The Scope and Impact of Open Source Software: A Framework for Analysis and Preliminary Cost Estimates

Carol A. Robbins* (NCSES, NSF)
Gizem Korkmaz (Biocomplexity Institute of Virginia Tech)
José Bayoán Santiago Calderón (Claremont Graduate University)
Claire Kelling (Pennsylvania State University)
Stephanie Shipp (Biocomplexity Institute of Virginia Tech)
Sallie Keller (Biocomplexity Institute of Virginia Tech)

Abstract

Open source software is everywhere, both as specialized applications nurtured by devoted user communities, and as digital infrastructure underlying platforms used by millions daily. This type of software is developed, maintained, and extended both within the private sector and outside of it, through the contribution of people from businesses, universities, government research institutions, nonprofits, and as individuals. This paper proposes and prototypes a method to document the scope and impact of open source software created by these sectors, thereby extending existing measures of publicly-funded research output. We estimate the cost of developing packages for the open source software languages R, Python, Julia, and JavaScript, as well as re-use statistics for R packages. These reuse statistics are measures of relative value. We estimate that the resource cost for developing R, Python, Julia, and JavaScript exceeds $3 billion dollars, based on 2017 costs.

Key words: Open Source Software, Intangibles, Innovation, Network Analysis

Introduction and Contribution

Beginning in the early 1980s, open source software (OSS) projects have provided users with zero dollar cost and freely modifiable software tools. These range from the LaTeX typesetting program introduced initially in 1983, to Linux, Apache, Mozilla Firefox web browser, and the WordPress content management system. As OSS has continued to grow, estimates of its market penetration have emerged in areas including operating systems, servers, and specialized

This work was supported by U.S. Department of Agriculture (58-3AEU-7-0074).

We thank Daniel Chen for providing insights and support. We also thank Virginia Tech's Data Science for the Public Good program students Keren Chen, Hannah Brinkley, and Eirik Iversen for their contributions.

*Corresponding author: crobbins@nsf.gov

languages. The most widely used operating system on the internet globally is the Linux-based Android operating system (GlobalStats statcounter 2018). As of July 2018, Apache is the most frequently used server on the internet (W3Techs, 2018). Based on the cost of the nearest available substitute, Greenstein and Nagle (2013) estimated the value of capital stock of Apache software in use in 2013 at between $2 and $12 billion.

OSS is increasingly developed and used by businesses; the top institutional contributors to the code repository GitHub in 2017 are Microsoft and Google. Nonbusiness contributors are active as well on GitHub, with Stanford University, the Massachusetts Institute of Technology, The Broad Institute, and University of California at Berkeley among the top contributors. Additionally, substantial contributors from the U.S. Federal Government include Sandia National Laboratory and the General Services Administration (Hoffa 2017). Recent policies of the U.S. Federal government now promote the posting and sharing of software source code developed by or for the Federal Government (Scott and Rung. 2016).

While this particular policy to promote reusing and sharing of software created with public funding is relatively new, public funding has an important and not fully accounted role in the creation of open source software (OSS). How much of the OSS currently downloaded and used has been created with public funding, in the U.S. or in other nations? We currently don't know. We do know that many of the most widely used OSS tools available have been developed in universities and other publicly-funded institutions. Apache is open source server software developed with federal and state funds at the National Center for Supercomputing Applications in Illinois. Linux was developed at the University of Finland. The R language was developed at the University of Auckland in New Zealand by professors for use in their teaching laboratory, with extended development by Hadley Wickham, including at Rice and Stanford Universities

(Ihaka, 1998, Wickham, n.d.). Independent nonprofit institutions have also played an important role in OSS, assuming the management and coordination of OSS projects; examples include the Apache Software Foundation, the Mozilla Foundation, and the Linux Foundation.

The work of OSS developers across sectors of the economy also highlights the role of non-business activities that result in important innovations. As the OECD's relevant measurement manual defines innovation: "the implementation of a new or significantly improved product (good or service) or process, a new marketing method, or a new organization method in business practices, workplace organization, or external relations" (OECD and Eurostat, 2005). The creation of new software products and their significant improvement through version upgrades and package extensions that are systematically shared with potential users fit well within this definition. As policy-makers seek better information to identify innovations and the systems that create them, OSS provides a window into innovation in all sectors of the economy. This is an aspect of innovation that has been relatively under-explored (Gault, 2018).

The contribution of this paper is two-fold. First, we present a framework for the presentation of statistics about OSS, categorizing it as a subcomponent of a recognized investment category in national economic accounts: own-account software. Second, we present and implement a prototype methodology for resource cost estimation for OSS investment that is conceptually consistent with current measurement of own-account investment. Our "bottom-up" methodology uses data collected from within OSS package code and data about OSS packages; we estimate that R, Python, Julia, and JavaScript packages currently available to users have a cumulative resource cost in the order of magnitude of $3 billion dollars. This method can be replicated with other OSS packages to better account for investment in these software tools. We also show how network analysis using these same data can be used to measure the relative impact of OSS packages. Linkages between OSS packages show how one package may depend upon another. Linkages are represented with outdegrees and are measures of the relative impact of frequently-used packages. This bottom-up approach will make it possible to identify the sector of contributors. Together, the cost and impact measures can provide indicators of research dollar outcomes, currently shown primarily with patents and bibliometric indicators.

Literature Review

A decade ago, interest in OSS centered around understanding the motivations for participation in OSS projects and evaluations of its growth potential. Surveys conducted in the early 2000s described the contributor community (Ghosh, et al. 2002; Boston Consulting Group 2002; David, Waterman, Arora 2003). While the motivations described for participation include skill development, creativity, and interest in the open source community, user-need and functionality consistently rate as critically important. A Boston Consulting Group survey (2002) probed the motivations for participating in OSS projects; over 60% reporting either work or non-work functionality.

For many academics and researchers, software tools and databases are by-products of their own work that can also be used by other academics as well (Gambardella and Hall 2005). Advantages of OSS include the ability to scale customization projects and to resolve program bugs quickly through many users (Lerner and Tirole. 2005). OSS communities can also be viewed as user innovation networks, where contributors more successfully develop solutions to their own software needs through the OSS community (von Hippel 2005).

Interest in better measurement of the economic impact of computer software and the increased digitization of knowledge led to parallel development in national economic accounting in many countries. For example, Gross Domestic Product (GDP) statistics for the US have treated computer software as investment since 1999, extending this treatment to research and development expenditures and entertainment and literary originals in 2013 (BEA 2013). Beyond these three categories, Corrado, Hulten, and Sichel (2005) provide a framework for consistent accounting for expenditures on intangibles that generate future benefits. Arguing that public expenditures yielding long-lived returns should be understood as investment, Corrado, Haskell, and Jona-Lasinio (2015) propose a public investment category, information, scientific, and cultural assets, which includes software and databases along with R&D, mineral exploration and cultural products. They argue that better accounting of public investment in intangibles would provide a more complete picture of economic growth (CSL, 2015).

[Figure 1 about here]

Figure 1 shows software investment in the US for 2016 (in current dollars) from the Bureau of Economic Analysis. Overall, software accounts for more than $400 billion dollars a year in fixed investment in 2016. Less than $100 billion of this was for public sector investment (federal labs and facilities, public universities, state and local government entities). Most investment accounted for, more than $300 billion dollars, is for private sector investment in software. For comparison, BEA reports investment in R&D on the same set of tables. Private sector R&D investment is $320 billion and the public sector (government)'s investment was $170 billion in 2016.[1]

[1] This paper was prepared prior to the release of BEA's comprehensive revision on July 27. The treatment was changed for R&D expenditures that overlap with own-account software expenditures, and thus data revised. [https://blog.bea.gov/2018/07/03/coming-july-27-gdp-statistics-updated-to-keep-pace-with-ever-changing-economy/] Overlap between R&D and software is accounted for within software prior to the revision, and after the revision is accounted for within R&D.

