LINK DIGITAL
+ ONLINE + STRATEGY + DESIGN

OPEN DATA FOR THE PEOPLE

Dr. Greg von Nessi
Link Digital, Lead Data Scientist

WHAT DO PEOPLE WANT FROM DATA?

It may be obvious but people are not businesses. With all the buzz and talk surrounding 'Big Data' making just about everything better, it's easy to forget that 'Big Data' is really a collection of concepts that benefit businesses much more than the individual; and that's not necessarily a bad thing. Yes, the person in front of a screen may get some more relevant search results or advertising pop-ups, but ultimately, it's the market edge a company gets when using 'Big Data' that has led to its boom. Again, not a bad thing.

"...for most people, data translates into getting something done or learning something."

But what about the people? While everyone uses data all the time to make decisions, learn, formulate views, etc., no single individual has the ability to statistically process the amount of data that a company like Google or Amazon does on a continuous basis. If you're looking to buy a home and want to figure out what the fire risk to the property is, chances are you are not going to be too happy if your real estate agent just plops a bunch of actuarial tables in front of you. However, if you are an insurance company, those tables are exactly what you want for determining fire risk. People need something to pre-process the raw data before they can make sense of it.

So, if people need an intermediary to make sense of raw data, what are they? We categorise intermediaries as either being an application or piece of data journalism. Here are some examples:

Applications:

- Search Engine Interfaces
- Trip Planners
- Online Maps

Data Journalism:

- Sustainable Energy Without the Hot Air *(excellent book by D. J. C. MacKay)*
- XKCD *(funny webcomic with some fantastic data visualisations)*
- Political polls/trends

Obviously, the line between data journalism and applications can be pretty blurry; but we generally classify an application as something that empowers an individual to accomplish some task, where a piece of data journalism facilitates some form of learning or development of a personal view or opinion. So, for most people, data translates into getting something done or learning something. Of course, 'getting something done' or 'learning something' may be just a part of a larger, individual goal; but we think those two broad motivations are particularly effective in distilling what people overwhelmingly want to get out of data.

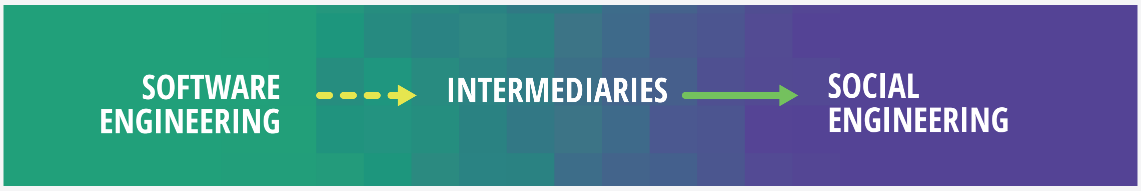
DECISIONS, DECISIONS

Most people have no problem articulating what their preferred news sources, writers, phone/computer apps, etc. are. While humans are not so crash hot on processing raw data, we seem quite comfortable making judgment calls on entities that process the data for us. Do you prefer to watch CNN, Fox News, BBC or Al Jazeera? Each one of those news outlets, by and large, has access to the same news sources; but they all present the data from those sources in very different ways to their viewership. Despite this, humans have no problem picking one or more as being their preferred. In some very real sense, this sort of decision making is something we have evolved to being good at. However, with the advent of the internet, the target of this data filtration is no longer restricted to faceless markets/demographics; it can be tailored right down to the individual. For instance, modern internet search engines will normally try to factor in a user's previous search history when processing a list of search results. So, while one person searching for the term 'fusion' may get a list of links about nuclear physics, another may get a list of cooking sites. As social creatures, reliant upon communities to survive, we have always had to make decisions on who can do what the best *(i.e. no one person is the best at everything)*. This ability to make judgments of authority and delegation is something our brains excel at by design.

