in their identities. (Indeed, thousands of repository and actor names comprise key sequences such as `qwerty`, `asdf`, `poiu`, `lkjh` or `1234`, all of which derive from the keyboard layouts.) Something similar happens to the repository names. Only a few components from the early days are recognisable in type (that is, things like a `blogengine` or `footer-fixed-bootstrap`, and, as we will soon see, the `bootstrap` Web-development framework is almost a fixation for GitHubbers, since it provides the look and feel of GitHub itself). Many other git elements of this stream are also recognisable and proliferate widely on GitHub: `practice`, `Hello-World`, `testing` and `temp` repositories occur in huge numbers on GitHub (of the order of millions or more). These trivial fluxes are like people starting to edge into a crowd, to form part of

Table 6.2 Most-forked repositories on GitHub in January 2013

Forks	*Repository*
1478	`bootstrap`
1276	`Spoon-Knife`
763	`dotfiles`
504	`rails`
426	`html5-boilerplate`
410	`jquery`
375	`homebrew`
345	`linux`
343	`android`
291	`phonegap-plugins`
280	`node`

a mass on the move. Broad swathes of generic imitation surface here too. We can also see here the appearance of quite disparate matters of concern: `game-cho-android` and `AI_Project` may have some similarities, but at first appearance they lie quite a long distance apart from each other. It is very hard to see any flux of imitation here. Like the many repositories named using convenient patterns of keystrokes, these repositories seem almost like code-noise. This might support a sense of code traffic as transient and somewhat disorderly multiplicity, but it hardly seems to reflect the coadaptation of imitative fluxes that, following crowd sociologists such as Tarde and Park, we might see as materialising code in the city.

If we start counting imitative events, things look a little different. For instance, in January 2013, around 200,000 repositories were forked (or copied). Forking, as suggested previously, is a basic imitative event in `git`-like practices. The most-forked repositories have a now familiar look (see Table 6.2). This list is reasonably familiar since it has major platforms like `linux`, `android` and `node`, as well as a popular test repository on GitHub called `Spoon-Knife`. But even the most-forked repository in this list, `bootstrap`, is not a single or homogeneous imitative flux. Look at what else was forked heavily in January 2013 relating to `bootstrap`, listed in Table 6.3.

While it was copied more often than any other repository in that month, the `bootstrap` repository itself is accompanied by many variational imitations. The total count of `bootstrap`-related forks during January 2013 is, for instance, 3074, almost twice the number of the forks of `bootstrap` itself (1478). Widening the frame slightly, we can see (Figure 6.3) that since 2012, `bootstrap` has been an important imitative flux running through GitHub, and has been the most highly 'starred' code repository on GitHub for several years.

The stackplot of ForkEvents (or their equivalent and underlying `git clone` operations) associated with the `twitter/bootstrap` repository shows a more differentiated flux of imitation. The imitation varies over time,

Table 6.3 Most-forked `bootstrap`-related repositories during January 2013

Forks	*Repository*
1478	`bootstrap`
109	`bootstrap-datepicker`
84	`jekyll-bootstrap`
60	`bootstrap-wysihtml5`
54	`bootstrap-tour`
48	`Twitter-bootstrap-rails`
48	`bootstrap-sass`
46	`bootstrap-datetimepicker`
43	`jquery-ui-bootstrap`
42	`bootstrap-modal`
40	`wordpress-bootstrap`
36	`bootstrap-daterangepicker`
31	`sass-Twitter-bootstrap`
31	`Bootstrap-Image-Gallery`
26	`rails3-bootstrap-devise-cancan`
26	`bootstrapwp-Twitter-Bootstrap-for-WordPress`
24	`metro-bootstrap`
24	`bootstrap-timepicker`
23	`bootstrap-toggle-buttons`
23	`MopaBootstrapBundle`
19	`twitter.bootstrap.mvc`
16	`CodeIgniter-Bootstrap`