Software as investment has three types, based on data sources. These are prepackaged, custom, and own-account. Prepackaged and custom software are purchased inputs, and in national economic accounts industry receipts data are used for these estimates. Own-account software is not purchased or sold. For measurement purposes, it is new, or significantly-enhanced software created by business enterprises or government units for their own use and its value is estimated based on in-house expenditures for its creation (Parker and Grimm 2000).

From a classification perspective, this category of own-account software is where OSS belongs. OSS fits the definition of own-account software as laid out in BEA (2000) and OECD (2010) methodologies: OSS is original software that provides benefits by using the software directly, and these benefits are expected to last for years.   The key to its treatment as intangible investment is that resources shift from consumption to creating something that provides value in use over a long period of time to its creator. Open source software is a component of own account software. We expect to find measurable amounts in universities, both private and public, in institutions run by state and local governments, and in Federal investment.   Private university-created OSS would be within the private estimates, as would any created by private nonprofit institutions.

OSS and Related Definitions

Commits: With respect to OSS development, a commit is a submitted proposal for improvement to existing code.

Dependencies: The other packages that a particular package needs to use or install to function. In network analysis a dependency link between two packages i and j can be formulated as a directed edge from i to j (i → j) such that that the package j requires i to be installed to function.

Innovation: The implementation of a new or significantly improved product (good or service) or process, a new marketing method, or a new organizational method in business practices, workplace organization, or external relations. This definition is based on the Oslo Manual, which provides guidance for internationally comparable measurement (OECD and Eurostat 2005).

Intangible capital: As a subcategory of capital investment, intangible capital includes economically useful ideas, creations, or plans that can be reused repeatedly in production over a long period of time. Research and development expenditures, computer software and databases, and entertainment and literary originals are counted as intangible capital in national economic accounts. Additional categories of intangibles that have similar features are economic competencies, made up of brand equity, firm-specific human capital, and resources devoted to organizational development (Corrado, Hulten, and Sichel,2005).

Open source software: Computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. For this paper, we treat as open source any software language or package with an Open Source Initiative (OSI)-approved license.

Own account software: A category of software investment in national economic accounts. Own-account software is long-lasting software created for internal use, rather than as a market product.

Production Ready Release: A software package that is ready for production in its current ecosystem

Registry: A location that hosts, manages, and distributes OSS. The Comprehensive R Archive Network (CRAN) and PyPi for Python are examples.

Repository: A location that hosts and maintains different versions of software programs. GitHub is an example.

Empirical Framework

Counts and Linkages: The production and delivery of open source software through publicly accessible websites provide harvestable count and linkage data for software languages and packages. These data can be analyzed with methods developed for bibliometrics and patent analysis. Indicators of research activity and impact are calculated from databases covering scientific article publications and their citations. Many well-developed methodologies and extensions exist, and a research community continues to grow, invigorated by improved computing power and algorithms. For patents, an intersecting literature thrives around the use of data sets from patent offices. For software code, Ghosh (2002) describes methods of extraction, interpretation and analysis of empirical data from software source code that we apply and extend in this paper.

Cost of production: Investment in own-account software, created for internal use, is estimated for national investment statistics based on its cost of production. This approach is the same one used for other types of products not sold in the market. BEA researchers have proposed a similar methodology to measure advertising-supported products: when bundled with advertising, free content created in the business sector can be valued based on its production cost (Nakamura, Samuels, and Soloveichik 2018).

Production costs for own account software include those for analysis, design, programming and testing, and exclude maintenance and repair (Parker and Grimm, 2000). As originally described, cost of production is the sum of: labor costs and intermediate inputs (such as materials and supplies and overhead). In US economic statistics these costs are estimated with the mean wage rate for computer programmers and system analysts, adjusted for compensation costs, and multiplied by the number of these workers in each industry. Wage, employment and compensation are from the U.S. Bureau of Labor Statistics data. To account for time spent on tasks other than software development, BEA uses an adjustment ratio from a survey of software developers time spent on different tasks of 487 businesses (Boehm,1981). Non-labor costs for OSS development are estimated with industry production ratios (Parker and Grimm 2000).

Data Collection and Preparation

Keller et al (2018) describes the overall approach used here to explore data sources beyond surveys to improve and extend indicators of science and engineering activity and of innovation. This approach includes structured processes to discover, acquire, profile, clean, link, explore the fitness-for-use, and statistically analyze the data. Here we gather and use publicly available metadata about individual packages and their contributors, as well as information within the code.

The natural way to obtain the information about the development of an OSS project that is shared as packages is to inspect the repository that hosts the code for that language or application. The first step is to catalog all projects available to the programming language or application. Package managers are the tools used to discover, retrieve, and bring added functionality to users and developers. A package manager obtains the list of available packages and repository locations through a registry. A registry contains basic information such as a unique identifier (usually the package name), a release version to identify what version of the package should be retrieved, and the repository location (where to find the actual code). Registries collect this information from the package manifest file which in turn holds the project's metadata: name, author, maintainer, license, description, dependencies, project status, etc.

Our data collection strategy is illustrated in Figure 2. Based on our definition of OSS, we focus on projects that have an Open Source Initiative (OSI)-approved license and that are production ready, i.e., not in development stage. This definition determines the data to be collected.

[Figure 2 about here]

We use four registries for the analysis: (1) The Comprehensive R Archive Network (CRAN) at https://cran.r-project.org/web/packages/available_packages_by_name.html , (2) The Python Package Index (PyPI) at https://pypi.org/search/ , (3) METADATA.jl (the official registry for the Julia language) at https://pkg.julialang.org/, and (4) CDNJS (one of the most commonly used JavaScript content delivery systems) at https://cdnjs.com/.

Registries differ in multiple aspects that affect coverage and data duplication. For example, CRAN and CDNJS use a mirror system such that the registry itself hosts the repository. This makes it permanent and self-contained, but prone to duplication of information (e.g., multiple packages having the same repository in its metadata). In contrast, METADATA.jl only gives the package manager the location to obtain the repository. This allows the registry to synchronize the package metadata and history directly with the "truth" (based on the repository).

This feature comes at the cost of potentially purged data, in cases when repositories are no longer accessible. Registries such as CRAN and PyPi display dependencies, license, and continuous integration information from the manifest file. In contrast, CDNJS and METADATA.jl use a detached repository-oriented tracking system and do not contain this information.

Data collection starts with a unique identifier and the repository location of each package. For repositories hosted using *Git* version control on GitHub, the GitHub API provides access to additional relevant information. Two examples are identification of the package license in cases where it is not specified in the manifest file and, where applicable, the presence of continuous integration of small code changes. The latter helps identify production-ready packages. For some repositories, production-ready packages are straight-forward to identify. CRAN submissions include only production ready packages while other registries allow for developing stage projects or potentially fully depreciated and outdated packages (e.g., uncurated registries such as METADATA.jl).

For those packages that are identified as OSI-approved and have a current production ready registered release, we obtained the contributions for the top 100 contributors. This set of contributors is determined by weekly counts of lines added, lines deleted, total number of commits. We treat lines of code as a first order proxy for effort, though it can be quite noisy under certain scenarios. For example, this method in its current version does not distinguish between lines of code, documentation, generated output, data files, etc.

[Table 1 about here]

Data Description

Table 1 shows the number of packages we collected from GitHub for the four major open source software programming languages: R, Python, Julia and JavaScript. The number of packages collected for each language is given in the final column of Table 1. For the latest release of the package, we collected development information including the number of contributors, the lines of code (added and deleted), number and time of commits (suggested code edits from contributors), and profile of contributors.

The data demonstrate overall a power-law distribution for each language; most of the packages are developed with a small number of contributors (a core-team). Figure 3 illustrates the distribution of the number (in logs) of contributors of each package for each language.

