Open Source Software as Intangible Capital: Measuring the Cost and Impact of Free Digital Tools

Preliminary Draft October 31, 2018[1]

Carol A. Robbins*(1), Gizem Korkmaz (2), José Bayoán Santiago Calderón (3), Daniel Chen (2), Claire Kelling (4) , Stephanie Shipp (2), Sallie Keller (2)

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, yet its value and impact are not currently measured (with small exceptions). We develop an approach to document the scope and impact of open source software created by all sectors of the economy: businesses, universities, government research institutions, nonprofits, and individuals. We use a bottom-up approach to measure subset of OSS projects and languages, collecting data on open source software languages R, Python, Julia, and JavaScript, as well as from the Federal Government's code.gov website.

Using lines of code and a standard model to estimate package developer time, we convert lines of code to resource cost. We estimate that the resource cost for developing R, Python, Julia, and JavaScript exceeds $3 billion dollars, based on 2017 costs. Applying this approach to open source software available on code.gov results in an estimated value of more than $1 billion, based on 2017 costs, as a lower bound for the resource cost of this software. We analyze the dependencies between software packages through network analysis and estimate re-use statistics. This reuse is one measure of relative impact.

Key words: Open Source Software, Intangibles, Network Analysis

National Center for Science and Engineering Statistics, National Science Foundation; 2) Social & Decision Analytics Division, Biocomplexity Institute & Initiative, University of Virginia; 3) Claremont Graduate University; 4) Pennsylvania State University

[1] An earlier version of this paper was presented August 21, 2018 at the International Association for Research on Income and Wealth. [http://www.iariw.org/copenhagen/robbins.pdf]

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

The views expressed in this paper are those of the authors and not necessarily those of their respective institutions.

CR and GK designed and performed the analysis and took the lead in writing of the paper. JC, DC, CK collected the data and contributed analysis tools. All authors contributed to the writing of the paper. The authors 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

.Introduction and Contribution

Open source software allows free access to digital tools and are a part of intangible investment with the qualities of public goods. Beginning in the early 1980s, open source software (OSS) projects have provided users with an unknown amount of freely modifiable software tools and other useful products that are used for work and for leisure. OSS projects are those where the underlying computer source code is made available with a license for which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. Popular OSS licenses include the MIT license, the GNU General Public License, the Apache License and the ISC (Open Source Initiative) License.[2] While these kinds of software in recent years are generally downloaded from the internet for free, for many years OSS software was shared via disks or other physical media.

These tools include LaTeX typesetting program introduced initially in 1983, LAMP web services (Linux, Apache, MySQL, and PHP); Mozilla, Firefox web browser, and the WordPress content management system. OSS gaming software includes games and free tools to make games. Freeciv, is a multiplayer game where players use technology to create and conquer; GDevelop is an open source tool that allows users to create their own games. OSS has penetrated markets in areas including operating systems, servers, and specialized 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 HTTP server on the internet (W3Techs 2018). Based on the cost of the nearest available

[2] The BlackDuck repository lists the licenses most frequently used on its site: https://www.blackducksoftware.com/top-open-source-licenses

substitute, Greenstein and Nagle (2013) estimated the value of capital stock of Apache software in use in 2013 at between $2 and $12 billion.

While OSS projects can compete directly with proprietary software, as in the Apache example, OSS projects are increasingly used in combination with proprietary code, either as extensions of a proprietary base, or with proprietary software extending the open source base. These cases highlight the potential for complementarity between open source and proprietary software. Top institutional contributors to the code repository GitHub in 2017 are Microsoft and Google.

However, OSS is created and used outside of the business sector as well. 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 OSS.

We do know that many of the most widely used OSS tools 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 in part while its creator was at the University of Helsinki. 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. Private nonprofit institutions such as Stanford University, the Massachusetts

Institute of Technology, The Broad Institute, and University of California at Berkeley are among the top contributors to OSS on Github, a popular repository.