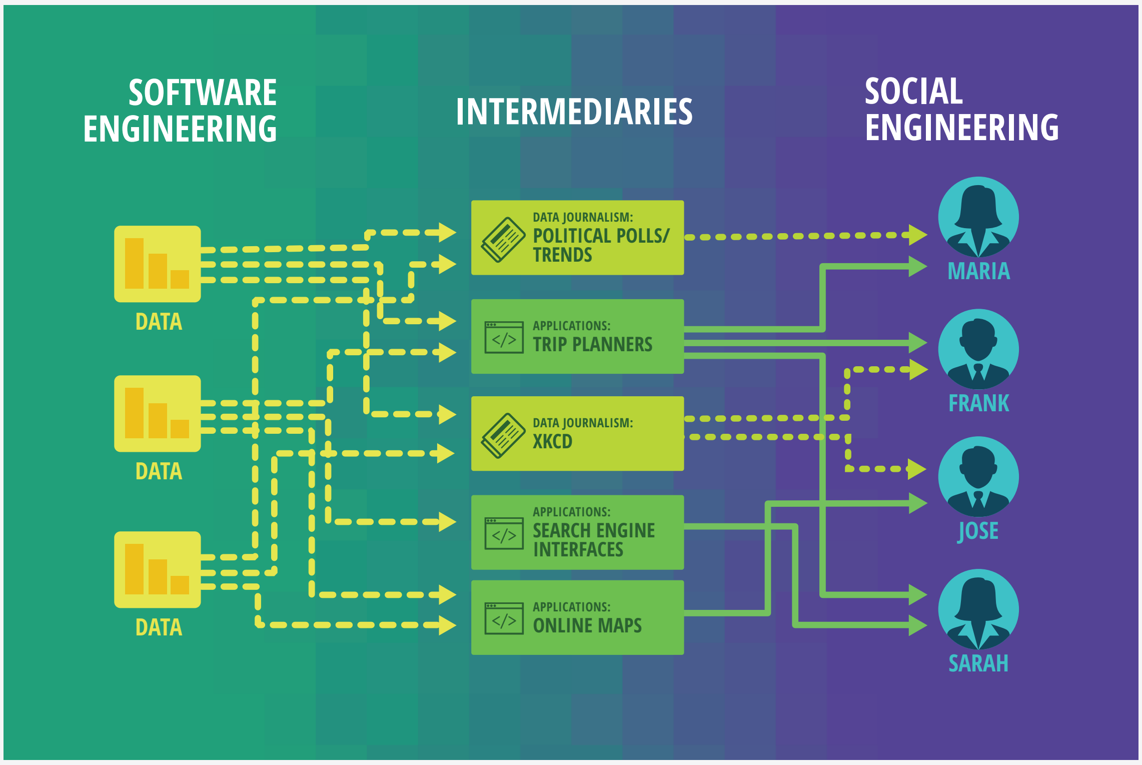
NOT JUST A SOFTWARE ENGINEERING PROBLEM

OPEN DATA PORTALS (ODPs) AS A FOUNDATION OF TRUST

Coming back to applications and data journalism, a picture starts to emerge.



So, to bring data to the people there needs to be a facilitator between data and the applications/journalists that need to use it. With that facilitator in place, apps will be created, data journalism pieces written and the people will then be able to start using those pieces of work. However, for this data chain to be effective and efficient, certain key requirements need to be fulfilled: data needs to be accurate and easily discoverable, while the data processing entities need to be known and trusted. Trust is the only one these requirements that has a dependency on the others. To put it another way, a trusted processor of information has to be known to individuals, while being able to access as much accurate data, relevant to its context, as possible. Obviously, these are not the only necessary requirements that empower trust, as it is still possible to misuse good data. In this context one can look at application and journalism entities, as transforming the individual's task from processing data to that of making a judgment call based on how much they trust the processor.


SOFTWARE ENGINEERING
INTERMEDIARIES
SOCIAL ENGINEERING
DATA
DATA
DATA
DATA JOURNALISM: POLITICAL POLLS/ TRENDS
APPLICATIONS: TRIP PLANNERS
DATA JOURNALISM: XKCD
APPLICATIONS: SEARCH ENGINE INTERFACES
APPLICATIONS: ONLINE MAPS
MARIA
FRANK
JOSE
SARAH

"With a strong open-source community, the pathways for extending awareness, generating effective advertising, etc. for data processing applications becomes a lot clearer and are more easily distributed than in the case of using a piece of proprietary software with developers being very isolated from users."

WHY OPEN SOURCE?