[Figure 3 about here]

Next, we collect the lines of code for the top 100 contributors on GitHub. The distribution of the total lines of code for each language is given by boxplots in Figure 4.

[Figure 4 about here]

Finally, we compute the contributions to the packages by each developer (the top 100 contributors based on the number of commits made). We observe that most of the contributions are made by a small number of developers, as illustrated in Figure 5.

[Figure 5 about here]

For these four OSS languages we observe the same kind of power-law distribution found in other kinds of count-based distributions. The skew shows up in the findings that most contributions to packages are made by a small number of contributors. Articles and patents share features of skewed count-based distributions, where only a small number of articles or patents generate measurable impact, and those that have impact can be blockbusters. For bibliometrics for example, Bornmann and Leydesdorff found in an analysis of 2010 papers that nearly half of the citation impact is accounted for by the top 10% most-frequently cited papers (2017). Similarly, for patents only a small number of patents have high valuations, based on the market valuation of owner-firms (Hall, et al, 2000). This skewness for patents and papers also emerges in the analysis of OSS packages themselves. In an analysis of R packages, Korkmaz et al (2018) find 1) that while the median number of downloads for R packages is 8.5, the mean number is 45,775 and 2) for R packages that have citations, the median value is 0, while the average is 8.

Implementation

Dependency Network: When an OSS package requires the code of another package to do its work, that package is dependent on the other. We use reverse dependencies (reuse) of packages as a measure the impact and value. The greater the reuse, the greater the value. The dependency information is obtained from the manifest files described earlier. Outdegree, as described below, is a measure of this value. However, taken together, network analysis provides a fuller picture of impact.

We use the dependency interactions to study the connection between the structural features and the cost of the OSS projects. We generate the dependency network where a directed edge $i \rightarrow j$ indicates that the package j requires i to be installed to function. Packages with no edges (i.e., dependency links) are removed from the network. The main characteristics of the dependency network for all languages are given in Table 2, and the features are defined in the table caption.[2]

We observe that the R dependency network has a high number of connected components and communities compared to the other packages; it also has a higher number of packages. Figure 6 presents a subgraph of the R dependency network; this is the largest component after removing the small-size communities with less than 5% of the nodes. The size of the node is proportional to the out-degree, i.e., number of packages that reuses the package, and the different colors indicate communities (different uses of R) identified using the modularity algorithm implemented in Gephi (Bastian, Heymann, and Jacomy, 2009). As seen in the figure, ggplot2, which is one of the main R packages used for visualization, has a high outdegree. This high outdegree means that many packages use ggplot2. In fact, it is the package with the highest outdegree of 925, followed by Rcpp with an outdegree of 838.

[2] Note that the number of nodes (packages) for the languages are different than in Table 1. This is because the dependency information is collected from the manifest files that have information about more packages than those on GitHub.

[Figure 6 about here]

Cost of Creation: The creation cost of complete packages is estimated using data that we can collect from the R package code itself, and from registries and repositories, summed across completed packages. Our unit of analysis is a completed and released OSS package. We use cost model approaches from software engineering to estimate the input of OSS contributors to completed packages in person-months. We assume that the cost of contributors' time is roughly equivalent to the average wage for computer programmers plus additional intermediate input and capital services costs.

Cost estimation for software projects is a recurring topic in software engineering literature, motivated by the challenge of keeping large software projects on schedule and within budget (Sharma, Bhardwaj, Sharma, 2011). Estimation models emerged first in the 1960s and evolved from a linear function of the number of instructions. However, as software projects grow, experience shows that effort increases nonlinearly. Cost estimation models have evolved that account for complexity, reliability, and scale in a variety of ways based on characteristics of the product, the platform, the contributors, and the project. Examples of these estimation models include Constructive Cost model COCOMO II, the Putnam Software Life Cycle Management model, and models based on function points (Boehm and Valerdi, 2008).

The logic of the constructive cost model is that:

$$Production\ time\ in\ PersonMonths = Calibration\ factor\ x\ lines\ of\ code\ x\ effort\ multipliers$$

The calibration factor represents the person months needed for a set number of lines of code, unadjusted for effort factors. The effort multipliers account for complexity, reliability, and scale for these models; they lead to increased cost. In COCOMO II for example, the effort factors are based on qualitative assessments for each attribute with ratings from very low to extra high.

Translating this approach to our data on OSS, the package specific data we collected provides lines of code (added and deleted by each contributor) for each completed package. We use the COCOMO II calibration factor (Boehm, et. al, 2000) to estimate person-hours per package. The formula below shows the parameters we used, selected for the organic software class which consists of software dealing with a well-known programming language and a small, but experienced team of contributors.

$$\begin{aligned} Effort &= 2.4(KLOC)^{1.05} \text{ Person-months} \\ Nominal\ development\ time &= 2.5(Effort)^{0.38} \text{ Months} \\ Development\ cost &= Monthly\ wage \times Nominal\ development\ time \end{aligned}$$

KLOC stands for kilo (thousand) lines of code. With these person-month calculations per OSS package, we estimate a resource cost by multiplying by monthly wages for programming occupations plus additional costs. Table 4 shows the steps and data sources for the estimation. To summarize our method, we assume that the input time of contributors is roughly equivalent to the average wage for computer programmers (from Bureau of Labor Statistics (BLS) 2017 Occupational Employment Survey data) plus additional intermediate input and capital services costs (from Bureau of Economic Analysis (BEA) 2007 Input Output table data). The per person month cost for OSS contribution is obtained as $18,096, which is the amount used in our cost calculations.

[Table 4 about here]

Results

With our estimates of the cost of all packages that use R, Python, Julia and JavaScript, we obtain ***power-law distributions*** for all languages. The density plots and the boxplots are given in Figure 7. We view these as order of magnitude estimates. The total costs of all packages for each language are as follows: R ($854 million for 3,396 packages), Python ($747 million for 3,804 packages), Julia ($239 million for 1,324 packages), and JavaScript ($1,199 million for 3,213 packages). Summing costs for these four languages yields over $3 billion dollars in total costs, based on 2017 wage rates. We view this as a lower bound because not all JavaScript packages are on CDNJS. Further, while this amount represents costs based on U.S. wages, not all of this development took place in the U.S.

We report the top packages with the highest total cost in Table 4. Many of these packages are those used for web-development (e.g., googleAnalyticsR, Django-workon, Selenium, Webkit.js). We add the node properties of these packages from the dependency network to the creation cost to understand whether there is a connection between the cost and the value. Table 4 presents the top packages with the highest out-degree, i.e., the number of other packages that depend on the package, and other network features (defined in the table caption) together with the creation cost of these packages.

We do not observe a consistent pattern between cost and value based on the dependency network features. This is also illustrated with the correlation heatmap in Figure 6, which shows that the correlation between the log(cost) of packages and network characteristics is low and positive for most of the features. The heatmap illustrates the Pearson coefficients (the color scale given in the legend) between the network features of the packages/nodes (centrality measures) and their cost.[3] We observe that there is a positive correlation (given in blue) between most of

[3] Correlation coefficients takes values between -1 and 1, a value of 1 (and -1, respectively) indicates a perfect positive (negative) linear correlation, and a value of zero indicates that there is no linear relationship between the variables of interest.

the centrality measures and the cost, however the correlation coefficients are lower than 0.4 indicating a low correlation between these variables. The highest positive correlation (a coefficient of 0.36) is between the cost and indegree of the nodes, implying that the cost of the packages that depend on more packages are likely to be higher. In-degree (the number of used packages) has the highest positive correlation with cost for all the languages.

Further Work

To illustrate the contribution to OSS from public spending, more detailed breakdown by sector is needed, both for relative measures based on counts and uses, and for dollar-denominated cost measures. That level of detail would show the contributions of public sector institutions, households, businesses, academia, and other nonprofits.

