Data-Driven Rankings: The Design and Development of the IEEE Top Programming Languages News App

Nicholas Diakopoulos
College of Journalism
University of Maryland
nad@umd.edu

Stephen Cass
IEEE Spectrum
cass.s@ieee.org

Joshua Romero
IEEE Spectrum
josh@joshromero.com

ABSTRACT

In this paper we explore the design and development of a data-driven app that ranks programming languages. We detail the design rationale and editorial decisions that constitute the app, including several novel features that allow end users to perturb and compare rankings by interacting with the data and algorithm used to synthesize that ranking. Our analysis of the social media response to the app on Twitter and via comments found that there was some, though not extensive, use of these features and that a rich commentary arose around the app to discuss the definitions, classifications, and data proxies employed. These results suggest several directions for further research in understanding the integration of data, algorithms, and transparency as they affect the user experience of a data-driven ranking.

Keywords

Computational and data journalism, algorithm transparency

1. INTRODUCTION

The publication of rankings in the media can provide a wealth of information that informs the public and serves as a valuable input to decision making processes, from the day-to-day choice of a restaurant reservation to the potentially life-changing decision of what university to attend. Responsibly and accountably communicating a data-driven ranking to the public opens a host of interesting questions in computational and data journalism. While such rankings may appear authoritative due to apparent quantification, editorial decisions abound: from definitions and data inclusion/exclusion decisions to the choice of the importance of various inputs, the algorithm used to synthesize the ranking, and even what kind of interactivity is exposed to the end user [6].

In this paper we examine some of the editorial decisions apparent in building data-driven rankings and explore the possibility for increased transparency in such rankings. We detail the design and development of a data-driven ranking of programming languages: the IEEE Top Programming Languages (TPL) App. Such a ranking is potentially valuable in a number of decision contexts such as what language to learn next for a job, what to teach, or what might be appropriate for a project in a particular domain.

There have been a number of previous efforts to build programming language rankings that we studied and learned from in developing our own design. Oftentimes rankings hinge on only one type of data input, such as Search Engines[1], Google Trends[2], or in some cases producing separate rankings from a variety of different data inputs[3,4]. In our design we instead chose to weight and synthesize a number of different data inputs into an aggregate ranking. In order to provide more interactivity and flexibility our design allows for data sources to be re-weighted, and to compare how different combinations of data inputs and weights affects the output ranking. Our approach thus offers additional transparency to the ranking algorithm used to produce the output, and enables new forms of interactivity with the rankings.

In this paper we contribute a detailed description of the design of the app, including an in-depth treatment of editorial considerations related to definitions and classifications, data sources as proxies, and transparency information, which are broadly applicable to other data-driven news apps as well. We also present the results of analyzing the social media response to the app on both Twitter and via the on-page comments to the app in an effort to better understand what features were shared and discussed, and what aspects of the data journalism were most criticized. We found evidence that a minority (16%) of users who shared a link to the app on Twitter had manipulated the ranking

Figure 1. Top Programming Languages App

[1] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[2] https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of Programming-Language

[3] http://langpop.com/

[4] http://redmonk.com/sogrady/2014/06/13/language-rankings-6-14/

(e.g. data sources or weights) and that there was a rich discussion relating to many of the editorial decisions relating to definitions, classifications, and proxies used to measure the "popularity" of a language. Our results suggest that commentary can enable a useful feedback loop that encourages reflection on the quantifications and operationalizations used in a data-driven news app.

2. THE APP

The IEEE Top Programming Languages (TPL) app was published online on July 1, 2014 (http://spectrum.ieee.org/static/interactive-the-top-programming-languages) with a non-interactive print version also appearing in the July issue of IEEE Spectrum Magazine. Here we focus only on the online interactive version. Shown in Figure 1, the TPL shows an initial ranking of the top ten languages according to a default "IEEE Spectrum" weighting of twelve different metrics from ten different data sources including everything from Google search and trends, to Github and Stackoverflow activity, job postings, and an academic digital library from the IEEE. The choice of these data sources, as well as others that were not included is detailed in the next section. While only ten languages are shown initially, in total there are forty-nine languages ranked by the app. The user can access these by clicking a "show extended ranking" button (Figure 1 bottom).