Our contribution to measuring economic welfare in the digital age is to focus on one aspect of digital products that are produced both in the market and outside of the market. We are measuring the value and use of OSS using a familiar sectoral framework and cost estimation approach from economic accounting, thus adapting current methods in new ways. We do three things in this paper. First, we present preliminary data on OSS shared publicly by the U.S. government on code.gov, including counts of OSS projects by government organization and lines of code. Second, we present a framework for the presentation of statistics about OSS, categorizing it as a subcomponent of two recognized investment categories in national economic accounts: own-account software and custom software. Third, we estimate resource cost for OSS investment that is conceptually consistent with current measurement of own-account software investment in national accounts. Our "bottom-up" methodology uses data collected from within OSS package code and data about OSS packages. 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 represent how one package depends upon another, and the outdegree of a package (the number of packages that require the package to function) can be used as a measure of the relative impact of those that are frequently-re-used. Together, the cost and impact measures provide potential indicators of research dollar outcomes, currently shown primarily with patents and bibliometric indicators.

Related Literature

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; Lakhani, et al. 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. Lakhani and co-authors (2002) probed the multiple motivations for participating in OSS projects; over 40% reported intellectual stimulation and reported improvement of skills. Functionality, either for work or non-work motivated over 60% of contributors at that time.

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).

Open source code share by the US Government

A lot of re-useable code is first created as part of the ongoing work of the Federal Government. This can happen in the performance of a Federal contract or other Government-funded activity. As of late July 2018, more than 4000 separate software code projects are shared for reuse on code.gov. The projects are shared as part of an effort to make custom-developed code broadly available across the Federal Government, including publishing as OSS when appropriate. Twenty-six Federal departments, agencies, and other entities have contributed open source code on projects that have start dates between 2009 and 2018.

Most, but not all, of these projects listed on code.gov provided links to the repository to download code. We used the applications programming interface (API) provided by code.gov to collect data on projects associated with 26 Federal government agencies. We obtained a total of 4,457 projects listed on code.gov as of July 30, 2018, and 4,051 of these projects included a URL to a code repository. These repositories include well-known ones such as Github, SourceForge, and Bitbucket, as well as webpage repositories run by units of the Federal government, such as those run by NASA and Sandia National Lab.

Table 1 shows the number of projects on code.gov for the Federal government. The third column shows the number of projects for each organization that included a link to a Github repository location where we obtained development information by project. We collected this detailed information on Github for 2,977 of these projects, yielding detailed data for 67% of the projects on code.gov. Limiting our analysis to those projects that started before January 1, 2018, we have data for 2,688 projects, or 60% of all projects on Code.gov. The data we collected include number of lines of code, both additions and deletions, contributors, and commits. These 2,688 projects contain 2.5 billion lines of code, represent 950,000 commits, and over 8,000 contributors.

Open Source Projects by Federal Government Organization as Posted on Code.gov

for projects started before January 1, 2018

Organization Name	Total Projects on Code.gov	Number of Projects linked to Github Collection	Kilo-lines of code (kloc)	Commits	Number of Contributors
Total	4,457	2,688	2,486,209.95	950,625	8,292
General Services Administration	1,501	1,368	266,860.27	318,676	4,631
Department of Energy	899	704	1,219,834.85	485,726	2,433
Consumer Financial Protection Bureau	261	243	753,447.22	49,781	334
National Aeronautics and Space Administration	998	141	179,916.58	51,936	358
Environmental Protection Agency	156	61	14,327.45	4,711	78
Department of Labor	72	52	5,663.03	1,185	50
Department of Homeland Security	19	19	8,051.15	2,526	63
National Archives and Records Administration	19	19	3,001.92	9,324	50
National Security Agency	19	19	5,133.72	3,725	36
Department of Agriculture	21	11	3,499.21	4,361	23
Department of Defense	13	10	5,555.56	4,965	46
National Science Foundation	98	6	54.42	28	2
Department of Transportation	7	5	994.28	1,903	15
Department of Health and Human Services	27	4	193.55	654	25
Department of Justice	6	4	33.25	240	1
Department of Veterans Affairs	4	4	2,294.33	429	28
Small Business Administration	16	4	397.37	1,058	13
Executive Office of the President	4	3	501.40	549	25
Office of Personnel Management	3	3	5,134.72	2,022	18
Agency for International Development	5	2	2.21	10	1
Department of Commerce	3	2	0.18	14	4
Department of the Treasury	2	2	11,177.75	6,007	24
Department of Education	2	1	110.60	787	33
Social Security Administration	129	1	24.92	8.00	1
Department of Housing and Urban Development	172	0			
Nuclear Regulatory Commission	1	0			

