What Lies in the Path of the Revolution

Antranig Basman
Raising the Floor - International
amb26@ponder.org.uk

Abstract

Increasingly, the rights and capabilities to own technological artefacts, where they exist at all, are reserved to corporations and not to citizens. There are historical, economic, metaphysical, ideological and cognitive reasons for this situation, in addition to purely technological factors, which we will trace by following the fate of various concrete examples, analysed into five categories of ownable elements. These categories are those of ownable function, ownable expression, ownable data, ownable installations and ownable economies. In this paper, we attempt to align these goals of ownability into a research and activism program by describing a set of revolutionary goals in each category, and tracing ways we could reach them.

1. Introduction

One of the principal effects and aims of the 1789 French Revolution was the dismantling of France's feudal system, under which some citizens (nobles, officers of the church, etc.) enjoyed special rights, protections and income, while others (serfs, peasants) faced special taxes and in some cases were prohibited from holding personal property at all. We argue that a form of "digital serfdom" has rapidly grown up around us, where various important classes of artefacts, increasingly essential for everyday life, including participation in political and economic life, cannot be effectively owned by ordinary individuals. The power to own these artefacts is increasingly reserved only to corporations, which as a result of legal frameworks arising from the end of the 19th century, especially in the Anglo-Saxon nations, are considered legally as persons.

However, the barriers to ownership are not merely legal, but cognitive, economic, technological and even metaphysical.

Many of the technological barriers to ownership, especially as regards software, can be seen as embedded in certain questions of "reuse" — one of the central affordances of ownership is the ability to transplant a thing from its original location to a different one, following the desires or person of the owner. For most kinds of software this is possible in only a crude way — an "application" can be installed on one machine rather than another — and with the rising prevalence of rental or cloud-based models for the deployment of software, it is decreasingly possible at all.

In this paper, we use "ownership" to refer to a set of related use patterns where owners can pursue alternatives, repurpose and adapt their property, or maintain their property as the surrounding world changes. We will survey the kinds of things that we might want to own and how we could set about transforming our environment so that they could be owned.

In the rest of this paper, we step through examples of different technological artefacts occurring in the ecosystem encompassing operation, installation, and development of software: we describe actual interactive interfaces and tools as artefacts of *function*, the underlying source code as artefacts of *expression*, the content manipulated through software as *data*, the machines and networks running software as *installations*, and the means for organising labour around software as *economies*. For each of these, we describe examples of hypothetical ownable use, and then analyse why enacting those examples is today impractical or impossible.

In terms of aligning with a historical trajectory, we should note that even if we envision a socialistic far

future, establishing the means for personal property and thus ending feudalism is a basic requirement to even operate capitalism. Marx did not even propose to operate communism without a basis for personal property[1].

It would be supremely easy to make our point about poor ownership values by picking on the output of one of today's major corporations. For example, we could refer to recent versions of Adobe's "Creative Suite" which is now no longer ownable even nominally, since these are exclusively available through leasing models. Or we could refer to Microsoft's or Google's apparently "free" suite of office tools which are readily available in the cloud, but are only ownable on the basis of offering our personal data or our eyeballs to advertisements in exchange for them — these services may be unilaterally withdrawn without recourse at any moment[2], and are the touchstone of today's digital serfdom.

We will take such problems as read — we are not seeking to restrict choices of operating models by corporations, but instead to increase choices that are even in theory available to citizens.

2. Ownable Function

If you find a piece of software that does something of value to you, it should be possible to make it your own. This implies that you can keep using it as part of the collection of tools you carry with you, in familiar contexts, or experiment with using it in novel contexts. In this respect it would be like a tool that any workman or craftsman applies to their work. We'll illustrate the kind of thing we mean, with respect to the software of today, firstly with a small-scale example, and then a larger-scale example.

2.1. A small-scale example: colour picker

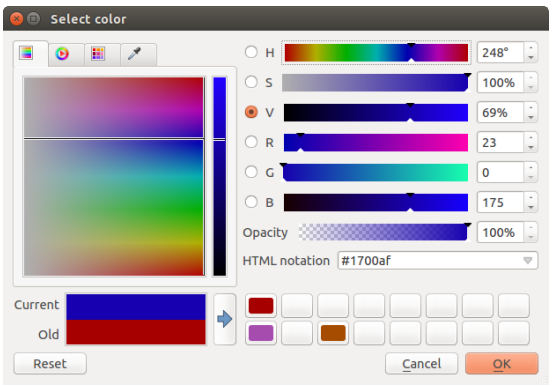

Figure 1 – One community's colour picker

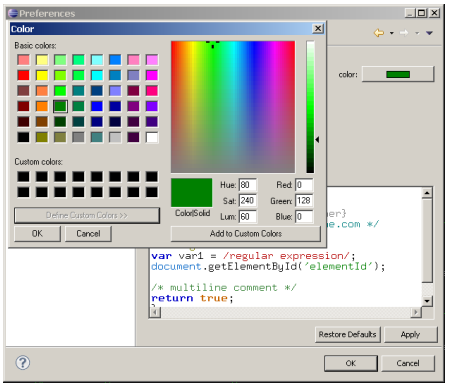

Figure 2 – The Windows colour picker

Figure 1 shows a colour picking tool designed by a particular community[3]. It is a competent if unexciting example of its breed. Imagine that you had got particularly used to the layout, idiom and affordances of this colour picking tool and wanted it to be your colour picker of choice in some other context, such as formatting a text document, and wanted to know how to achieve this. People unfamiliar with the nature of today's software ecologies would be deeply surprised to learn that the answer to this question is that it is economically impossible. For comparison in Figure 2 is the standard Windows colour picker, as being invoked from a popular code editor, Eclipse, which is written in the Java language. It is visually clear that they perform the same basic function, and that they are even capable of sharing the same

[1] "We by no means intend to abolish this personal appropriation of the products of labour, an appropriation that is made for the maintenance and reproduction of human life, and that leaves no surplus wherewith to command the labour of others." (Marx and Engels, 1848)

[2] As numerous of Google's products have been in the past: `https://en.wikipedia.org/wiki/Category:Discontinued_Google_services`

[3] An open source geographical information system named QGIS

representations for the colour value[4]. What could be the source of the difficulty in dropping in one in the place of the other?