We will also need to validate or understand differences between our prototype estimates and existing statistics. One way to validate the estimates would be in comparison with national accounts statistics for own-account software at the economic sector level. Figure 9 shows the categories of investment in national income accounts that include OSS.

Our prototype estimates used one simple set of parameters in the conversion of lines of code to person-months. We can explore ways to use characteristics of the contributors, the languages, and the packages to create bounds around the costs that better account for heterogeneity and our blunt assumptions.

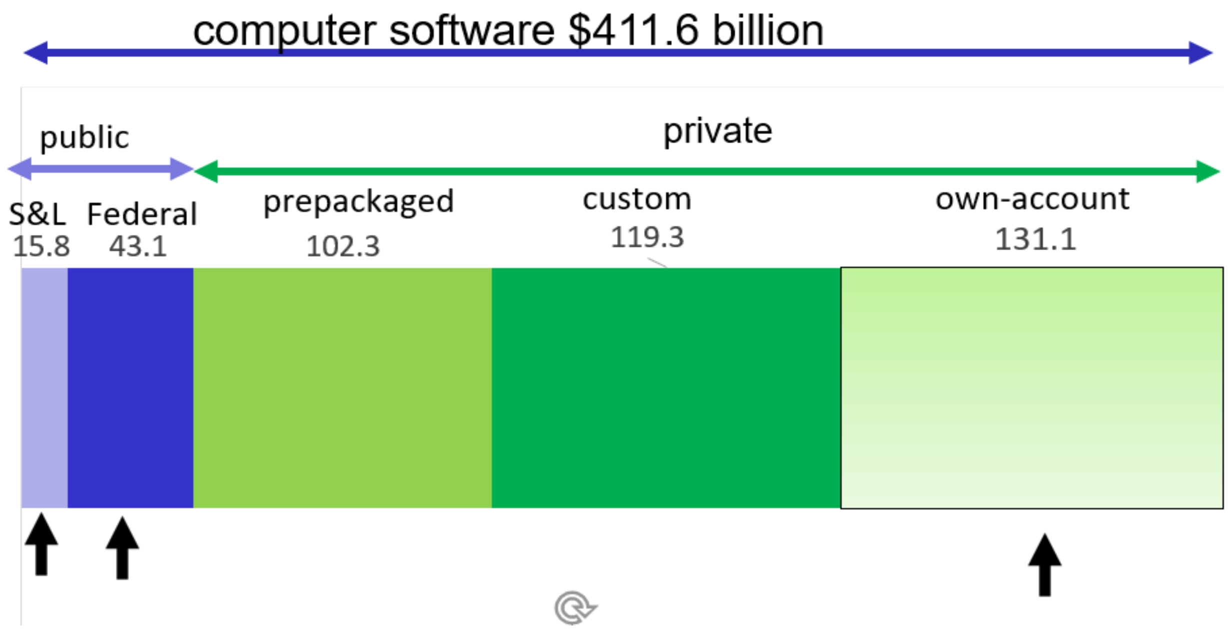
Beyond the cost estimates, we have these questions to address:

• Flow Measures: How much is created each year?
• Categories: What types can be identified?
• Sectors and collaborators: Who creates it?
• Users: Beyond the developers of OSS, who benefits from its development?

The resource cost estimates we present here include contributions over multiple years, estimated based on costs as though they had all occurred in 2017. Next steps will be to use information about the date when a package is made available to create year-by-year flow estimates, using cost deflators. Repositories and registries also include information about functions and use of each package that allow us to categorize the packages. For example, the open source resource SourceForge provides categories for projects that include communications, system administration, games, audio and video, business and enterprise, and security and utilities. (Keller et. al. 2018). Contributor-specific information on business, government, and university affiliations will allow us to develop preliminary sector measures. As yet, we have not identified a non-survey method to determine how much open source software is used by entities in each sector of the economy who have not participated in its development.

Conclusion

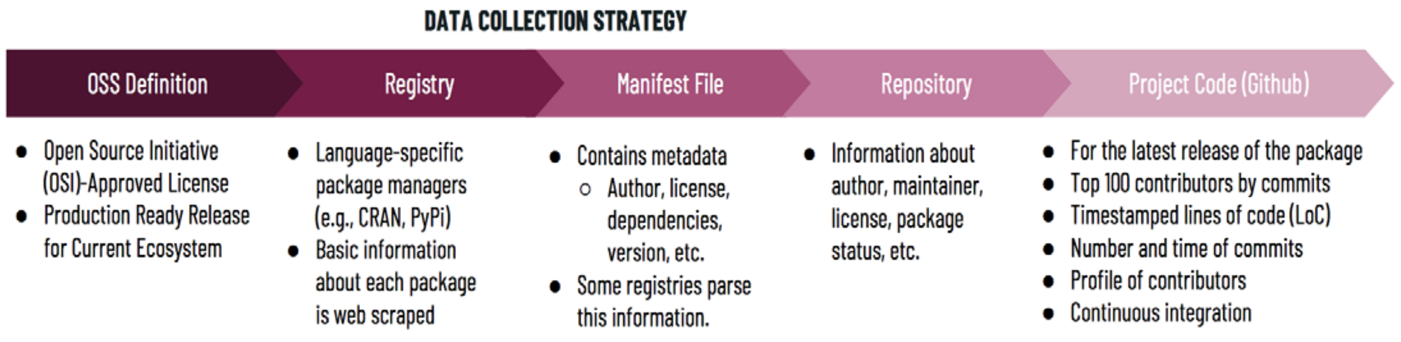
In this paper we have described a methodology to use freely-harvestable data about OSS packages to develop statistics on the scope and use of OSS languages. These non-survey data provide characteristics of OSS packages and their contributors that are used in two different ways. With lines of code to estimate effort, we use a method derived from the measurement of own account software in national economic accounts to estimate an order of magnitude resource cost. We estimate the resource cost of producing R, Python, Julia, and JavaScript exceeds $3 billion dollars, based on 2017 costs. Network analysis provides a framework for estimating relative impact. Outdegrees, a measure of package reuse, provide an indicator of impact or value and we find it is not highly correlated with cost. However, a more complete picture of impact results from a fuller set of network parameters.



Note: S&L: State and Local government Federal: Federal Government

Source: U.S. Bureau of Economic Analysis. Intellectual Property Products Fixed Asset Tables (private) and Investment in Government Fixed Assets (Table 7.5B), accessed, May 2018.

Figure 1 US Investment in Computer Software, 2016



OSS Definition
- Open Source Initiative (OSI)-Approved License
- Production Ready Release for Current Ecosystem

Registry
- Language-specific package managers (e.g., CRAN, PyPi)
- Basic information about each package is web scraped

Manifest File
- Contains metadata
  - Author, license, dependencies, version, etc.
- Some registries parse this information.

Repository
- Information about author, maintainer, license, package status, etc.