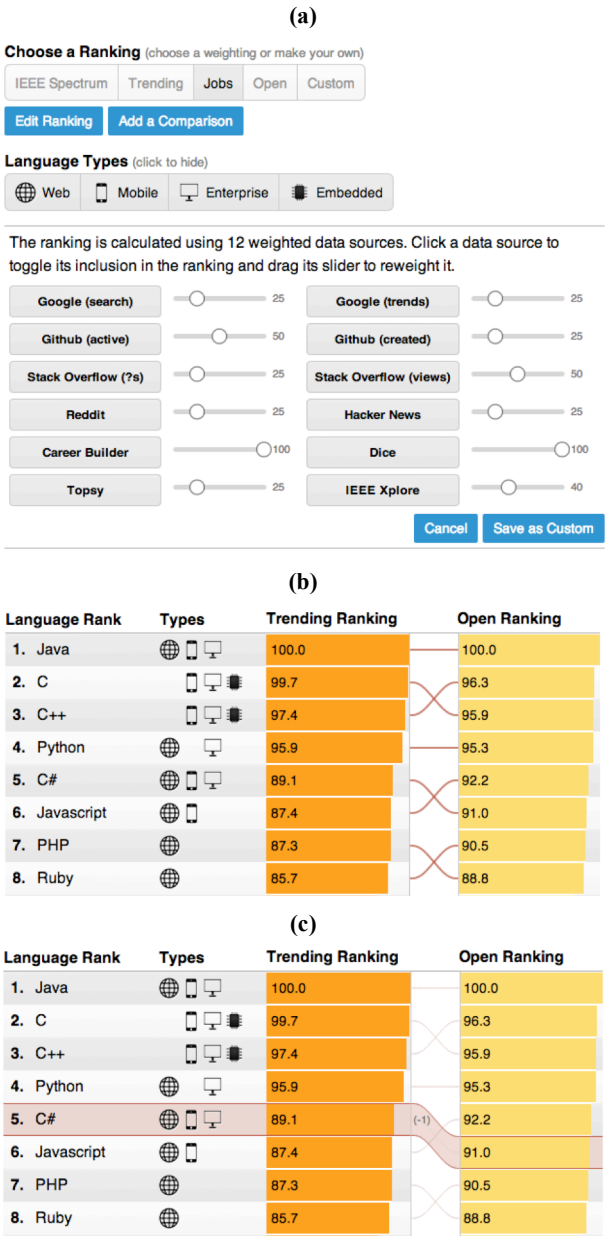

Figure 2. (a) interface for editing weighting of data sources that contribute to the ranking; (b) comparing the "trending" ranking to the "open source" ranking; (c) detailed comparison and rank shift for the C# language.

Weighting presets are provided so that the user can quickly see different rankings that prioritize the influence of different data sources on arriving at the final ranking. These include weights for emphasizing languages that are "trending", "open source" or in demand for "jobs". The user can also create a custom weighting where they decide what data sources to use and how each of those data sources should be weighted (see Figure 2a). The user can click a data source to toggle its inclusion or use a slider to reweight it. Any of the presets can be tweaked and tuned to match the user's own mental model of what they think the ranking should entail. Moreover, once these weightings have been customized the URL of the app is updated so that the customized ranking can be shared as a deep link to app state [4].

Each language has also been tagged with "types" indicating if the language is relevant to different programming environments including: web, mobile, enterprise, and embedded and these are surfaced in the UI as filters that allow the user to show or hide languages of a given type.

Figure 2b and 2c show an advanced feature of the app which allows the user to visually compare two different ranking schemes. So, for instance, Figure 2b shows the "trending" ranking versus the "open source" ranking and allows the user to see how much up or down a language moves in the ranking between the different weighting schemes. Each of these weightings can also be customized, thus enabling scenarios where a user could compare their own unique ranking to one of the presets. Figure 2c shows the interface when the user clicks on a specific language: it highlights the rank increase or decrease between the two rankings.