A naive user expecting the colour picker to be an ownable artefact might discover that it cannot pick or drop colours outside the window of the host application. We could imagine them closing the host application expecting the colour picker widget to stick around, instead of disappearing. In practice, these kinds of problems require users to carefully mediate between applications, e.g., using the host application of the colour picker as a transit station where colours to be sampled are imported by screenshotting, and exported by copy-pasting e.g. the HTML notation for the colour.

These barriers to ownable use originate in development choices outside even the expert owner's control. Our example colour picker originates in an application, whereas a luckier user might have access to and prefer a colour picker hosted at the operating system level, which may travel between different applications. On a deeper level, these problems are embedded in the stack of technologies used to implement and run the colour picker, which we return to in section 3.1.

The economic reuse value of this widget across this boundary is zero, a poor compensation for the hundreds of hours of expensive developer time that went into producing it. As we outlined in our introduction, there is a close relationship between the faculty of ownable function, and the faculty called "reuse", which software developers talk about a lot. In fact, one faculty is a subset of the other — while "reuse" is a phenomenon which may be encountered when swapping out elements of a system which are in direct contact, ownability also brings into consideration elements of the system that form more distant context.

2.2. A large-scale example — iNaturalist

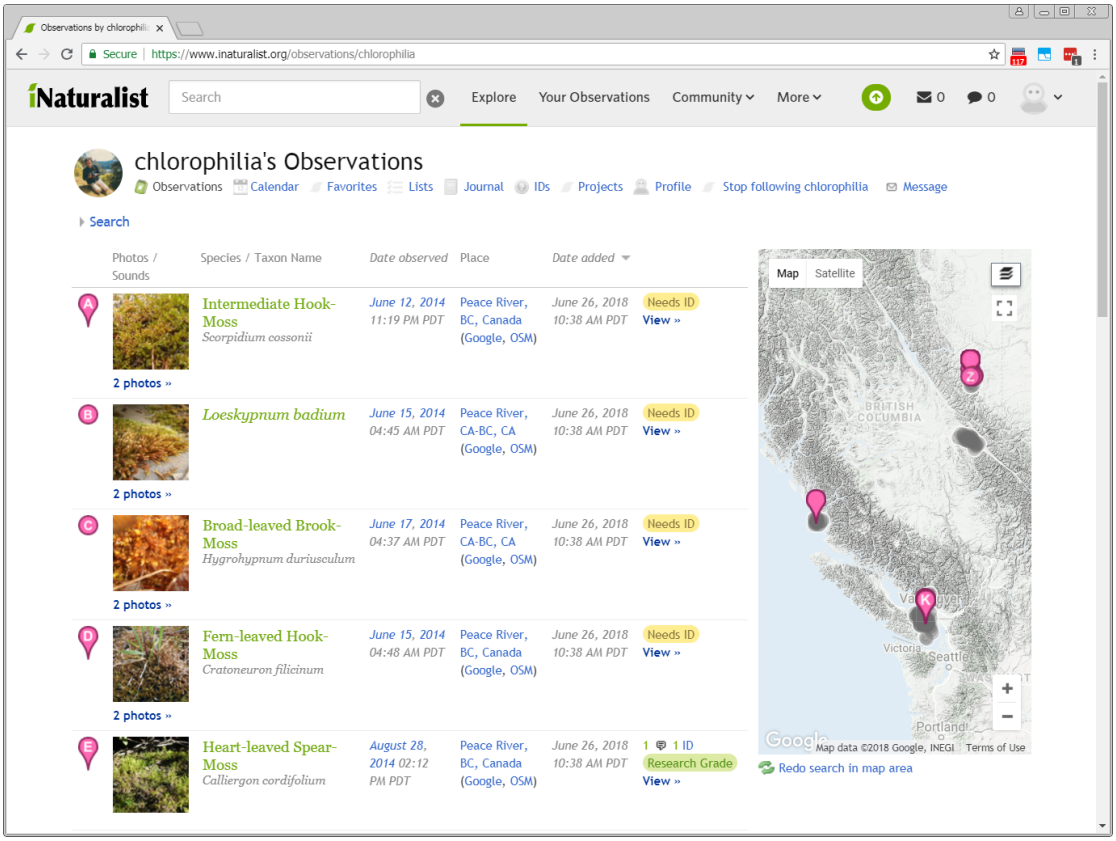

Figure 3 – A view of one contributor's observations on iNaturalist

[4]The range of realistically useful alternative colour tools is in fact much larger. Jalal et al. (2015) study how expert designers invent new mechanisms and contexts to manipulate colour, and argue that digital colour manipulation has been wrongly reduced to a simple problem of selecting and storing colours in contextless palettes.

Our chosen large-scale example is iNaturalist[5], a platform which is free for use by any citizens wanting to contribute their observations of living things, which are categorised by species, confirmed by expert enthusiasts, and plotted on a geographical database. This is an open source application written mostly in the Ruby language, with the significant admixture of JavaScript essential to any modern webapp. Figure 3 shows a typical view of this web application in use — one expert contributor is viewing the first page of their own contributed observations to the system. We will adopt this application as a "running example" through our succeeding sections on ownable expressions, data and installations.

iNaturalist is an application developed to serve the public good, whose source code is readily available, well-maintained and has a welcoming community. This should offer us a radically different perspective on ownability compared to commercial closed-source software such as the Adobe Creative Suite, but as we will discover, attempts at ownable use have very much the same results for the citizen: ownability doesn't come at a price they can afford.

In our example, an expert user of iNaturalist wishes to adapt this existing, rich platform to assist their own projects by making a number of additions to its functionality and interface. This example is sourced from a real expert contributor. The expert has a general familiarity with the popular statistical analysis packages which are used in their own field, and an everyday acquaintance with web technologies, but is certainly far from the skill-set of a professional developer. We will survey in the following sections how the economics currently play out for the assistance that our field can currently offer them, and then lay plans for how a revolutionary uprooting of its metaphysics could lead to a better story. We'll also survey a few early "green shoots" of existing starts towards better authorial values and consider how they could be promoted and supplemented with other approaches to eventually form an integrated forest of ownable technology.

2.2.1. Planned Adaptations

Our expert wants to adapt the iNaturalist platform in (at least) the following ways:

- To be able to contribute lists of species that are considered to be available in particular locations, and to associate those lists with geographical boundaries drawn in a free-form way on maps
- In addition to these lists being curated in an on-going way by assigned "project managers", each list will contain fields in addition to those maintained by the iNaturalist platform — e.g. the natural "habit" of a species, whether it is bush-like, tree-like, etc.
- While the core iNaturalist interface allowing the contribution of new observations and viewing existing ones should remain in place, certain new views should be available — for example, to select two or more lists and visualise their intersection or difference in some compelling graphical terms, e.g. a sunburst diagram

while many more examples of interesting adaptations could be listed, the above are sufficient to stress all the issues of ownability we are interested in exploring here.