Project Code (Github)
- For the latest release of the package
- Top 100 contributors by commits
- Timestamped lines of code (LoC)
- Number and time of commits
- Profile of contributors
- Continuous integration

Figure 2. The Data Collection Strategy

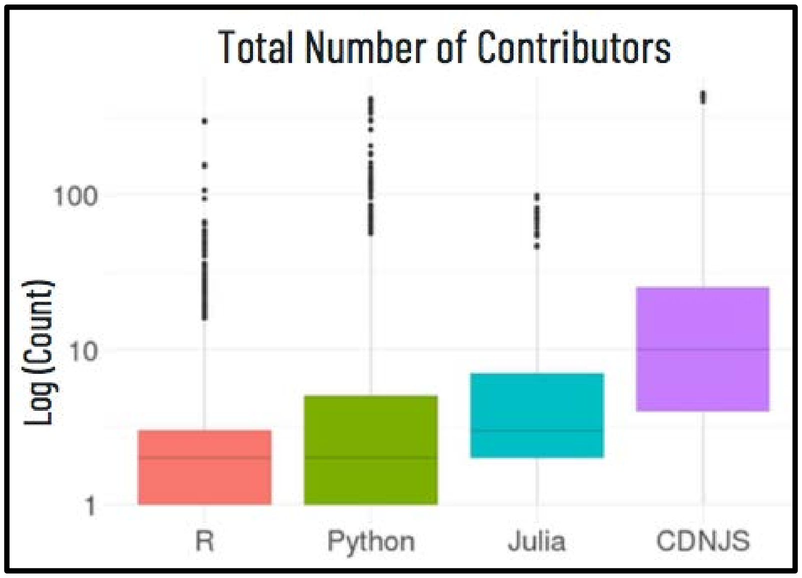


Figure 3. Distribution of the Number of Contributors to Each Language’s Packages (Log scale)

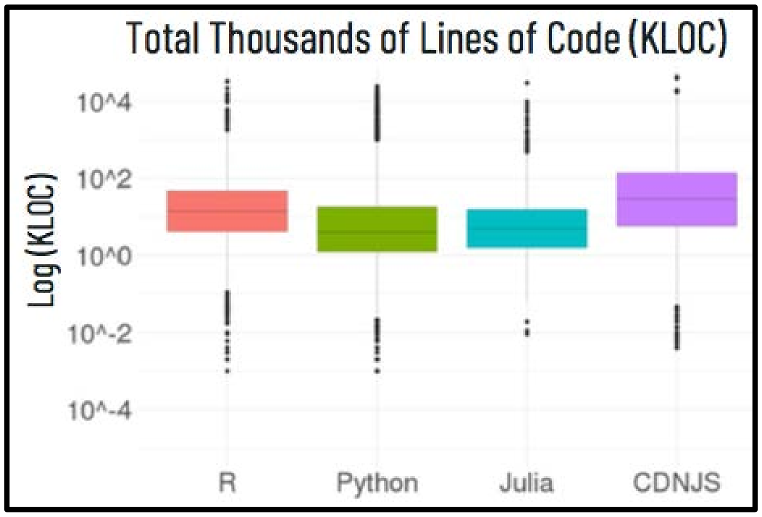


Figure 4. Distribution of the Total Lines of Code in Each Language’s Packages (Log Scale)

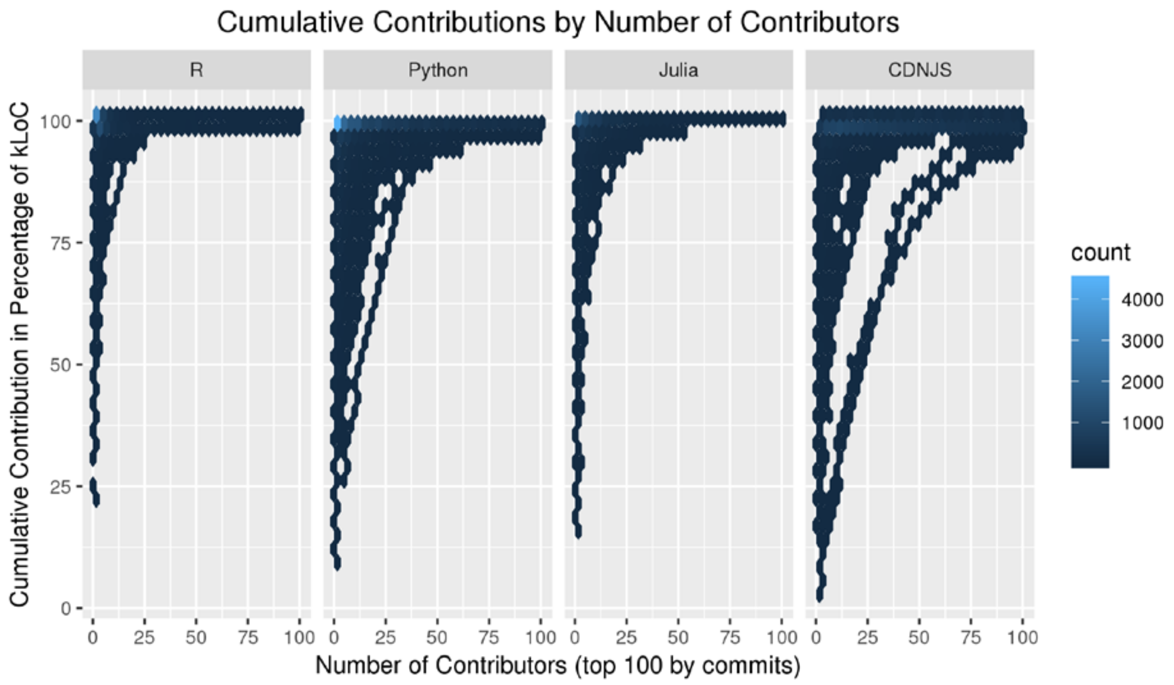


Figure 5. The Cumulative Contributions by Top Developers Based on the Number of Commits.

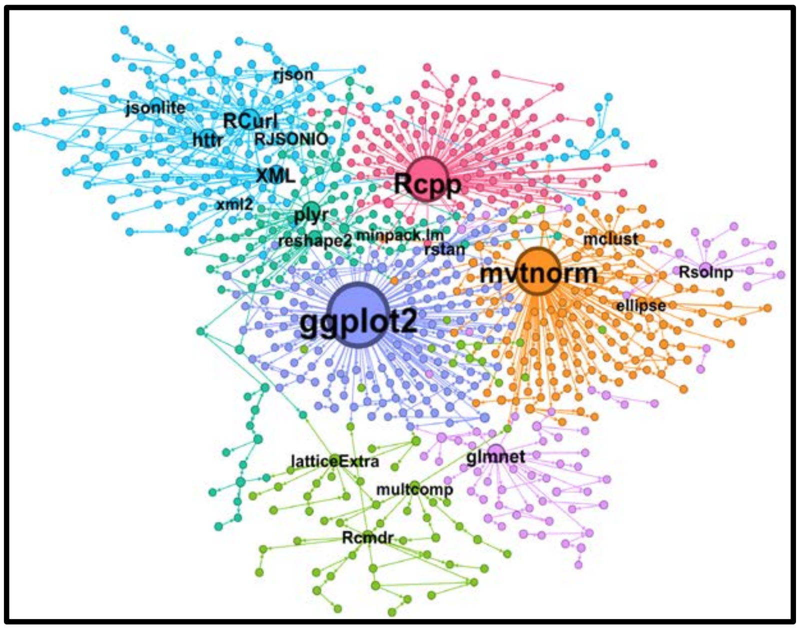


Figure 6. A subgraph of the R dependency network.

The size of the node is proportional to the out-degree, i.e., number of packages that reuses the package, and the different colors indicate communities (different uses of R) identified using the modularity algorithm (Blondel, et al. 2008) implemented in Gephi (Bastian, Heymann, and Jacomy (2009). The small communities (less than 5%) are removed for illustrative purposes.