3. EDITORIAL CONSIDERATIONS

As the app was designed and developed many editorial decisions shaped what the app would be, what data it would incorporate, and how it would use that data to arrive at rankings. Chief amongst those considerations were questions of what languages warranted inclusion in the app, what proxies could be used to measure different types of activity related to each language in a reliable and valid way, and how to expose the ranking in a way that enabled end users to effectively interact with the data.

3.1 Definitions, Inclusions, and Classes

Some rankings of programing languages, such as the popular TIOBE index, require that languages ranked be Turing-complete (that is they can express all possible algorithms). But rather than take a normative approach, we instead defined what constituted a language for inclusion in our ranking both pragmatically and by leveraging data. First, we adopted a relaxed but pragmatic definition of programming language: "a particular syntax for commanding a computer to perform some task" which could apply to markup languages like HTML as well as query languages like SQL in addition to languages like Python and Java. Next, we created a candidate set of languages by examining other programming language rankings such as LangPop and TIOBE, as well as scraping the list of 150 "official" languages visible in the

GitHub trending repository dropdown menu. Then we fed these candidates through a Google search of the form "X programming" and recorded the number of hits for that query. This allowed us to assess the strength of the signal for each language and get a quick understanding for which languages had enough of a measurable activity level. Languages with very few hits, like "OpenEdge ABL" or "Objective-J", were thus filtered out. A final pass was made to assess language distinctiveness; we decided to remove "Pascal" since its practical use was already captured by Delphi, and to remove "Synergy/DE" which is a derivative of COBOL.

Once the final set of 49 languages was complete, we wrote descriptive text for each that would later be included in the app. We also tagged each language as being relevant to any of four different programming domains: web, mobile, enterprise, and embedded. This tagging was done by an editor of the app who judged each language and assessed if it was a predominant and important language to each of those programming domains. The idea of typologizing languages this way was to provide additional capabilities for filtering languages in the UI.

3.2 Selecting Data Sources as Proxies

We took a human-centered approach towards identifying and selecting data sources that could reliably measure the popularity of the various languages we chose to include in our ranking. We first asked ourselves what the use-cases or scenarios of a language ranking app would be. For instance some users may be interested in job prospecting and seeing which languages have the most demand in the market, while hiring managers might want to see where there is more supply of programmers when starting a new project. Still other users might be interested in their career development to see what languages have higher compensation or what the trending languages are that they need to pick up to stay relevant. Finally, teachers might want to know the market demand for various languages when choosing what to teach.

As we considered these use cases we began to assess possible sources of data that could help us evaluate each language's rank: how popular was it in search or trends, was it used for open source projects, did people ask a lot of questions on social sites about it, were there many job postings, what kind of teaching resources were available for it, etc. We acknowledged that whatever proxies we chose they would be imperfect, since the signal that surfaces online and is readable through APIs and web analyses only captures what is expressed publicly about languages rather than information that might be internal to an organization like the cost or success rate of using a language. Of course there are many other ways that we could have assessed language use, through surveys for instance, though we chose to use publicly and freely available sources as reasonable proxies of popularity.

For each of the sources in Table 1 we looked at the quality of the data from the source including (1) its comprehensiveness, (2) rate limits that might affect the feasibility of collecting enough volume of data to provide a reliable signal, (3) the ability to minimize query collisions and ambiguities for the source (e.g. searching for "C" vs. "C++" is sometimes not supported because some indices do not differentiate the "++" characters), (4) the ability to take time bounded measurements from the source (e.g. limit a query to just 2013), and (5) any legal liabilities or terms of use considerations for the data source. Some of the sources were removed from consideration over legal concerns (e.g., Amazon and LinkedIn) while others were simply not distinct enough (e.g. Bing Search). The final 10 sources (and 12 metrics) that we settled on covered a wider range of proxies for measuring programming language popularity than any previous rankings we had encountered and included everything from Google search and trends, to GitHub repositories, Twitter mentions, Stack Overflow activity, Reddit threads, Hacker News mentions, Job listings on Career Builder and Dice, and mentions in academic literature though the IEEE Digital Library.

Table 1. The various data sources considered for the app including what each source is designed to measure (light gray sources were dropped).