The image on the previous page defines two engineering domains, which have some level of overlap where they both meet at the App/Journalism layer of the graph. As one moves from the datastore to the user, the blend of engineering tasks gradually shifts from software to social. At the data layer, it is obvious there are a lot of technical tasks that need to be addressed with hardware and software implementations. Similarly, at the user layer, there are obvious social tasks that need to be engaged that surround advertising, awareness, community building, etc. In this view, open software solutions start making a lot of sense, as a successful open source data portal will be one that not only produces a strong software solution but one that also fosters a strong developer and user community around that software. Moreover, if we are considering a data chain that is sensitive to individual and not just a market context, an open source solution helps protect the interest of minority users within a processing application's target. In particular, an open source project can develop a diverse set of features, which may not otherwise fall in line with a proprietary solution's need to maximise the profits of its software. With a strong open-source community, the pathways for extending awareness, generating effective advertising, etc. for data processing applications becomes a lot clearer and are more easily distributed than in the case of using a piece of proprietary software with developers being very isolated from users. Thus, the gradual shift from software to social engineering can be deeply merged into an open-source development workflow. An excellent reflection of this point is how open source projects provide a cornerstone to many civic hacktivism efforts, which include public hackathons. The explosion in open source project awareness and proliferation has subsequently led to a boom in these groups and efforts. Indeed, by using existing open source codes, developers involved with civic hacktivism do not have to re-invent the wheel every time they engage in a project and are thus able to create genuinely useful pieces of software in a very short amount of time. While civic hacktivism is a growing social trend, it requires heavy lifting in both the social and software engineering domains whose continued growth will be greatly aided by the expansion of open data publishers and greater engagement/appreciation from end users.

Beyond the above arguments, having an effective integration with a distributed, open-source community helps manage the risk associated with getting locked into proprietary solutions or a single vendor. It also becomes a lot clearer about what features are well-supported/established and which are not; again, this can facilitate better risk management on the part of the ODP in terms of determining which features/extensions get deployed. In general, the open source model is naturally resistant to information/business silos being constructed within the project, which are generators of risk.

Some potential issues with open source are that it can take a long time for the software to get features to a mature, well-supported state. Thus, there may be situations where an ODP needs to implement a feature on timescales faster than which the community is moving. This can lead to frustration and some poorly thought out implementations. However, this risk can and should be mitigated by strongly feeding back into the core open source implementation via pull requests, posting issues and core developer involvement. This feedback ensures that features get implemented correctly in the codebase, or at the very least alert the core developers to potential needs for architecture restructuring. Again, the development of an ODP is both a software and social challenge; thus, a good ODP will always have a strong feedback into its core open source community.

MACHINES ARE USERS TOO

Stepping back into a higher level view of ODPs, it is clear that a good ODP is needed to empower the apps and journalistic works that ultimately bring data to the people. However, this also implies that the primary users of ODPs are machines, not people. This perhaps unintuitive conclusion is of key importance when it comes to the technical design of an ODP, as high-value features for machines are generally quite different from those that are of high value to individual users. Of course, ODPs need to be usable by people as well as machines, as humans are still required to make many of the decisions surrounding data curatorship; and if nothing else, individuals should always have the ability to investigate data directly for themselves. However, the vast majority of ODP usage will be via machines and not via a direct human interface.

Ultimately, this combined with the above leads us to the following set of desired characteristics for an ODP:

- the ODP be open source
- built primarily for app/journalistic intermediaries that, in turn, help people with tasks or learning
- empower intermediaries to deliver data to individuals that best fit their personal context
- empower intermediaries to build trust with their users by ensuring provided data is easily discoverable and reliably accessible

NITTY-GRITTY TECHNICALS OF AN OPEN DATA PORTAL

WHAT EVERYONE EXPECTS OF AN OPEN DATA PORTAL

We have so far painted a picture of how the ODP fits into the wider chain that supplies data to people and that an effective ODP has to be one that empowers intermediaries in their efforts to garner trust amongst potential users. Now, we zoom in on some of the technical requirements an ODP needs to satisfy to be an effective part of this data delivery chain.

Before moving into specifics, we make some assumptions about features an ODP is to provide; a standard base level of expected functionality:

- maintains a catalog of metadata
- both custodian users and machines are able to manage the metadata catalog
- options for custodian users and machines to upload datasets to at least one persistent, highly available datastore
- reliable data downloading from the ODP's datastore*(s)*
- options for custodians to link in external data
- ability to extend the ODP's core functionality without breaking the core software upgrade path
- open sourced *(see the previous section)*

While most people will agree that at least some of the above points should be core features of any software powering an ODP, there are still some technical points that are not so clear cut.