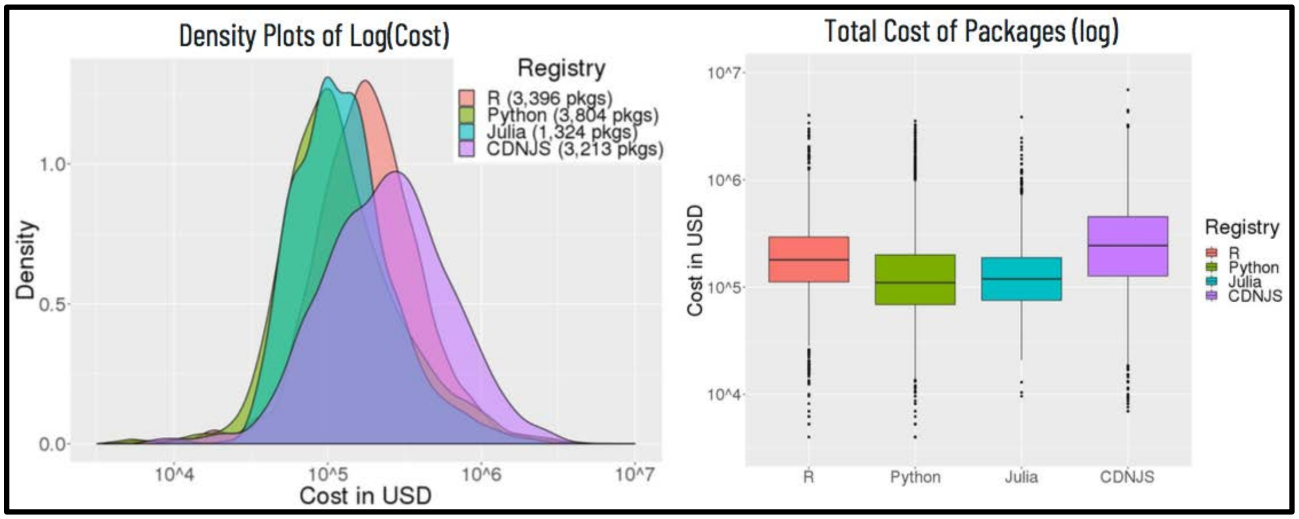


Figure 7. The Distribution of Total Cost of Packages for each Language, Density and Boxplots.

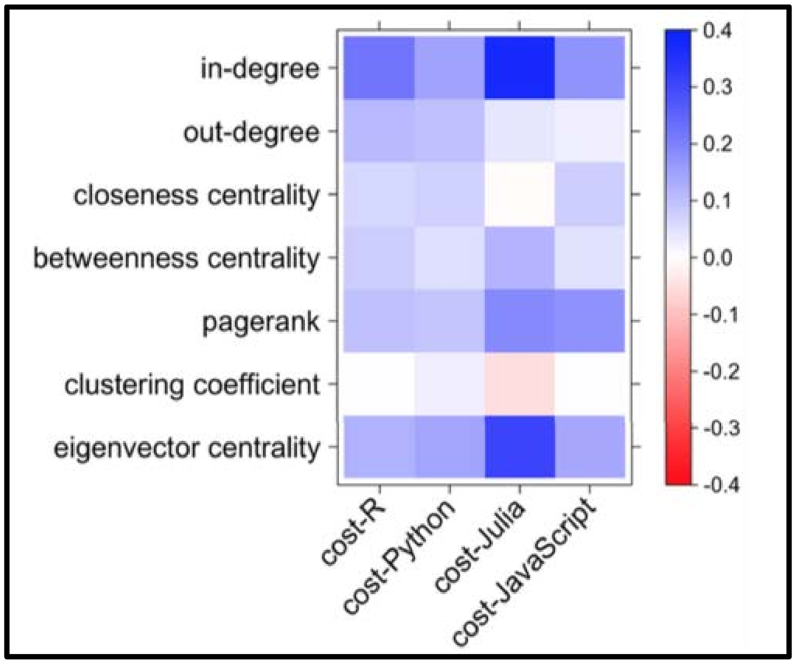
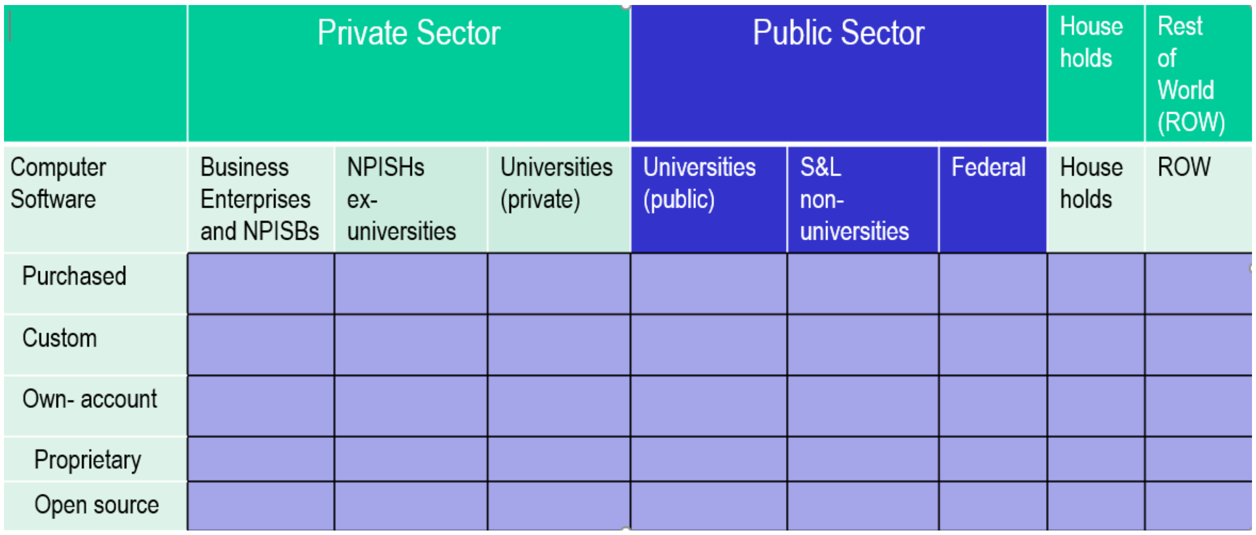


Figure 8. Heatmap of the Pearson Correlation between the log(cost) and network characteristics of packages.



	Private Sector			Public Sector			House holds	Rest of World (ROW)
Computer Software	Business Enterprises and NPISBs	NPISHs ex-universities	Universities (private)	Universities (public)	S&L non-universities	Federal	House holds	ROW
Purchased								
Custom								
Own- account								
Proprietary								
Open source								

Notes NPISH: Nonprofit institutions serving businesses NPISB: Nonprofit institutions serving households

Figure 9. Open Source Software is a Subcomponent of Own Account Software Investment

Data Collection				
Language	Package manager	Number of packages	OSI-approved & production ready	Packages on Github
R	CRAN	12,614	11,886	3,396
Python	PyPi	143,047	7,392	3,804
Julia	Pkg.jl	2,040	1,324	1,324
JavaScript	CDNJS	3,367	3,367	3,213

Table 1. Number of Packages for Four Major OSS programming languages: R, Python, Julia and JavaScript

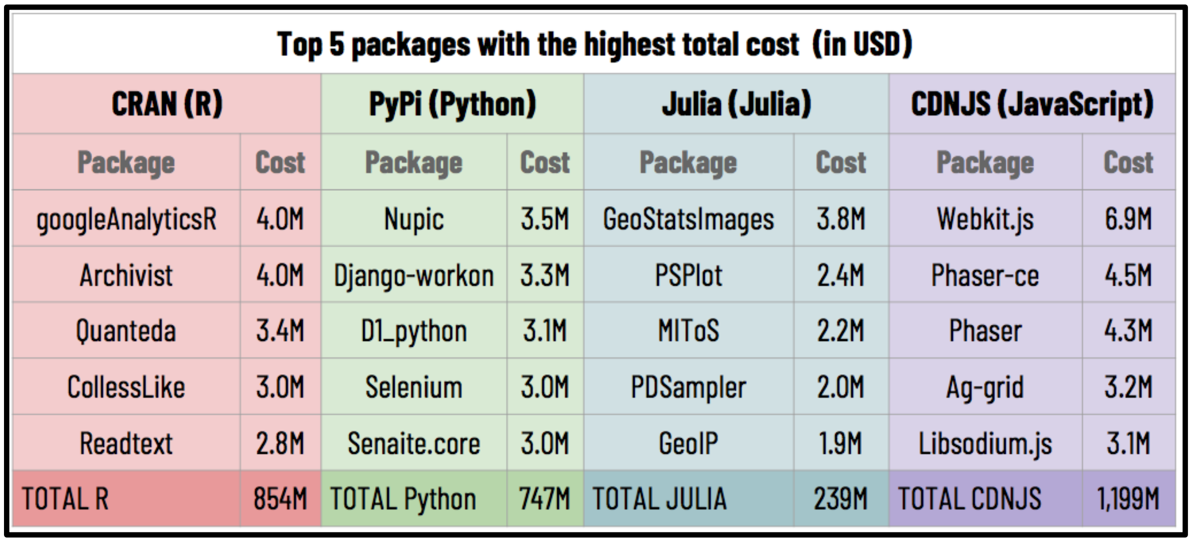
Dependency Network Characteristics							
Language	#Nodes	#Edges	Average degree	Diameter	# Connected Components	Largest Component Size	# Communities (Modularity)
R*	6,684	22,024	3.29	7	334	6,303 (94.3%)	351
Python	2,419	3,020	1.25	4	102	2,113 (87.3%)	133
Julia	1,711	5,440	3.18	7	11	1,691 (98.8%)	24
JavaScript	1,279	2,591	2.02	8	13	1,255 (98.1%)	25