The core ownability issues relate to how shared function is to be shared, how it is to be developed and tested, and at what cost, and how it is to be deployed. The expert's need for "ownable function" in this case goes beyond those of a craftsman merely contemplating a single "ownable hammer" to put in their workshop. In this case we are imagining function shared between overlapping communities, in the structure that Suchman (2002) describes as "multiple, located, partial perspectives". Our expert wants to plan for related or unrelated communities structured around other experts, each with their own perspectives on what species (there) are, and what aspects of them are relevant to their communities. These all need to coexist with the "base function" offered by the core iNaturalist platform itself, and its base community. How this is done in practice has implications for the expressions of the authors assisting the expert, the expressions of the core community, the data they work on, and how the resulting networks of cooperating, actually installed software are constructed, which we will consider in succeeding sections.

[5] `https://www.inaturalist.org`

3. Ownable Expression

It should be possible to own the means to create and modify software, which we currently call programs. At a small scale software is built up out of collections of authorial expressions. These expressions describe the kinds of things that can exist, command particular selections of these things to actually exist or not exist, or change their properties while they do exist.

For these expressions to be ownable, all authors should have the ability to freely contribute their expressions to the work of others, and freely "buy into" and "buy out of" the expressions of others. We have formulated this requirement as the *Open Authorial Principle*: "Any expression by one author can have its effect replaced by an *additional* expression by a further author" (Basman et al., 2018). When we say that this "should be possible", we imply that it should be possible in practice — that is, that the authors' expressions can be successful at a cost which is affordable. This is a statement as much about economics as it is about engineering. These costs can be measured in numerous terms, for example, investment in a particular skill-set, direct monetary terms, effort in building a relationship with a particular community, or cognitive effort.

3.1. Small-scale example: colour picker

We return to our colour picker example from section 2.1 and the alternative in Figure 1, which a user wishes to reuse instead of the standard choice (Figure 2) that is brought up by an interaction in their application.

Given that our user is or has recruited the assistance of a developer, the first road block to performing this adaptation is to penetrate to the inside of a delivered application to locate the colour picker to-be-substituted. Second, the developer must reconcile the form of the migrating colour picker with the expectations of its new environment. Added to the difficulties created by the mismatch in implementation languages (C++ on the left, and a sandwich of Java and C on the right), these come together with different idioms for interpreting the contents of memory, and even for addressing and painting areas on the screen.

We note here that while some heroic efforts might succeed in transplanting this harmless-looking colour widget, the resulting assembly would be little more than a curiosity — the hapless user would effectively have taken on the responsibility for maintaining their own personally customised version of the entire application.

To add insult to injury, the techniques for implementing and delivering software create impenetrable boundaries from the point of view of the user as well as the developer, as we described in section 2.1.

Sometimes designers explicitly plan for this kind of reuse, for example by creating "plugin" architectures where a particular well-characterised part of a system is designed to be replaceable by alternatives. The converse is a "component" architecture where the one part of a system is free to appear in many different locations. Biddle and Tempero (1998) name these two kinds of reuse as "context reuse" and "component reuse" respectively. Where the designer has not laid plans of these kinds, developers, integrators, and users are out of luck — especially where language or platform barriers make reuse impossible. Even in the case where these kinds of reuse are planned for, they usually result in a strongly bounded ownability where creators and users of plugins commit economically to one particular environment over others.

The fact that the user has to be troubled by these irrelevant considerations of platform betray that we are in a pre-industrial environment plagued by irregularities like screws that are not of a size to match screw threads, trains that cannot run on each other's tracks, or electricity that may not travel between different wire networks (David and Bunn, 1988).

The ultimate implication of these issues is that programming languages as they are currently constituted need to be abolished, through a process of being dissolved into an "integration domain", a construct described by Kell (2009). An integration domain is a space in which one aligns multiple artefacts to co-operate, and which, importantly, is the only space that may contain assumptions about the particulars

of several artefacts. This implies that artefacts to be integrated expose structured representations of themselves (Clark and Basman, 2017).

In this example, the integration domain should allow a developer to substitute the existing expressions that connect the colour picker in Figure 2 to its operating context for a set of similar expressions mapping to the picker in Figure 1. Tasks for an even more ambitious integration domain would be to bridge these kinds of divides across multiple, separated but cooperating devices rather than just across process, language and integration boundaries within a single machine.