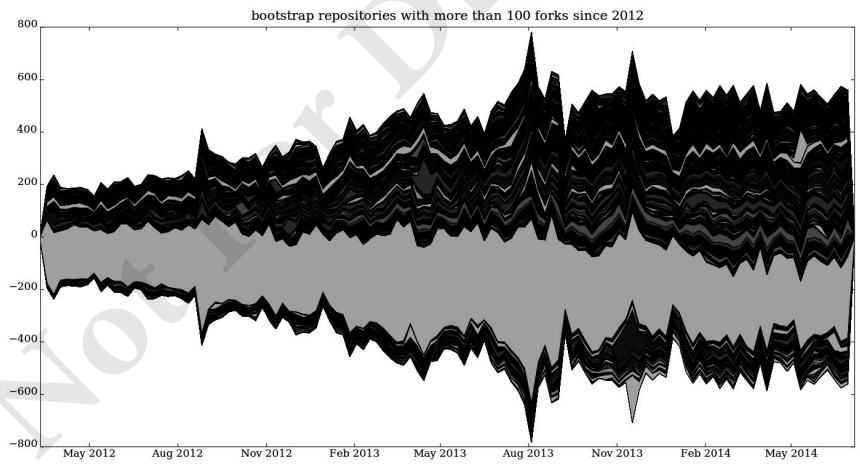

Figure 6.3 Bootstrap repositories with more than 100 forks since 2012

as we would expect, but the patterns of imitation are heavily interconnected with each other through repositories that juxtapose or merge different repositories with each other. The several thousand `bootstrap`-related repositories are much more diverse than `bootstrap` itself. They combine variously

with mobile devices (Android, iOS), with Web browser software (IE6), with various Web-development infrastructures (django, rails, ASP, PHP), with media platforms (WordPress, Google, CodeIgniter) and server management systems (sinatra). They respond to events in the main `bootstrap` repository, but also have a life apart from that repository that relates to other platforms and other software projects. Here, something like the coadaptation of imitative flux begins to come into view. Each of the devices, platforms or systems in these variations has its own code traffic, its own crowds of coders and code, that begin to mix here with `bootstrap`. In this copying, varying, merging and diverging, there is more than simple imitation, but encounters between different things taking shape in aggregate form. This convolutional change is not captured or controlled by the GitHub platform and its social coding devices. It is perhaps true, however, that GitHub, for almost the first time, allows these collective dispositions, expressed in confused and transient pluralities, to be tabulated.

Conclusion

Code deeply shapes infrastructures, devices, services, protocols and many mundane capillary orderings in cities. Or does it? Might not code itself be patterned by the urban, by dynamics and transients, by diagrams and vectors that code itself expresses and enunciates without fully controlling? I have been suggesting that any response to such questions has to grapple with the changing character of code traffic. That traffic has many different facets, some of which can be architecturally reorganised and enclosed, others of which remain subtly diffuse and imprecise. The simple contrast between `git` traffic and its socially coded concentration on GitHub is a useful guiding thread.

There are many other dynamics on GitHub that we might analyse in terms of coadaptation of imitative fluxes. The very crude tracking of imitation based on forking and repository names could be refined in various ways. We could ask, for instance, what are the most-forked repositories, and how do changes in copying practices help us think about different directions of movement of code and coders. I have mentioned the profusion of `dotfile` repositories in the last few years. These repositories can be analysed in terms of micro-gestural and micro-perceptual differentiations at work in the writing of code. Choices of colour, font size, line separation, shortcuts for keystroke commands and the multiplicity of configurations for code development could be used to develop a much richer account of how people move through code, almost like a 'gait analysis' for code. Similarly, the metrics of the name space could, at a very different end of infrastructural dimensioning of code help us see how, for instance, the contemporary rescaling of computational infrastructures through 'Cloud' computing or virtualisation ripple across code-as-crowd. In all of these settings, the mergers, coalescences, branching and replication of bodies of code suggest that there is no single operational level at which code governs cities.

Regardless of these possible directions of analysis, the broader point here is that software today is less like a machine, a system or even an assemblage, and more like a crowd. That is, it has a fluxing, flowing and somewhat disordered existence that generates powerful flashes and movements, that creates atmospherics and densely woven patches of order, but remains unstable and dynamic.

Notes

1 Thrift and French's empirical response to their own question begins with the Y2K bug, and the long lists of software potentially affected by it: keypad locks, pagers, solar panels, smoke detectors, camcorders, video cassette recorders, elevators. Although these lists now look dated, when a similar listing would include many things that did not exist in 2002, Thrift and French's (2002) description of the effect of software development on urban space remains recognisable: 'We will exist in a broadband world in which the Internet will be a permanently available 'Cloud' of information able to be called up through a number of appliances scattered through the environment. These appliances will be something more than portals for information. Rather, like many other devices which will not have Internet connections, they will be "practice-aware"' (315) and 'will, through a process of cultural absorption into practices, sink down from the representational into the non-representational world, so becoming a part of a taken-for-granted set of passions and skills' (318). The fact that these developments more than a decade later are still very much in train suggests that there is something quite predictable about the development of software and coding in organising urban life and spaces.