DATA AND HISTORICAL PATH ENTROPY

The one big difference between code and data, is that code is almost always manipulated via the auspices of a programmer typing into a console, i.e. transforming the uncompiled code. Data, on the other hand, can be transformed by any sort of codified mathematical operation, which there is a countless amount of.

A concept that motivates much of what we think an ODP should and should not provide, surrounds a concept we call 'Historical Path Entropy'. The quotes are probably unnecessary, as we believe the term to be pretty accurate from an information theory point of view.

One can view any dataset as the endpoint of a timeline encompassing a sequence of create, read, update and delete (CRUD) events that a dataset has gone through to arrive at its current state. The historical path entropy of a dataset corresponds to the number of possible timelines that could have led to the current state of the dataset. Historical path entropy is minimised if a historical log of its timeline is available *(i.e. there is only one path)* and increases whenever data is duplicated *(i.e. a snapshot branch off another timeline)* or transformed without being noted in its historical log.

A concrete example of just how important this is can be found in software engineering, which has been hugely influenced by tools like CVS, SVN and GIT to basically manage a historical log of code development. These tools enable developers to know exactly how a piece of code got to its current state and has facilitated a revolution in how code is developed and documented. We're basically saying data should be created and maintained in almost an identical fashion.

The one big difference between code and data, is that code is almost always manipulated via the auspices of a programmer typing into a console, i.e. transforming the uncompiled code. Data, on the other hand, can be transformed by any sort of of which there are a countless number. So, instead of associating a user with a particular code change, an entity and a unique identifier of the transformation operation itself should be associated with the data manipulation record.

Any imperfections *(or outright absence)* of such an authoritative dataset log, effectively leads to a higher historical path entropy, which means the uncertainty of how the data arrived in its current state increases. At the risk of stating the obvious, this is extremely important for both businesses and journalists using the data; such historical uncertainties can hide hidden biases and data processing errors that could easily have a business or journalistic impact on its use.

APIS AND POST-PROCESSING

In general, an ODP will constantly grow in terms of the number of datasets it provides. Moreover, it will often be the case that these datasets will have some wide degree of variance in terms of size and complexity. This leads to an intrinsic problem with ODPs providing API endpoints or any post processing facility for its datasets, as it becomes impractical to give any concrete service level agreements (SLAs) for the functionality of those endpoints.

Most data-oriented business models will be built around SLAs of the services they provide. Obviously, if a business decides to leverage an ODP's API endpoint for their product, the associated SLAs cannot be better than the SLAs of the ODP API endpoint. So, we come to a problem with ODP APIs being unable to provide assurances for API functionality.

Of course, an ODP with sufficient funding and staff could provide SLAs for its datasets; but this still would imply either 1) The cost of running the site scales with its size, 2) The data provided by the ODP is constrained to a size manageable by the current resources. The first option is almost always unacceptable for obvious reasons; unbounded budgets don't exist. The second option will force many ODPs to go against some core Open Data philosophies: it may require deletion of datasets or holding data away from public view. Neither option is a good one.

Compounding this is the fact that controlling un-throttled APIs can lead to larger businesses placing a high, persistent load on the system or even white DDoSing the site.

In short, ODPs shouldn't be advertising API endpoints under their control for business-critical applications. There just is no way to minimize business risk over a long term interval when providing this sort of endpoint.

Instead, businesses should be periodically updating via download in a batch process from the ODP, subsequently using their local copy to power their application. This protects the business from being directly reliant on real-time data availability from the ODP. It also provides a buffer in case the dataset changes in a way that breaks functionality in their application.

Of course, it is perfectly fine for an ODP to provide references for external API endpoints, which will presumably have their own SLAs, user agreements, etc., as the risk associated with managing that endpoint will not fall directly on the shoulders of the ODP.

Data post-processing is something closely linked with APIs, as one common function of Data APIs is to furnish some sort of SQL-like querying interface, which can easily communicate data processing instructions. Realtime post-processing of the data faces all the issues listed above with APIs and then some. In particular, post-processing data leads to higher historical path entropy. Even in the process of converting one data format to another, errors can creep in, numerical precision lost or falsely extended, etc. Again, we believe that all such conversions need to be included in the historical log of the dataset.