3.2. Large-scale example: iNaturalist

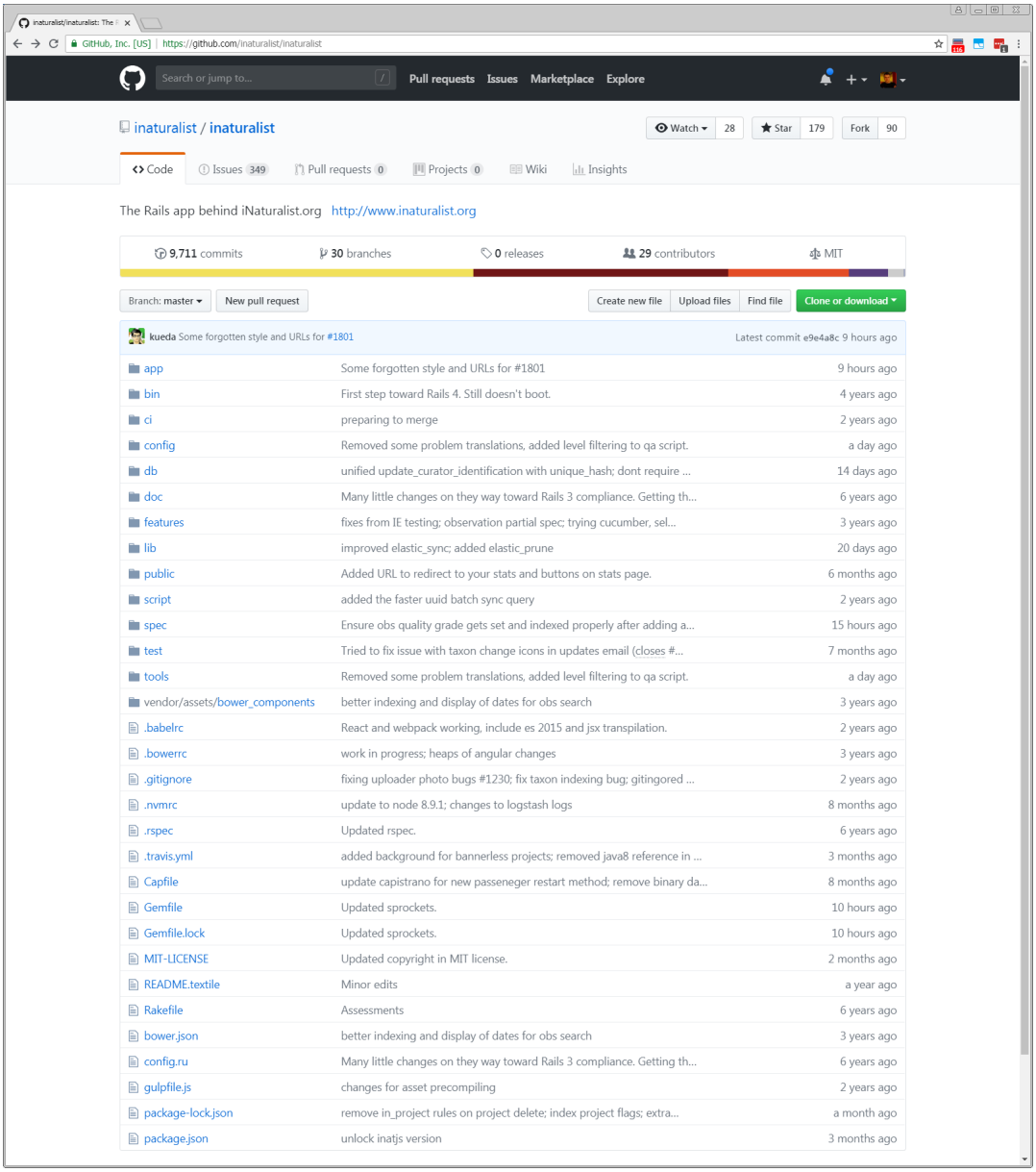

Figure 4 – The source code for iNaturalist in GitHub

The hypothetical process of adapting iNaturalist follows a similar procedure to the previous example: determining the sites where changes are to be made, then reconciling existing expressions with the new features we want to add.

3.2.1. Correspondence between expression and function

In Figure 4, we show the root page of iNaturalist's source code, the primary representation of its authorial expressions, as shown in GitHub, a very widely used tool for developers to share and manage their

code. The first thing to note is the total lack of correspondence between this view and the view of the application in use shown in Figure 3. Again, those unfamiliar with the choices made by those who designed our programming idioms might be deeply surprised to learn that this isn't just a superficial phenomenon — in fact, there is no reliable way, even with the most powerful analysis tools available, to determine for example given the view of the "Intermediate Hook-Moss" observation in Figure 3, which part of the source code in one of the numerous deeply nested directories shown in Figure 4 is responsible for it. This is nothing to do with the particular technology choices made by the developers of the application — it is instead a fault in our entire paradigm of software construction. The particular choice, of implementing a Ruby on Rails app with a fairly rich JavaScript front end, represents a mainstream, conservative choice of a well-supported and generally well-liked technology stack, but any other credible technology choice would have led to just the same result.

On the other hand, those who are indeed experienced in software construction would find this point so obvious as to take it completely for granted. Anyone inducted into the Eleusinian mysteries of code takes on board as implicit knowledge that they will have to acquire a broad spectrum of "jungle skills" which help them navigate these kinds of informational swamps — not least, an in-depth study of the structuring conventions used by the particular technology in question, but more broadly applicable techniques such as deliberately banging on things to see what breaks, studying the structure of long stack traces, or searching across the codebase for the occurrence of tell-tale strings.

In (Basman, 2017) (section 6.3) we present an alternative idiom for building software which would substantially bridge this kind of problem. Rather than building software with opaque "machines" such as function calls and objects, we would instead use reversible elements named "lenses" to express and enforce relationships between their parts. These would act both "vertically", enabling any part of the user interface of the application to be traced back to its source expressions, and "horizontally" in order to structure data flow around the application itself. There exists a mature community (Bx, 2018) devoted to the study of such reversible elements under the name of "bidirectional programming" but its activities are confined to establishing proofs of esoteric theoretical properties of such elements rather than constructing practical alternatives to general-purpose application development.

3.2.2. Sharing expressions between communities

In order to effectively share the expressions leading to the kinds of adaptations that our expert desires in section 2.2.1, we incur difficulties in other areas of ownership as well. While we will specifically treat problems of shared installation in section 5, some of these will be incurred during the process of developing shared expression and design and so are addressed here.

Firstly, we would like to consider how the expressions (source code) implementing the adaptations will coexist with the ones shown in figure 4. This involves social, design as well as technical and economic issues. To start with, these are adaptations which may not be appreciated by all users of the platform. A typical design solution to problems like this is the kind of "giant ribbon" familiar to all from modern office applications, where every conceivable function is packed into a crowded menu or ribbon widget, each represented by a very tiny icon in the hope that the presence of irrelevant functions is as small a distraction as possible. Some of the forces behind such designs are the difficulties in arranging for the expressions forming the adaptations to act in a suitably context-aware way, presenting themselves to some communities and not others.

Setting aside this design issue, we have a meta-design issue, in that we couldn't expect the adaptation to instantly appear fully-blown in its final form. More realistically, we expect an extended process of co-design, where competing alternatives can be evaluated together with the communities of interest. While simple prototypes and mockups might be suitable for early stages of this process, it is inevitable that the later stages of co-design will have to centre around patterns of real use of a functioning system with real data. The question is — what system, based on what expressions, operating on what data, installed and maintained by whom?

The standard answers offered to these kinds of questions in communities which can currently be sup-

ported rely on some kind of "centrism". The community wanting to make some adaptations must start to build a relationship with the core community. Modern distributed source control systems such as git reduce the formality of the early stages of this considerably. Since this is an open source project, the adapting community can simply "fork" the source code shown in Figure 4 and start modifying it. Beyond this point, however, the costs start to escalate rapidly, along two dimensions:

- Unless the adapting community wants to commit indefinitely to maintaining their own adapted version of the artefact, they will need to build a strong relationship with the upstream community, and become familiar with their coding guidelines, administrative practices, and ensure that their adaptation is structured in terms that they are prepared to accept and maintain
- During the process of design, they will need to become familiar with the deployment and integration requirements for running their own installation, as well as considering what data sharing idiom is appropriate between their installation and upstream.