2 Writing in 2014, Thrift again addressed coded cities: 'Take just the case of coded cities understood as a whole' (Thrift 2014: 13), and then proceeded to offer a sixfold topography of the code city – as externalisation of capitalist power, as prescribed matter of concern, as care-laden responses to the demand for resilience, as projected-retrojected dream life, as navigational geometry and as materialised visualisation. He finally suggested that something links this diversity: the possibility for 'these entities to learn […] to transform themselves'. Echoing the 2002 discussion of the geography of indeterminacy, he attributed this possibility to 'emergent tendencies arising out of complexity' or 'through simple happenstance which places them in unexpected situations which require adaptation' (13). The sixfold evocation of the coded city somewhat complicates the geographies of software in the city, but it reiterates the transformative capacity of indeterminacy. The 2002 formulations on indeterminacy and 'traffic between beings' grow into forms of novel encounter in 'unexpected situations'. (While urban sociology has long understood cities in terms of encounters between strangers, Thrift's account shifts the emphasis to unexpected encounters between other beings.) Coded cities' 'capacity to learn' transpires, according to this account, in elementary forms of movement understood as *code traffic.* If this is the case, the 'authoring', the 'learning' and the transformations should not only be traceable in code, but coding itself matters greatly as a process where externalisations, matters of concern, geometries, projections, visualisations, resilience-care, etc., come together and affect each other. The volume and composition of the traffic of coding itself as a cycling and recycling might be an important trace of more general transformations.

3 This, I should note, is a departure from most crowd theory, crowd psychology and crowd sociology. In most crowd theory, things hardly figure at all. As I will sketch, points of identification occur between systems, platforms, protocols and patterns just as much as between individuals or groups.

4 The GitHub team takes a strong interest in bootstrap, a set of components such as buttons, forms, progress bars, tables, typographic elements and colour themes for Web front end development. These visual elements figure heavily in the visual appearance of GitHub as a social media platform; hence, bootstrap already matters to the social coding of code that GitHub itself does.
5 The only problem with this is that the GitHub co-founder, Tom Preston-Werner (aka mojombo, id = '1' in the GitHub user list – in other words, the first hubbernaut ever) has recently had to step down as CEO after much publicised allegations of sexist and discriminatory language and behaviour in the workplace (Newman 2014). I largely leave the workplace dynamics of GitHub aside here, but they are symptomatic.
6 This geography refers to the 'hubbernauts' themselves, not to GitHub users, who are much more dispersed.
7 We can see some of what the GitHub team has been doing in GitHub repositories by running queries against the GitHub API (application programmer interface) or using the archived datastream of GitHub activity at GithubArchive.org. All of the numerical data in this paper result from queries run against the GithubArchive.org record of GitHub traffic. In the case of Figure 6.1, the query process went: find all the public repositories on which the named 'actor' works, and count the different actions they perform on those repositories. If we run this query for all hubbernauts (233 at the time of writing), as well as mojombo, something of the network of work done on GitHub itself begins to appear. I am assuming that the coding of GitHub can be seen in git traffic.
8 Tracking imitative fluxes means engaging with things that inherently lack any full formatting or clear outline. I focus here on the names of repositories and the names of actors. I have argued elsewhere that naming practices and code name spaces offer a rich resource for thinking about recursive and imitative processes in software culture (Mackenzie 2014).

References

Borch, C. (2005) 'Urban imitations: Tarde's sociology revisited', *Theory Culture & Society* 22(3), 81–100.

Guattari, F. and Deleuze, G. (1988) *A Thousand Plateaus: Capitalism and Schizophrenia*, London: Athlone.

Horvath, R.J. (1974) 'Machine Space', *Geographical Review*, 64(2): 167–88.

Mackenzie, A. (2014) 'useR!: aggression, alterity and unbound affects in statistical programming', in O. Goriunova (ed.) *Fun and Software: Exploring Pleasure, Paradox and Pain in Computing*, New York: Bloomsbury Academic.

Newman, L.H. (2014) 'The results of GitHub's harassment investigation are vague', *Slate*, 21 April 2014, available from http://www.slate.com/blogs/future_tense/2014/04/21/github_ceo_tom_preston_werner_is_cleared_but_still_resigns_over_harassment.html [accessed 24 August 2015].

Tarde, G. (1902) *Psychologie Économique*, Paris: F. Alcan.

Thrift, N. (2014) 'The "sentient" city and what it may portend', *Big Data and Society*, 1(1): 1–21.

Thrift, N. and French, S. (2002) 'The automatic production of space', *Transactions of the Institute of British Geographers*, 27(3): 309–35.