Source	What it measures
Google Search	Information resources available online
Google Trends	Information demand
GitHub	Public code repositories both (1) created and (2) active in the last year
Topsy	Activity and social chatter on Twitter in the past year
Stack Overflow	(1) Number of questions related to each language (self-tagged by poster) and (2) magnitude of attention paid to those questions over the last year.
Reddit	Number of posts mentioning language
Hacker News	Stories and comments mentioning language
CareerBuilder	Job listings mentioning language
Dice	Job listings mentioning language
IEEE Xplore	Use of languages in academic publications
Meetup	Real-world social activity and learning
Slideshare	Presentation and teaching materials
LinkedIn	Job listings
Indeed.com	Job listings
Amazon Search	Book information resources available
Bing Search	Information resources available online

3.3 A Transparent User Interface

The design of the user interface was guided by a few high-level design objectives aimed towards facilitating a deeper understanding of the rankings and data sources as well as how they were synthesized into the various rankings that the user perceived. Most importantly we wanted users to be able to *quickly see* the overall IEEE ranking and then be done if that's all they wanted; a low bar for participation. But just as crucial for our technical audience, we wanted to allow users to drill into the ranking by *editing the weightings* of the various data sources, be able to *compare* different sets of weights, and *filter* to see sub-rankings according to the language domains we had tagged. These capabilities are shown in Figure 2.

The motivation for transparency was twofold: (1) as an attempt at a user experience for enabling algorithmic transparency into how the ranking was synthesized [3], and (2) as an entry-point for engagement for users to express disagreement with the editorial choices / weights that we had set as defaults. We hoped a transparent approach toward the UI could be good for building credibility with the audience so that the ranking didn't appear so

cut-in-stone and to publicly acknowledge the various decisions in terms of data sources and weights that needed to be made. In the next section we report on how some of these transparency features were utilized.

4. ONLINE RESPONSE

After the July 1 launch there was substantial social media activity surrounding the app. In the two weeks post publication there were almost 1,300 tweets sharing the link to the app, roughly 1,500 Facebook shares, and more than 270 comments on site as well as several hundred more comments scattered across discussion forums like Reddit[5] and Slashdot[6]. Here we consider what can be learned about the story by analyzing this social media response, with a focus on the Tweets that were shared and the comments that were made directly on the website of the app. With an eye toward understanding how to better design future ranking-based data journalism apps, we approached this material with questions about what features of the app were shared and discussed and what aspects of the data journalism were most criticized.

4.1 Twitter

Using the Topsy API[7] we gathered all tweets referring to the app from July 1 to July 18, 2014. We acknowledge that Twitter may not be a representative sample of users [8] however the platform is still a useful way for us to understand how the app was shared. Our collection resulting in a set of 1,285 tweets (844 original and 441 retweets). Among the 844 original tweets we further identified 127 tweets (15.5%) that had shared a URL indicating that the user had edited either the type filters or weighting scheme in the app. These deep links reflect app state and allowed us to further drill into how users were interacting with the rankings. Of the 127 tweets we found 68 (53.5%) that reflected non-default filters and/or weightings in some way: 28 (41%) changed only the filter, 23 (34%) changed only the weighting, and 17 (25%) changed both the filter and weighting.

In many cases the text of these 127 tweets was the boilerplate that would be shared via the "tweet" button on the app page. But in at least 6 cases we saw users customizing the tweet text to refer to specific weightings and filters. One tweet that shared a link to a jobs weighting of the ranking wrote: "*Interactive #Infographic: Top #Programming Languages by Popularity, #Jobs, and Type*" and another (written in Japanese and translated with Google) that filtered out all non-web languages wrote "*python is really in second place if you focus on web?*". Overall though, even amongst the tweets that share deep links we find limited editing (5%) of tweet text to refer directly to the filters or weights.

Perhaps most interestingly, in all cases where the weight had been changed it reflected one of the preset combinations of weights for "trending", "jobs", or "open source" indicating that at least amongst the users that shared the URL on Twitter there was no indication that users substantially customized a ranking aside from the presets that were provided. Among all 1,285 tweets shared there were none that shared a deep-link to a comparison view of the visualization, suggesting that this capability may not have been compelling for users to share, or the feature was not designed to be prominent enough.