Table 1. Summary Counts by Government Organizations on code.gov

By number of separate projects, the General Services Administration contributed the most, with 1,501 projects, while based on lines of code, the Department of Energy (DOE)

contributed the most, with 1.2 billion lines of code. One DOE project, *Raven*[3], accounts for 10.6 million of the more than 2.5 billion lines of code. *Raven* is statistical software for risk analysis in nuclear reactor systems. Another DOE project is *Qball*[4], with 9.5 million lines of code, which uses molecular dynamics to compute the electronic structure of matter. These projects include both code developed within the Federal Government and code created through contracting.

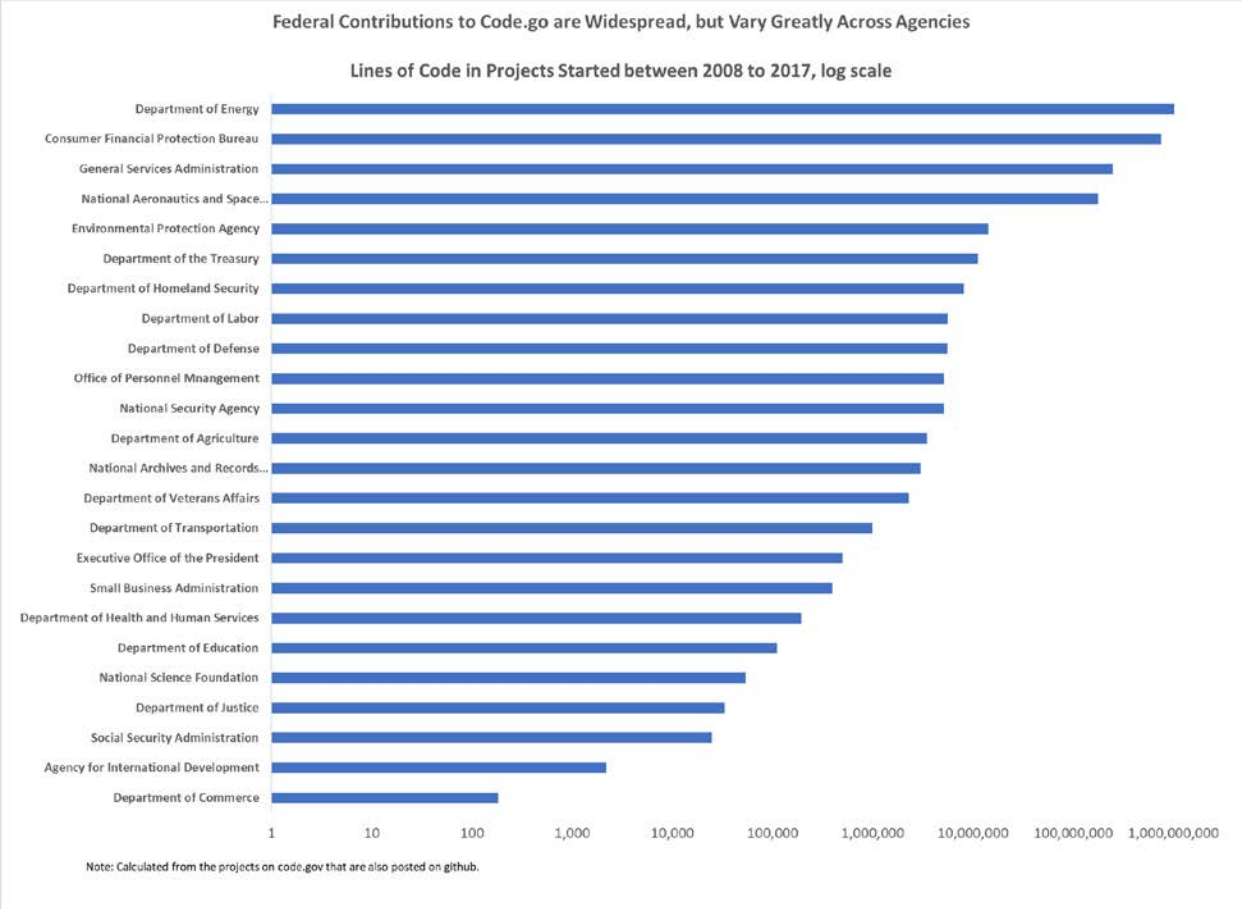

Figure 1. Contributions to Code.gov by lines of code.

[3] https://raven.inl.gov/SitePages/Overview.aspx
[4] (https://github.com/LLNL/qball

OSS and Economic Measurement

In economic accounts of intangible investment and intellectual property products, software investment is measured in three types, based on available data sources for each type. These are prepackaged, custom, and