These profound problems of design and expression, which should seemingly be our most urgent ones, are not considered interesting problems by academia, and are not considered profitable to tackle by corporations. They relate to what could be called a "commons of the mind" and fall to whatever resources society as a whole has to devote to such issues, which are minimal. The means by which these problems are solved in practice can be compared to Suchman's "articulation work", the often invisible work to actually make technologies work together inside a community of practice full of existing artefacts (2002).

Where they relate to problems of expression itself, we trace these problems back through the intellectual history and culture of the computer science and programming communities in (Basman, 2017) and (Clark and Basman, 2017).

4. Ownable Data

As with the situation with "ownable expression", there is both a formal and an ultimate aspect to the ownership of data. Firstly there is the possibility of physically laying ones hands on the bits involved, at an acceptable price, or for free. This formal sense of "open data" is widely acknowledged (although not universally practiced), and has a parallel role to that of "open source" in the category of ownable expression — it is the most basic "entry ticket" to ownability but doesn't necessarily get one very far into the venue.

The more ultimate aspect of ownable data is whether the data can in practice be put to the effective purpose its owner requires. This may include

- Whether it can be transmitted to or from a place of use in a format that is intelligible at both sites
- Whether it can be reconciled with data from other authorities which may use different standards for provenance, different ontologies, different schedules and policies for validation, or have a mismatched scope
- Whether legal or intellectual property issues effectively prohibit the data being put to the owner's purpose

Again, as with ownable expression, this is an economic proposition, rather than a formal or logical one. These capabilities must be delivered at a cost that the owner can afford.

As a corporate example, Google is slightly ahead of some rivals in providing formally open data. Google in theory provides the facility to export your data from its services using "Google Takeout", a product of the "Google Data Liberation Front". Facebook also allows you to download some proportion of the data it holds on you. However, given this data, it is not always clear how its value may be liberated:

- Some services such as YouTube and Facebook have ended up as natural monopolies, and so there is no other equivalent platform on which one's data may be used
- The damage may already have been done to its capacity for use by having been incorporated under the service provider's licencing conditions.

Balkan (2017), further, notes that in Facebook's case, the data in question is *about* the citizens we are planning to empower with ownership. In this case, it is yet more crucial to empower these citizens to express and enforce the boundaries of ownership where the material constitutes the extension of themselves into a digital realm. He calls for an infrastructure to achieve this which is funded from and owned by the commons.

4.1. Ownable Data Curation — The Problem from the Front End

Let's return to our running example of adapting iNaturalist, which has many interesting connections with ownable data.

Firstly, at the level of the expert himself, the adaptation we plan in section 2.2.1 requires the entry and maintenance of the species lists. In practice, this is something that the expert already does for themself in a tool of their own convenience. In this case, the tool is the Macintosh-specific spreadsheet named "Numbers". The standard workflow that is available for this kind of situation is to ask the expert to manually operate some kind of CSV export functionality within the application, to upload this file to the installation server, to there operate some format conversion as part of a build or ingest process, etc.

This clunky workflow is obviously hugely unsatisfactory. The adaptations to iNaturalist should include an interface for the expert (and other users sharing the adaptations) to enter and manage data such as the species lists live. This presents quite a few problems, and a range of choices none of which offer favorable economics in today's ownability landscape.

One approach might be to embed some suitable data management UI and persistence mechanism within our designed adaptation, and have the expert use that. One might think that freeform "spreadsheet-like" interfaces for the management of arbitrary tabular data could now be picked off the shelf in a variety of competent open source incarnations, with adaptable styling, interaction idiom and layout, ready for embedding in any online interface of choice — given the basic interaction idiom was already mostly perfected in the 80s. However, the fundamental difficulties in developing ownable function and expression that we described in the previous sections have prevented the democratisation of this capability as well.

In addition, there are further difficulties raised by available algorithms for collaboratively working on shared data, for example under the heading of operational transforms (OT[6]), "Conflict-Free Replicated Data Types" (CRDT), etc. These are hard to implement in an ownable way as a result of their great complexity, and proceed from the faulty assumption that that there is always a magic "right answer" to every question of conflicting updates that never involves any further user interaction and might involve losing one user's data.

Some work on a much more straightforward solution to a restricted class of this problem based on modelling practical intentions of data owners appears in (Haverbeke, 2015), although the complexity still isn't negligible, and the algorithm is optimised for document-structured rather than tabular data.

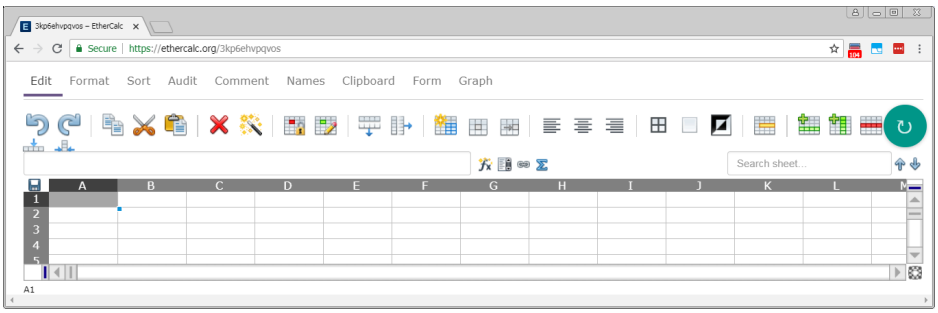

Figure 5 – EtherCalc's rather bulky interface for collaboratively editing tabular data

[6] https://en.wikipedia.org/wiki/Operational_transformation

One notable open source and freely hosted collaborative spreadsheet editor is EtherCalc[7], whose interface is shown in Figure 5. However, this is an extremely heavyweight product actually of substantially greater complexity than iNaturalist itself. Since it has been built with a standard software development idiom, it is the product of an unownable software tradition, it will be prohibitive to adapt it to lightweight use embedded within an interface rendered by another application written in a different set of technologies, such as iNaturalist — which, naturally, suffers exactly the same problem in return. EtherCalc is simply "a collaborative spreadsheet application running on the web" and offers no useful user-directed dismantling of the expressions which led to its creation — although of course it offers the standard dismantling of its expressions into insulated implementation modules, each performing a specialised task of meaning only to developers, as is considered the standard virtuous behaviour in a traditional community composed of developers.