If an ODP finds the necessity to post process data, then it should be run as an asynchronous batch process, with appropriate additions being made to the datasets historical log in its metadata. A good way to handle this would be through the auspices of a micro-service. A micro-service here would constitute a running piece of software which would be sandboxed away from the primary ODP software *(e.g. running on its own virtualised instance)*. Micro services would be managed uniquely from the core ODP, having a uniquely defined deployment domain over datasets, SLA, hardware requirements and associated business contracts. With any micro-service, there should be some general implementation in place to ensure dataset logs are appropriately updated for any datasets manipulated by the micro-service.

HARVESTING AND SEARCH FEDERATION

A pattern that has emerged in ODPs is the use of data-harvesting from other portals (pushing or pulling). Harvesting from other portals causes several technical issues and is, in general, too difficult to make robust enough for enterprise-level applications. Problems with harvesters typically have to do with the following: endpoint having a different metadata schema, endpoint changing unexpectedly, endpoint becoming un-responsive, batch-pull harvesting taking an excessively long time for large harvests, data duplication, data deletions handled incorrectly, and harvester not running fully to completion. Compounding these technical issues is the fact that most harvester implementations increase the dataset historical entropy, if not obfuscating what the real source of truth for the dataset is completely.

Another issue we've seen is when most the data held by an ODP is harvested data. The situation is obviously not a good one, as most the ODP's datastore is dedicated to duplicate data for which the ODP is not the source of truth for. This can be taken to an extreme where the clear majority of an ODP's system resources are dedicated to handling data for which it is not the source of truth.

At the end of the day, we have never seen a compelling use case for harvesting data in an ODP context. Unfortunately, funding for many ODP's is based on the number of datasets they host, so harvesting is used to inflate those numbers.

A much more sensible alternative to harvesting is building search federations: augmenting local dataset search results with those of partnered datasources/ portals. This basically presents search results as comprising of both local datasets and non-local datasets whose entries directly link to an external data portal. In practice, we have found this to be much easier to support and maintain. Moreover, the non-local search results can be generated in real-time, which ensures non-local results are always up to date; and since data isn't being duplicated, the process is adding to the historical path entropy of the dataset.

WHAT SHOULD AN ODP PROVIDE?

So, after all this, we have a slightly extended wishlist of features for an ODP:

Data Journalism:

- Spec for site meta-data schema
- Data quality criterion and ratings
- Meta-data search capabilities
  - Federated with partnered ODPs
  - Historical log with links to previous transformations
- Reliable dataset downloads
- Throttled API endpoint

Applications:

- Spec for site meta-data schema
- Data quality criterion and ratings
- Meta-data search capabilities
  - Federated with partnered ODPs
  - Historical log with links to previous transformations
- Reliable dataset download

Overall, we think an ODP should facilitate the finding of data, creation and maintenance of a historical data log and provide a robust facility for acquiring data for which it is the source of truth.

With available cloud services, it is easy to provide an enterprise-level download endpoint for datasets, even with the nature of ODPs expected to monotonically increase the number of its datasets.

Data quality measurements are a good metric to present to the user, as it provides context for how well the data is relatively curated in an ODP with possibly many data-publishers, which is a unique and useful facility for an aggregating ODP to provide to its users. Even though data quality criterion can be somewhat subjective, statistically such an assessment will become more useful and refined as the ODP grows. This is one case where the ODP's nature to constantly grow works in its favor. Indeed, in an ODP with many datasets, such quality ratings become invaluable as a judgment of a dataset's quality relative to the set of other datasets under that same rating scheme.

Probably the most import facility furnished by the ODP is the ability to efficiently search through metadata. Indeed, one can argue the whole point of ODPs is to facilitate data discovery. To us, metadata searching should enable one to search via contextual and geospatial criteria. It also should enable one to explore the historical log of datasets, providing external links to parent datasets and/or processing codes whenever possible. It should also provide the ability to find child datasets *(in its search federation)* of a given dataset.

TL;DR

While an ODP can be easily understood as being key in bringing data to the public, it has to be much more than a website with a bunch of spreadsheet download links. People are generally bad at processing large quantities of numerical data, but they are really good at figuring out which applications and news sources they trust to do this processing for them. Hence an ODP needs to be designed to empower applications and journalists to produce trustworthy distillations of that data.