own-account. Prepackaged and custom software are purchased inputs, and in national economic accounts, industry receipts and government budget data are used for these estimates.

To make sense of the cost and impact estimates we are trying to develop, we need a framework to add things up using standards that are broadly used, but can be adapted to highlight OSS. The table presents broad economic sector categories that connect directly to existing measures of market-based software. Figure 2 extends BEA's software measurement framework in two directions, by sector and by proprietary vs OSS.

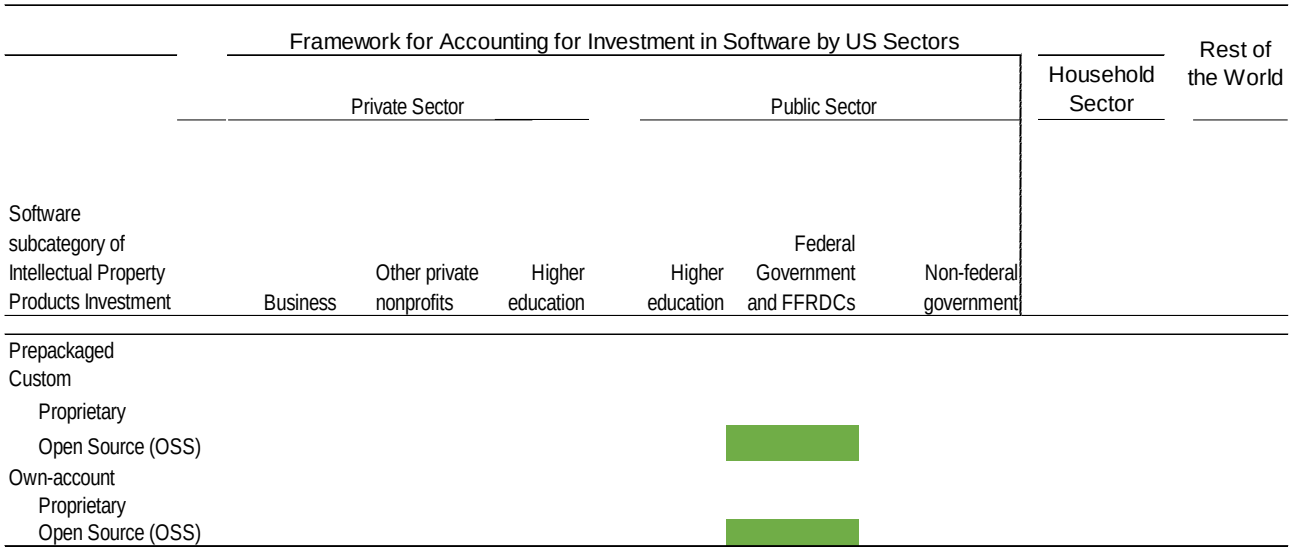

Figure 2. Economic Sectors and Measurement Categories for the Production Software

OSS fits into two subcategories of software investment, custom and own account. Figure 2 shows in green where OSS intersects custom software and own-account software for the Federal Government. Federal government purchases of custom created software are shared as OSS. When created inhouse, OSS would fit within the own-account measurement category: 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).