Table 2. Characteristics of the dependency network for all languages of the study

Notes: Average degree (indegree and outdegree) indicates the average number of packages that they depend on (and are used by). Diameter is the shortest distance between the two most distant nodes in the network. (Weakly) Connected components are subgraphs such that there is an undirected path from every pair of nodes in the subgraph. Communities are detected using modularity algorithm [4] that identifies densely connected subgraphs of the network. *R dependency network includes both dependencies and imports. Standard libraries that are supplied with R (e.g., stats, utils, graphics) are removed from the network.

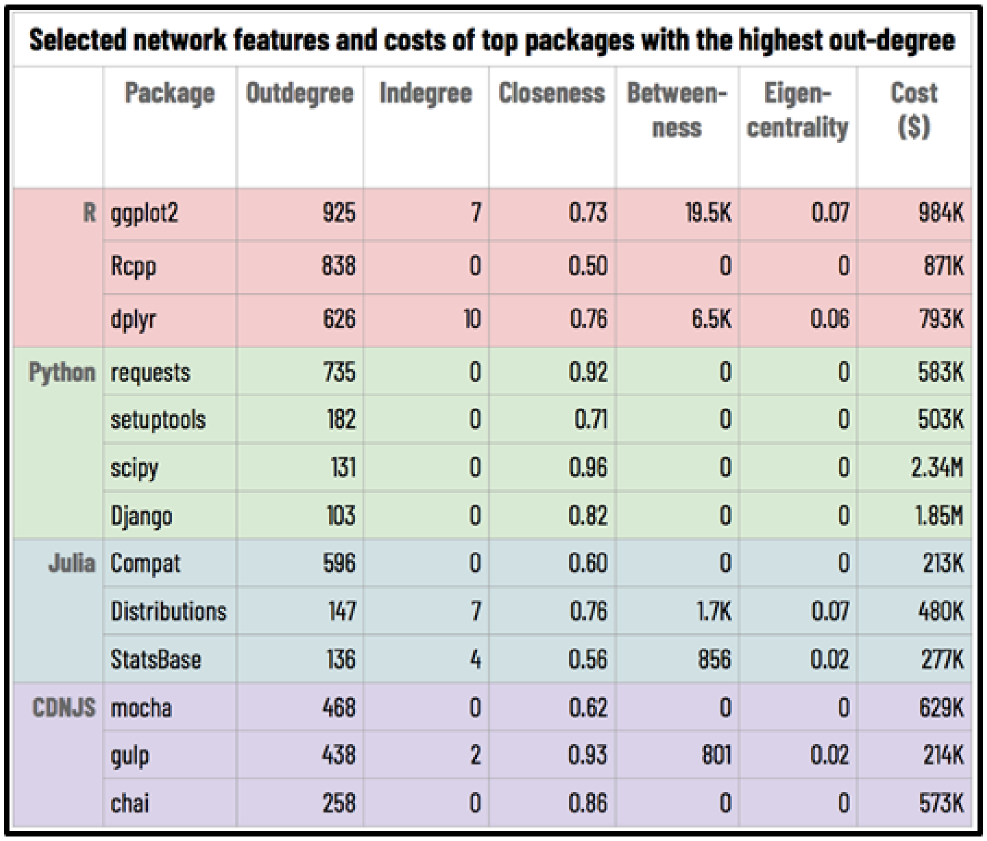
Cost Component	**Data Source**	**Cost in Dollars**		**Cost Factor**
		per person year		
Labor Compensation Cost		136,909		Applied to wages
Wage and Salary Rate	Mean Annual Wage for Software Developers and Applications, 2017, BLS		106,710	
Non-wage compensation	Estimated based on the inverse ratio of wages and salaries to total compensation for professional and technical services industries; Four quarter average for 2017, BLS		30,199	28%
				Applied to compensation
Intermediate input cost	Intermediate input ratio to compensation for computer systems design and related services from the 2007 Use Table, BEA		80,249	59%
Capital services (GOS)	Ratio to compensation for computer systems design and related services from the 2007 Use Table, BEA		22,405	16%
Estimated resource cost				
without gross operating surplus		217,157.69		
with gross operating surplus		239,562.62		
		per person month		
without gross operating surplus		18,096.47		
with gross operating surplus		19,963.55		

Table 3. Source Data for OSS Contributor's per Month Resource Cost



Top 5 packages with the highest total cost (in USD)							
CRAN (R)		PyPi (Python)		Julia (Julia)		CDNJS (JavaScript)	
Package	Cost	Package	Cost	Package	Cost	Package	Cost
googleAnalyticsR	4.0M	Nupic	3.5M	GeoStatsImages	3.8M	Webkit.js	6.9M
Archivist	4.0M	Django-workon	3.3M	PSPlot	2.4M	Phaser-ce	4.5M
Quanteda	3.4M	D1_python	3.1M	MIToS	2.2M	Phaser	4.3M
CollessLike	3.0M	Selenium	3.0M	PDSampler	2.0M	Ag-grid	3.2M
Readtext	2.8M	Senaite.core	3.0M	GeoIP	1.9M	Libsodium.js	3.1M
TOTAL R	854M	TOTAL Python	747M	TOTAL JULIA	239M	TOTAL CDNJS	1,199M

Table 4. Top 5 Packages Based on Total Cost for Each Language



Selected network features and costs of top packages with the highest out-degree

	Package	Outdegree	Indegree	Closeness	Between-ness	Eigen-centrality	Cost ($)
R	ggplot2	925	7	0.73	19.5K	0.07	984K
	Rcpp	838	0	0.50	0	0	871K
	dplyr	626	10	0.76	6.5K	0.06	793K
Python	requests	735	0	0.92	0	0	583K
	setuptools	182	0	0.71	0	0	503K
	scipy	131	0	0.96	0	0	2.34M
	Django	103	0	0.82	0	0	1.85M
Julia	Compat	596	0	0.60	0	0	213K
	Distributions	147	7	0.76	1.7K	0.07	480K
	StatsBase	136	4	0.56	856	0.02	277K
CDNJS	mocha	468	0	0.62	0	0	629K
	gulp	438	2	0.93	801	0.02	214K
	chai	258	0	0.86	0	0	573K

Table 5. Network Features and Costs of Top Packages with Highest Out-Degree

The network features of the top packages with the highest out-degree. Definitions: Outdegree (indegree, respectively) is the total number of outgoing (incoming) links, .i.e., the number of packages that depend on (are required by) the package. Closeness centrality measures how close a node is to every other node. Betweenness is a measure of being connected to other nodes that are not connected to each other (as a bridge). PageRank depends on (i) the number of links the node receives, (ii) the number of links given out by its neighbors, (iii) the centrality of its neighbors. Clustering coefficient quantifies the degree to which a node's neighbors are connected. Eigencentrality of a node considers the centrality of its neighbors.