[5] http://www.reddit.com/r/programming/comments/29m2hp/the_top_programming_languages/

[6] http://developers-beta.slashdot.org/story/14/07/05/1924232/ieee-spectrum-ranks-the-top-programming-languages

[7] https://code.google.com/p/otterapi/

4.2 Comment Threads

On July 18, 2014 we collected all of the comments that had been made on the app's main page. This resulted in a set of 278 comments which we then analyzed through a process of open coding by looking for themes and reoccurring topics that were brought up in the discussion [7].

In line with previous studies of comments around online data visualizations [5] we found that many of the comments to the app introduced various aspects of context from personal or professional experience relating to the languages, and that others were critical, though often in productive ways, like pointing out issues related to missing items, the distinguishability of items, the choice of proxies, classification decisions, or the definitions used. Virtually all of these criticisms related in some way to the various editorial decisions that were made in the course of designing the app such as inclusion/exclusion decisions for data elements, language definitions and type classifications, what data sources were used, as well as their validity as proxy variables for the "best" languages.

Classification-oriented comments manifested in several forms: critiques and questions of *definitions* (what is defined to be within or outside scope), and more direct critiques about the accuracy or *applicability* of a particular classification

Comments critiquing definitions were often related to the question of "what is a programming language and why is this particular programming language on the list?" or "what are the boundaries of a specific language type?" Perhaps the most critiqued inclusion in the ranking was HTML, which was not considered a programming language by several commenters. We took a pragmatic approach to the definition which ended up conflicting with more normative views. HTML wasn't the only language criticized though. Other comments asked about ASP.NET, Arduino, Verilog, and SQL. "*SQL - Indispensable tool for SQL database programmers? Yes. Programming language? I don't think so*," wrote one commenter. Yet, these kinds of definitional boundary questions also prompted additional clarifying context by other commenters, and in the case of SQL a counter argument that with some standard extensions SQL is Turing-complete and thus considered a programming language by a classical definition. Some questions of definition also focused on the language type we had assigned: "*Does Embedded mean a language that extends an application or embedded into a system?*" or on the boundaries of a language: "*Visual Basic includes VB6?*"

On the flip side of what languages were definitionally included in the app are also those that were *excluded*. On our methodology page for the app[8] we described the process by which we went from 150 candidate languages down to 49, however we might have been more explicit in explaining additional criteria that we used. The most typical reason for exclusion was that a language was too obscure and didn't have a strong enough signal in our data sources to measure reliably. But this didn't stop some commenters, likely fans of some of those obscure languages to chime in to ask why languages like APL or IDL were not included. One thoughtful commenter of a language with more prevalent use did ask, "*How come Mathematica never shows up in the list?" "mathematica programming" gives ~135,000 hits in Google, more than many languages listed …*" Future iterations of the app might usefully revisit any of the languages mentioned in

[8] http://spectrum.ieee.org/ns/IEEE_TPL/methods.html

comments to re-evaluate their wider relevance and our ability to reliably measure them.

Other comments were oriented towards the editorial choices we had made in classifying languages into various domains for web, mobile, enterprise, and embedded. Comments could either suggest additional types for a language (e.g. "*C# is also used for embedded devices with the .NET Micro Framework*") or criticize and claim that a particular language should not be typed in the way that we had. Oftentimes these comments would lead to long follow-on threads of discussion where other commenters could dissect the definition of e.g. "embedded" to understand why it may or may not be applicable. In some sense these "edge cases" provided ambiguity that allowed for more discussion, since the classification decision could be teased apart socially and by marshaling specialize knowledge from the crowd of commenters.

A final set of comments was more broadly critical of the *methodology* used by the app, bringing up questions about the representativeness and validity of data sources employed: *"...I expect that very few developers of banking software, for example, are searching the web for language information, or posting code to GitHub - so their languages of choice simply don't show up in the surveys…*" Some commenters questioned whether the proxies we had chosen for popularity were not really measuring popularity: "*Looking at your weighting, I wonder whether the ranking reflects the amount of help people need with the languages, not their actual popularity*" or "*An equally plausible ranking would indicate that the top languages are so difficult to write in that people have to ask questions over and over again, slowing productivity."* Still others suggested there may be other variables of interest besides popularity that might be more valuable, such as quality, maintainability, or success of projects.