Empirical Framework for Cost Measurement

The description of data from Code.gov provides a preview of what information from code repositories may be used to create cost estimates for OSS, including that which is created internally. The sections below on counts, linkages and cost of production summarize the work by others that we build on for our data analysis.

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 BEA's 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 2017 Occupational Employment Statistics (OES) program. To account for time spent on tasks other than software development, BEA uses an adjustment ratio from a survey of software developers on time spent on different tasks for 487 businesses (Boehm 1981). Non-labor costs for OSS development are estimated with industry production ratios (Parker and Grimm 2000). As of July 2018, the sum of costs now also includes the value of capital services for own account software (Kelly, McCulla, and Wasshausen, 2018).

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.

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/.

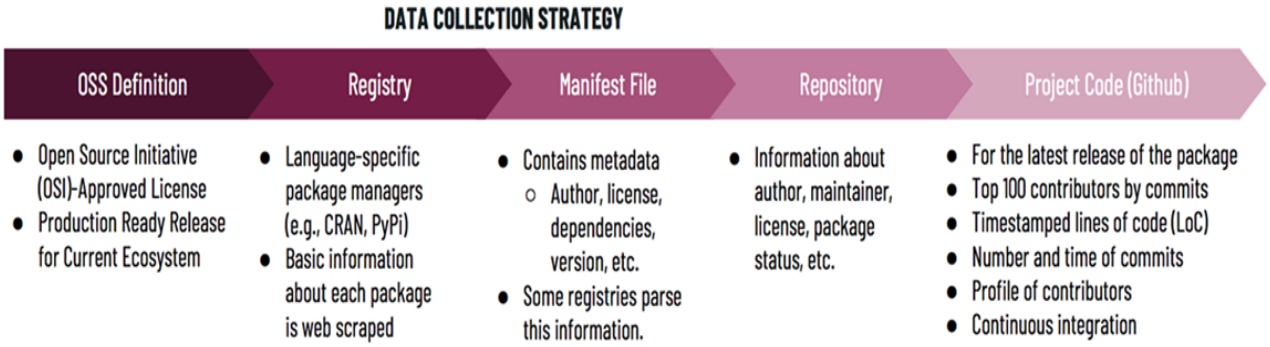

Figure 3. Data Collection Strategy

Our data collection strategy is illustrated in Figure 3. 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.

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. 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, and 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.

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 2. Languages and Packages Collected

Data Description

Table 2 shows the number of packages we collected from GitHub for the four major OSS programming languages: R, Python, Julia and JavaScript. The number of packages collected for each language is given in the final column. 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.

For these four OSS languages we observe the same kind of power-law distribution found in other distributions of digital and knowledge products.[5] 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,

[5] . 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 paper published in 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.

the mean number is 45,775 and 2) for R packages that have citations, the median value is 0, while the average is 8.

Cost and Impact Estimates

The resource cost approach we use is modeled after the method described earlier that is used to estimate own-account software and other kinds of own-account investment in economic statistics. In the US, those own-account software investment measures are created using data from the BLS Occupational Employment Survey to obtain the mean wage rate for computer programmers and system analysts other input costs, multiplied by the number of these workers in each industry in a given time period (Parker and Grimm 2000). This industry-level estimate provides a top-down measure of own-account investment. Our approach starts with a software package or project and works up from the bottom.

Cost of Creation: For the four OSS languages, 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. The unit of analysis is a completed and released OSS package and lines of code, both additions and deletions, are the measure of effort. Using lines of code and a cost model approach from software engineering we estimate person-months of OSS contributors to completed packages. 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. The model allows for these parameters to be adjusted based on additional data.

$$\boldsymbol{Effort = 2.4(KLOC)^{1.05}}$$

$$\boldsymbol{Nominal\ development\ time = 2.5(Effort)^{.38}}$$

$$\boldsymbol{Development\ cost = Monthly\ wage\ x\ Nominal\ development\ time}$$

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. Appendix Table 1 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 $19,963, which is the amount used in our cost calculations.