References

Bastian, M., Heymann, S., & Jacomy, M.,2009. Gephi: An open source software for exploring and manipulating networks. ICWSM 8(2009), 361-362.

Blondel, V. D., Guillaume, J., Lambiotte, R., and Lefebvre, E. 2008. "Fast unfolding of communities in large networks." Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P1000

Boehm, Barry. 1981. Software Engineering Economics (Englewood Cliffs, NJ: Prentice-Hall, 1981): 533-35, 548-50.

Boehm, Barry, Abts, C., Brown, W. Chulani, S, Clark, B. Horowitz, E., Madachy, R, Reifer, D. and Steece.B., 2000. Software Cost Estimation with COCOMO II (with CD-ROM). Englewood Cliffs, NJ:Prentice-Hall

Bornmann, Lutz, and Leydesdorff, Loet, 2017. "Skewness of citation impact data and covariates of citation distributions: A large-scale empirical analysis based on Web of Science data," Journal of Informetrics, Volume 11, Issue 1, Pages 164-175

Boston Consulting Group 2002. Survey of Free Software/Open Source Developers. [http://cmaptools.cicei.com:8002/rid=1203015502582_177114975_1313/BCG-HACKERSURVEY.pdf]

Corrado, C. Haskel, J. & Jona-Lasinio, C. 2015 Public Investment in Intangible Assets. EPWP #15 – 01 Economics Program, [https://www.conference-board.org/pdf_free/workingpapers/EPWP1501.pdf]

Corrado, Carol, Charles Hulten, and Daniel Sichel, 2005, "Measuring Capital and Technology: An Expanded Framework," in Measuring Capital in the New Economy, Carol Corrado, John Haltiwanger, and Dan Sichel, editors, University of Chicago Press.

Bureau of Economic Analysis. 2013. Preview of the 2013 Comprehensive Revision of the National Income and Product Accounts: Changes in Definitions and Presentations. Survey of Current Business, March 2013.

David, P. Waterman A,. & Arora, S, 2003. "The Free/Libre/Open Source Software Survey For 2003," Draft Working Paper. September 2003.

Gambardella, Alfonso and Bronwyn H. Hall. 2005. "Proprietary vs Public Domain Licensing of Software and Research Products." NBER Working Paper 11120.

Ghosh, Rishab Aiyer. 2003. Clustering and Dependencies in Free/Open Source Software Development: Methodology and Tools

Ghosh, R. A. R. Glott, B. Krieger, and G. Robles. 2002. *Free/Libre and Open Source Software: Survey and Study Report. Part IV.* [http://www.infonomics.nl/FLOSS/report/]

Ghosh, Rishab Aiyer, 2007. “Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU." Maastricht: UNU-MERIT, 2007.

Ghosh, Rishab Aiyer.2005 “Understanding Free Software Developers: Findings from the FLOSS Study,” in the volume, *Perspectives on Free and Open Source Software*, edited by Feller, Fitzgerald, Hissam, and Lakhani, MIT Press, reprinted 2014.

Glänzel, Wolfgang, n.d.   “A Concise Introduction to Bibliometrics & its History” https://www.ecoom.be/en/research/bibliometrics Accessed July 26, 2018.

Ihaka, Ross. 1998. “R: Past and Future,” (https://cran.r-project.org/doc/html/interface98-paper/paper.html) Accessed July 30, 2018.

Hall, B. H., A. B. Jaffe, and M. Trajtenberg, 2000. "Market Value and Patent Citations: A First Look,” NBER Working Paper 7741.

Hall, B. H., A. B. Jaffe, and M. Trajtenberg, 2001. "The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools." NBER Working Paper 8498

Hall, Bronwyn H. and Adam B. Jaffe, 2018, "Measuring Science, Technology, and Innovation: A Review", Annals of Science and Technology Policy: Vol. 2: No. 1, pp 1-74. ttp://dx.doi.org/10.1561/110.00000005

Hoffa, Felipe, 2018. “The top contributors to GitHub (2017)” (https://datastudio.google.com/reporting/0ByGAKP3QmCjLU1JzUGtJdTlNOG8/page/Q3DM) Accessed July 31, 2018

Gault, Fred, 2018. “Defining and measuring innovation in all sectors of the economy,” Research Policy, Vol.47:3, 617-622.

Keller, S., G. Korkmaz, C. Robbins, and S. Shipp. 2018. “New Opportunities to Observe and Measure Intangible Inputs to Innovation: Definitions, Operationalization, and Examples.” Manuscript under review for publication, July 2, 2018.

Korkmaz G, Kelling C, Robbins C, and Keller, S, 2018. “Modeling the Impact of R Packages Using Dependency and Contributor Networks.” Conference Paper, ASONAM 2018

Lerner J. and Tirole J, 2005. “Economic Perspective on Open Source” in the volume: Perspectives on Free and Open Source Software, edited by Feller, Fitzgerald, Hissam, and Lakhani, MIT Press reprinted 2014

Leonard Nakamura, Jon Samuels and Rachel Soloveichik, 2016. Valuing “Free” Media Across Countries in GDP BEA Working Paper WP2016-3

Leonard Nakamura, Jon Samuels, and Rachel Soloveichik, 2017. “Measuring the ”Free” Digital Economy within the GDP and Productivity Accounts.” BEA Working Paper WP2017-9

Moulton, B., Parker, R., and Seskin E. 1999. “A Preview of the 1999 Comprehensive Revision of the National Income and Product Accounts: Definitional and Classificational Changes”. Survey of Current Business, August 1999.

Moylan, Carol. n.d. Estimation of Software in the U.S. National Accounts: New Developments https://www.bea.gov/papers/pdf/USSoftware.pdf. Accessed July 26, 2018.

OECD and Eurostat, 2005. *Oslo Manual: Guidelines for Collecting and Interpreting Innovation Data, third edition.* OECD Publishing.

OECD 2010. *Handbook on Deriving Capital Measures of Intellectual Property Products*. OECD Publishing.

Open Source Initiative, 1998. (https://opensource.org/osd).

Parker R. and Grimm B. 2000. “Recognition of Business and Government Expenditures for Software as Investment: Methodology and Quantitative Impacts, 1959-98.” (https://www.bea.gov/papers/pdf/software.pdf)

Scott, T, and Rung, A. 2016. “Memorandum for Heads of Departments and Agencies: Federal Source Code Policy: Achieving Efficiency, Transparency, and Innovation through Reusable and Open Source Software.” M-16-21, (https://code.gov/#/policy-guide/policy/introduction) Accessed July 20, 2018.

Sharma, T., Bhardwaj, A, and Sharma, A. 2011. “A Comparative Study of COCOMO II and Putnam models of Software Cost Estimation. International Journal of Scientific and Engineering Research,” Volume 2, Issue 11, November 2011.

GlobalStats statcounter, 2018. “Operating System Market Share Worldwide Operating System Market Share Worldwide - June 2018,” (http://gs.statcounter.com/os-market-share) Accessed July 20, 2018.

Von Hippel, Eric A., 2005. “Open Source Projects as “User Innovation Networks” in Perspectives on Free and Open Source Software, edited by Feller, Fitzgerald, Hissam, and Lakhani, MIT Press 2014

Von Hippel, Eric A., and Georg von Krogh, 2003. “Open Source Software and the \Private-Collective" Innovation Model: Issues for Organization Science." Organization Science 14, no. 2 (2003): 209-223.

W3Techs. 2018. “Usage statistics and market share of Apache for websites.” https://w3techs.com/technologies/details/ws-apache/all/all. Accessed July 23, 2018.

Wickham, Hadley. n.d. http://hadley.nz Accessed July 26, 2018