There was little discussion of being able to re-weight the languages or play with the data sources used to generate the ranking. One person thought it would be helpful to have a weighting preset for the "easiest to learn" languages. And one comment did reflect on the availability of transparency information for the app, in particular in calling for the ability to see the full equation and calculation of the input data that led to the values used to rank each language. The commenter wrote: "*Is it possible to see the calculation analysis for how you came up with the ranking for each language? for example, I'm surprised that LISP showed up higher than VHDL in the extended ranking and was curious about the different inputs for each."*

5. DISCUSSION

Several of the novel features designed into TPL, including the ability to re-weight and compare rankings visually, were conceived of as ways to provide more algorithmic transparency as well as to engender new ways of interacting with the ranking. Our results from the social media response show that about 16% of users did interact with the weighting and filtering scheme though not in a deep and highly customized way. Also we found no trace of users interacting with the ability to compare different rankings. While it's possible that this feature just wasn't useful or interesting to users, it could also be the case that users simply didn't *share* customized weightings and comparisons but did look at them. In future work we would like to log these interactions directly rather than rely on a social media proxy. Also, it would be interesting to contrast our results to a comparable news app with an audience that is less obviously computationally aware.

By studying the comments that were made to the app we found that users often commented on, criticized, and argued about definitions and classifications. If nothing else, such criticism is useful for further iteration on the app by pointing out boundary cases relating to the editorial decisions of inclusion, exclusion, or classification. These cases might be revisited or better explained in the accompanying text to the app. Yet at the same time the substantive discussion of classification boundaries and "errors" allowed people to collectively dissect and explore those classifications, perhaps leading to a deeper understanding of them while underscoring the ambiguity that is often inherent to classification [1]. Perhaps by providing some space for ambiguity in data journalism (e.g. in classification), this leaves opportunities for the "audience" to engage in co-constructing meaning.

An open question for future work is how the perception of algorithmically determined classifications might vary [3]. If for instance the "type" tags of each language had been determined algorithmically would this make users less likely to question the classification, or instead more skeptical? How could we design a classification algorithm that has facilities for explaining itself and its uncertainty [2]? We believe there is a rich space here for further research in understanding the integration of data, algorithms, transparency and the effects on the user experience.

In summary, we have presented the IEEE Top Programming Languages news app and detailed the design rationale and editorial decisions that constituted the app. We developed novel features that allowed end users to perturb, interact with, and compare rankings and found that there was some use of these features in the social media response to the app. At the same time, a rich commentary arose around the app to collectively discuss the definitions, classifications, and data proxies employed. Future work will look at logging interactions directly on the app, and in exploring different interfaces and explanation modalities for communicating editorial decisions in data-driven journalism.

6. REFERENCES

1. Bowker, G.C. and Leight Star, S. *Sorting Things Out: Classification and Its Consequences*. MIT Press, 2000.
2. Darlington, K. Aspects of Intelligent Systems Explanation. *Universal Journal of Control and Automation 1*, 2 (2013), 40–51.
3. Diakopoulos, N. Algorithmic Accountability: Journalistic Investigation of Computational Power Structures. *Digital Journalism*, (forthcoming).
4. Heer, J., Viégas, F., and Wattenberg, M. Voyagers and voyeurs: supporting asynchronous collaborative information visualization. *Proc. Conference on Human Factors in Computing Systems (CHI)*, (2007).
5. Hullman, J., Diakopoulos, N., Momeni, E., and Adar, E. Content, Context, and Critique: Commenting on a Data Visualization Blog. *Proc Computer Supported Cooperative Work (CSCW)*, (2015).
6. Hullman, J. and Diakopoulos, N. Visualization Rhetoric: Framing Effects in Narrative Visualization. *Trans. on Vis. and Comp. Graphics.*, (2011).
7. Lofland, J. and Lofland, L. *Analyzing Social Settings: A Guide to Qualitative Observation and Analysis*. Wadsworth Publishing Company, 1995.
8. Tufekci, Z. Big Questions for Social Media Big Data: Representativeness, Validity and Other Methodological Pitfalls. *Proc. Int. Conf on Weblogs and Social Media*, (2014).