Faced with this uncompromising landscape, our expert currently falls back on embedding views of their data using the flagship tool of the corporate oppressors *du jour*, Google Sheets. This platform also offers some support for querying data and establishing live connections via a reasonably standardised API, although of course offers no ownable installation values at all — as with every Google product, it may one day disappear or change without recourse. It also has a lack of facilities for naming, archiving and ensuring the integrity of particular versions of the data it manages, which is a problem we will look at in the next section.

Another possibility might be to use whatever expression-hosting capacity is present in the expert's client tool, Numbers, to write some form of custom "macro" that periodically synchronises its data with some ownable server in the cloud. However, it was designed without such democratic aims in mind, and in any case would not solve the curation and versioning problems that we now turn to.

4.2. Ownable Data Curation — The Problem from the Back End

Widening our focus from our iNaturalist example for this section, let's pass on from looking at the problem from the front-end perspective, and look at what supports there exist for building networks of cooperating curators of live, version-managed open data. One important organisation in this area is the Open Knowledge Labs[8] which have an initiative known as Frictionless Data[9]. It tackles the problem of ingesting data from various popular housings (including the basic CSV files we recommend our expert to produce using our "clunky workflow" of section 4.1), and packaging it with sufficient metadata as a "Data Package" that may be safely forwarded into a variety of standard kinds of repository, including the git repositories we saw in section 3 as well as JavaScript's standard NPM registry, etc. This will certainly be sufficient for the modest-sized datasets that our expert is likely to generate directly in a UI, although this idiom may fall down for extremely large or continuously updated datasets such as sensor readings. In any case, it is a valuable "next hop" for our data even if, as we explored in the previous section, we lack any particularly competent user interface that would help our expert get the data in.

4.3. Back End Ownable Data for iNaturalist

The iNaturalist community already has substantial experience with data federation. As well as having several active localised downstream communities managing separate iNaturalist installations for Canada, Colombia and other nations, it also has several upstream communities, for example the Global Biodiversity Information Facility (GBIF) with which it shares observation data, and the Catalogue of Life, with which it shares taxonomic data. However, this federation has required significant maturity in the related communities and still happens on a somewhat ad-hoc, batched basis (currently updates are sent every week to GBIF) through specialised formats and protocols designed just for these communities. For example, Figure 6 shows part of the output of a JSON data feed from the Catalogue of Life, its taxonomic data held on the Lesser Black-Backed Gull *Larus Fuscus*.

Towards the bottom of the figure, you can make out the beginnings of a taxonomic tree classifying the

[7] `https://ethercalc.net/`

[8] `http://okfnlabs.org/`

[9] `http://okfnlabs.org/projects/frictionless-data/`

species in question. This kind of data is central in interpreting the "lists" that our expert and their peers may manage; they may subscribe to the classification in this catalogue, or another, or instead have their own opinion on how it should be organised. To support these activities there should be a choice of usable off-the-shelf means to subscribe to and submit amendments to data in such feeds, and the feeds themselves need to be organised to permit this.

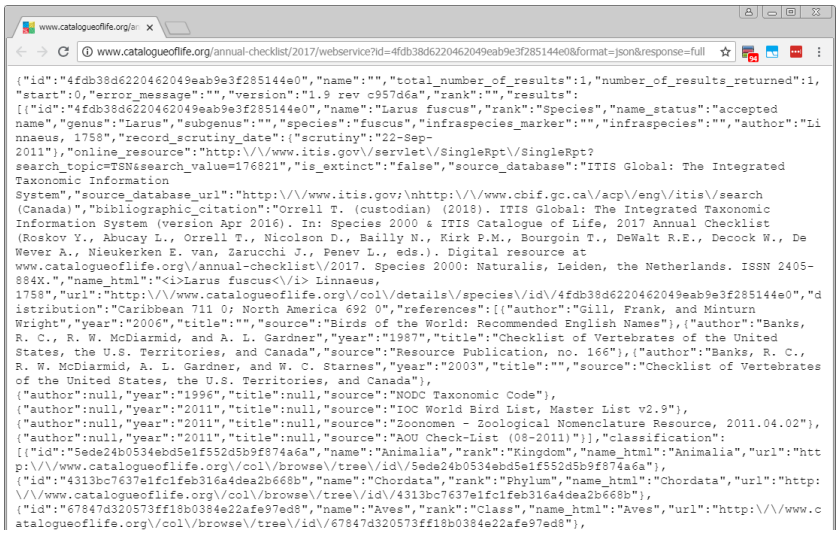

Figure 6 – A JSON data feed from the Catalogue of Life

What is our subject-matter expert to make of output such as that in Figure 6? Their enterprising nature tends to mean they can make a fair amount of it, but there are obviously many hurdles to be crossed in turning such "formally open" data into something of effective use. To start with, it helps simply that, rather than being notated in some old-fashioned, bulky encoding such as XML, it is made available in JSON, the natural format for the universal web language, JavaScript — from here it can be directly manipulated by popular data visualisation packages such as D3, Vega, etc., as well as the primitives of JavaScript itself. Some might contend that this is a mere surface phenomenon, but diSessa (2000) is clear that even small incremental notational improvements over the centuries have been responsible for vast cumulative effects in promoting thought and innovation.

However, it is also clear that there is a huge scope here for supporting our expert in the great mass of "articulation work" (Suchman, 2002) needed to put this data to use. For a start, it is clear that this data is part of a wider referential whole — the document contains embedded references to numerous other related documents in an essentially opaque way. Secondly, such documents typically require significant reorganisation, filtering, and tidying before the elements of interest for a particular task can be brought into sharp focus. For these tasks we imagine employing again the "lenses" that we spoke of in section 3.2.1 which reversibly image such documents into the focused elements of interest. This "dialect of lenses" would itself naturally need to form a domain of "ownable expression" of the kind that we motivate in section 3.

There are many articulation annoyances in lining up the data in such feeds with our expert's needs, but in the next sections we'll consider more profound issues related to how these feeds are constructed in the first place.

4.3.1. Feeds are Not Lenses

Firstly, there is the fact that these feeds themselves don't constitute any kinds of lenses in themselves. Data is pushed out through them, but it is received back in by another, generally ad hoc process. Given the Catalogue of Life is a heavily curated dataset, this is no great surprise since any updates should be vetted by expert human eyeballs before being accepted. But nonetheless, even the process of *requesting* an update doesn't take the same form as the feed itself. To some extent this is a reflection of the fact that

such systems (including iNaturalist) are implemented with previous-century persistence technologies such as SQL, and so the process of assembling a feed as a "view" is naturally uninvertible. More modern systems such as the CouchDB family of databases that we refer to in the next section 5 on ownable installation have partial answers to how updates propagate between databases, but these are still framed as forming part of the implementation level rather than the user level of the data network.

4.3.2. Ownable, Public Version Management

Secondly, there is the issue of version management. Since the Catalogue of Life is a highly mature public dataset, it has a well-formalised approach to issuing validated versions of the data, by publishing validated Annual Checklists in addition to a constantly evolving Dynamic Checklist. However, this is an ad-hoc system designed for a particular community, with its details encoded into the URLs of the feeds in a corresponding ad-hoc way. It would not be possible for a user to trace the lineage of a particular piece of data to discover how and when it had been introduced, or what previous versions it appeared in, without a costly manual procedure. The process of data version management needs to be as ownable as anything else — with the possibility to "take away" entire historical lineages of data with their sequencing and provenance relationships intact.

4.4. Plans for Ownable Data

To collect together all of our threads from the above sections, there is clearly scope for a great deal of valuable work in this area, producing an ownable platform supporting the kinds of data entry, encoding, transmission and reorganisation tasks that we have toured above.

Many of the version management issues that we raise in section 4.3.2 have credible solutions in the elite tools which developers have produced for their own needs, such as the git system we refer to in section 3. As we mention in section 4.2, in combination with some packaging conventions derived from the "frictionless data" community, and the use of further conventions on structuring mutually referential islands of data encoded in JSON, this might form the basis for a powerful and highly usable data distribution and referencing system. Returning to our starting point in this section, this would then need to be provided with a highly ownable and embeddable user interface for manipulating the data held from the point of view of any subcommunity.

5. Ownable Installations

It should be possible to own the physical machines and networks on which the software of individuals and communities runs. Another name for this capability would be "ownable infrastructure" — referring to the entire chain of supports necessary to make a piece of software available in a particular time and place, from the hardware (virtualised or otherwise), the energy and network costs of running it, its connections to sources of data in its community and those of others, the act of installing it itself, paying any licence costs, and ensuring that it stays running stably and free of security vulnerabilities.

We don't necessarily mean that people should be able to run their own data centres, but this should at least be in theory possible if the ones commercially on offer operate unsuitable policies, e.g. with respect to data privacy, licencing, etc.

Ironically, for web applications, this is an area of ownability in which there has been significantly greater progress than any of the other classes of "ownability" that we consider. The last ten years has seen a vast proliferation of "Something as a Service" platforms by cloud computing vendors, where "Something" can variously be replaced by "Platform", "Infrastructure", "Software" or others. Each of these provide models where one may rent standardised parts of the infrastructure needed to host a web application, delegating more or less of the responsibility for it to the provider as needed. These rely on various pieces of orchestration technology, many of which have increasingly solid open source implementation, such as OpenStack and Kubernetes, which automate the processes of spinning up one or more machines of a particular configuration which cooperate to provide a particular function.

This is a highly promising development, although the ecosystem is still in turmoil and is very far from

the state where an ordinary citizen (or expert) could turn up and at the push of a button set up a fully configured incarnation of their own version of a complex platform such as iNaturalist.

One community attempting to meet these aims directly is named hood.ie[10]. This community is unusual in the technological ecosystem in being wholly oriented towards newcomers, supplying copious and invitingly-formatted documentation aiming to help them get off the ground as quickly as possible with designing and hosting applications which are equally capable of working with data in the cloud as well as locally. This is an open source project which makes use of numerous other open source products (CouchDB for persistence, node.js for running JavaScript on the server) to orchestrate together easy-to-use solutions for non-expert developers. While this is a highly promising development, because of the profound kinds of problems in software structuring we mentioned in our sections on ownable expression, hood.ie is only capable of expressing relatively simple designs convincingly — and once the designs scale beyond this horizon, they may well need reimplementing in more traditional technologies. This onramp is smooth with a shallow slope, but fairly short.

6. Ownable Economies

Citizens should also be able to own the institutions and mechanisms that organise their labour and its products, and which are responsible for supplying them the means of life.

Our running large-scale example, iNaturalist, is a less good fit to our discussion here because of the way it is funded and operated, but it is largely because it has been able to sidestep these issues that it has likely had the space to create such a vibrant and healthy set of communities. It began life as a grant-funded project, and while its current funding model is not clear, it enjoys such an obvious "public good" mission, and its running costs supporting the current rate of about 10 observations per minute are likely to remain modest given the vast commoditisation of cloud computing resources which we've seen over the last several years, although naturally this doesn't cover the much greater cost of maintenance, continued development, outreach and other content creation.

Other communities with a less obvious "public good" mission will have to bite the bullet of issues such as monetization, governance and how ownership of the platform and its technology actually enacts itself in their lives. Many so-called "disruptive" platforms have promised to disintermediate the process of organising labour, but the majority of them, such as Uber, AirBnB, and the like have been found to be punishingly extractive (Wachsmuth et al., 2018). Rather than vehicles for equity, these platforms merely function to channel value away from communities and centralise it in the pockets of distant founders and backers, frequently to be found in Silicon Valley. Other platforms such as YouTube monetize the time, attention and privacy of its users in ways which can be difficult to trace and understand, e.g. the automatically generated YouTube videos targetted at children described by Bridle (2017).

A growing movement of "co-operative platforms" seeks to find ways in which these means of organising labour and work can be truly owned by the participants. The authors are involved in a recently funded project to produce a "Platform Cooperative Development Kit"[11] to produce some freely ownable infrastructure and blueprints that can assist communities like these.

7. An ideal adaptation scenario

We now imagine ourselves in a world after the revolution, and consider how the communities surrounding our iNaturalist running example might ideally structure their work if we had achieved all of our aims of ownability.

Firstly, following ownable expression, there would be no need for the adapting community to fork the entire code of the core product at all. Part of the necessary substructure for a lens-based design as

[10] `http://hood.ie/`

[11] `https://www2.ocadu.ca/news/initiative-co-led-by-idrc-and-the-new-school-receives-1m-google-grant`

described in section 3.2.1 would be a *natural alignment* between not only the structure and function of the original design, but also variant designs. The adapting community's contribution could be structured as a completely self-contained project, and at the time of installation, overlaid onto it using the natural *program addition operator* that such an alignment enables — the nature and requirements of such an operator are described in (Basman et al., 2018). This is currently the most far-fetched of our results.

Secondly, following ownable function and installation, the adapting community would be able to freely make the choice between hosting their own iNaturalist, or attempting to get modifications introduced into the central installation or one of its federated peers.

Thirdly, following ownable data, the adapting community and central community would be able to freely choose amongst themselves the policy on whether updates issued from the adapting community would be fed back to the central community and on what schedule and granularity.

In our post-revolution world, every community would be able to choose from a ready-implemented standardised set of protocols, policies and formats for these purposes.

8. Conclusion

We've drawn up a set of blueprints for radical progress towards citizen ownership of technological products on multiple fronts. Of these, the ones that relate to the means of expressing and designing software are the most intractable and difficult to achieve, since they involve fundamental reframings of many ideas that have become entrenched in mathematical, metaphysical and computational dialogues for a substantial time. In addition, there is essentially no recognition in mainstream programming language and system design communities of the kind of far-reaching extirpation that will be required. Credible progress in this area will take many decades.

Our ownership aims for actual software installations are less distant. As we mentioned in section 5, there are sufficiently promising developments underway in the infrastructure communities that we could expect such systems to be at the disposal of everyday citizens in a decade or so. However, unless the problems of ownable function and expression were solved, with aligned solutions to ownable data, these systems would be of little use.

The issue of ownable data is also one on which there is a reasonable and growing recognition that there is a substantial problem to be solved, of significant benefit to society, but progress so far is only moderate. This recognition is centred in communities such as data journalism and governmental statistics, and there are even institutions such as the Open Data Institute and the Open Knowledge Labs pursuing a subset of these aims.

It is the fact, as our presentation of a running example has shown, that these aims are intimately interrelated and that lack of progress on any one will hold back the development of several others that makes incremental progress towards this revolution difficult. This is why we have concentrated here on sketching a crude picture of many areas at once, with a rough indication of their relationship, rather than trying at the futile endeavour of proving a prescription of ordered steps for reaching all these goals at the same time.

There is nothing to be derided in this approach, since as Lakatos (1978) points out, genuine incrementalism is impossible in true science in any case. In his analysis of Newton's theories of motion and gravitation, he points out that under these, as with any credible general theory, no one part of the real system being explained could ever be effectively separated from distracting effects in order to be an unambiguously validating or refuting phenomenon for the theory. Furthermore, he points out that any observation which is intended to incrementally support an improved theory must first be interpreted with respect to the theory it intends to refute — we must always 'progress on inconsistent foundations'.

History is replete with the transformative effects that ownable artefacts and infrastructures, such as the printing press, can enact on the largest scales in societies — to return to our revolutionary theme, without

the highly ownable printing presses housed in the Cordeliers district of Paris (Hesse, 1991, Chapter 5), the inflammatory pamphlets written by Danton, Desmoulins and other revolutionaries could not have begun their work at opposing the extractive power of the French state. We expect our own universal media, once they are endowed with all the properties that ensure their ownability by private citizens, to be capable of similar effects in levelling systemic inequities.

Acknowledgments

This paper emerged in large part from discussions of what revolutionary goals our community should seek with Luke Church, Colin Clark, Clayton Lewis, Mariana Mărășoiu, Tomas Petricek, Oli Sharpe, and others.

References

Aral Balkan. 2017. Encouraging individual sovereignty and a healthy commons. (2017). https://2018.ar.al/notes/encouraging-individual-sovereignty-and-a-healthy-commons/

Antranig Basman. 2017. If What We Made Were Real. In *Proceedings of the Psychology of Programming Interest Group*.

Antranig Basman, Clayton Lewis, and Colin Clark. 2018. The Open Authorial Principle: Supporting Networks of Authors in Creating Externalizable Designs. In *Submitted to Onward '18*. https://github.com/amb26/papers/blob/master/onward-2016/onward-2016.pdf

Robert Biddle and Ewan Tempero. 1998. Evaluating Design by Reusability. (1998). http://www.mcs.vuw.ac.nz/research/design1/1998/submissions/biddle/

James Bridle. 2017. Something is wrong on the internet. (2017). https://medium.com/@jamesbridle/something-is-wrong-on-the-internet-c39c471271d2

Bx. 2018. Bx 2018 Seventh International Workshop on Bidirectional Transformations. (2018). https://2018.programming-conference.org/track/bx-2018-papers Organized by Kazutaka Matsuda and Jens Weber.

Colin Clark and Antranig Basman. 2017. Tracing a Paradigm for Externalization: Avatars and the GPII Nexus *(Programming '17)*. ACM, New York, NY, USA, Article 31, 31:1–31:5 pages.

Paul A. David and Julie Ann Bunn. 1988. The economics of gateway technologies and network evolution: Lessons from electricity supply history. *Information Economics and Policy* 3, 2 (1988), 165 – 202.

Andrea A. diSessa. 2000. *Changing Minds: Computers, Learning, and Literacy*. MIT Press.

Marijn Haverbeke. 2015. Collaborative Editing in ProseMirror. (2015). http://marijnhaverbeke.nl/blog/collaborative-editing.html

C.A. Hesse. 1991. *Publishing and Cultural Politics in Revolutionary Paris, 1789-1810*. University of California Press.

Ghita Jalal, Nolwenn Maudet, and Wendy E. Mackay. 2015. Color Portraits: From Color Picking to Interacting with Color *(CHI '15)*. ACM, New York, NY, USA, 4207–4216.

Stephen Kell. 2009. The Mythical Matched Modules: Overcoming the Tyranny of Inflexible Software Construction *(OOPSLA '09)*. ACM, New York, NY, USA, 881–888.

Imre Lakatos. 1978. *The Methodology of Scientific Research Programmes: Philosophical Papers*. Vol. 1. Cambridge University Press. https://doi.org/10.1017/CBO9780511621123

Karl Marx and Friedrich Engels. 1848. *The Communist Manifesto*. Workers' Educational Association.

Lucy Suchman. 2002. Located accountabilities in technology production. *Scandinavian journal of information systems* 14, 2 (2002), 7.

David Wachsmuth, David Chaney, Danielle Kerrigan, Andrea Shillolo, and Robin Basalaev-Binder. 2018. The High Cost of Short-Term Rentals in New York City. (2018). http://www.sharebetter.org/wp-content/uploads/2018/01/High-Cost-Short-Term-Rentals.pdf