Order of Magnitude Resource Cost Calculations for OSS Languages

Using this approach, the total costs of all packages for each language are as follows: R ($942 million for 3,396 packages), Python ($824 million for 3,804 packages), Julia ($264 million for 1,324 packages), and JavaScript ($1,323 million for 3,213 packages). Summing costs for these four languages yields over $3 billion dollars in total costs, based on 2017 wage rates. US wage rates were used, though clearly contributors come from many countries.

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	4.4M	Nupic	3.9M	GeoStatsImages	4.2M	Webkit.js	7.6M
Archivist	4.4M	Django-workon	3.6M	PSPlot	2.7M	Phaser-ce	4.9M
Quanteda	3.7M	D1_python	3.5M	MIToS	2.5M	Phaser	4.7M
CollessLike	3.3M	Selenium	3.3M	PDSampler	2.2M	Ag-grid	3.6M
Readtext	3.1M	Senaite.core	3.3M	GeoIP	2.1M	Libsodium.js	3.5M
TOTAL R	942M	TOTAL Python	824M	TOTAL JULIA	264M	TOTAL CDNJS	1,323M

Table 3. Order of Magnitude Cost Estimates.

We report the top packages with the highest total cost in Table 3. Many of these high

resource cost packages, such as googleAnalyticsR, and Webkit.js, are used for web-development.

Estimating Impact through Dependency Networks: OSS is generally distributed without cost, and so standard market measures of revenue cannot provide an impact measure. We use 'reuse of packages' as a measure of the impact and value. The greater the reuse, the greater the value. When an OSS package requires the code of a second package to do its work, the first package is dependent on the second. For example, an R package for statistics as well as an R package for inventory may be both be dependent on a separate graphing package. This reuse, through multiple dependencies, increases the graphing package's impact. The dependency information is obtained from the manifest files described earlier. Outdegree, as described below, is a measure of this value. Taken together with other measures 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 4, and the features are defined in the table caption. 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.

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 4. Dependency Network Characteristics

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.(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.

Figure 4 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). [6]

[6] We identified these using the modularity algorithm implemented in Gephi (Bastian, Heymann, and Jacomy 2009).

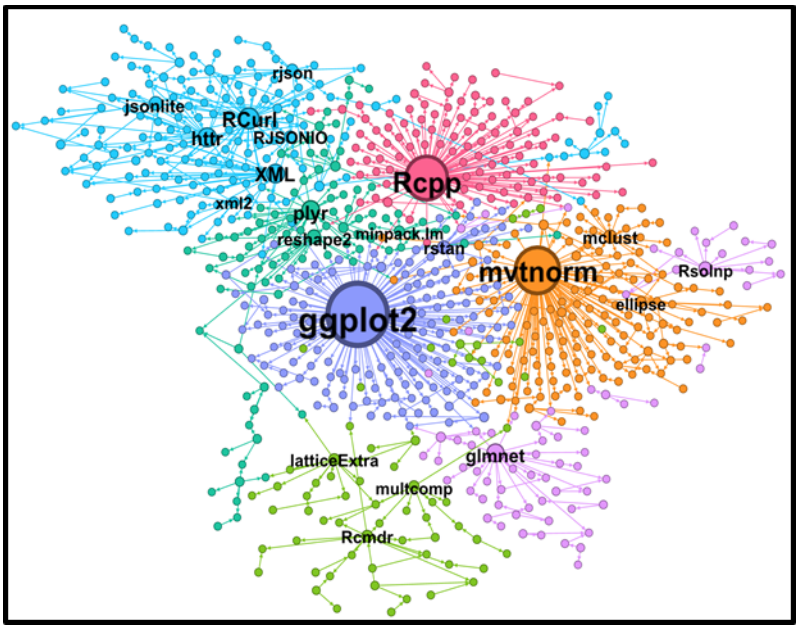

Figure 4. A subgraph of the R dependency network

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.

Table 5 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 find that ggplot2, requests, Compat and mocha have the highest number of packages that depend on them in the languages R, Python, Julia and CDNJS, respectively. This outdegree measure, the dependency linkage between two packages, is described further below.

Closeness centrality measures how close a node is to every other node in the network, betweenness is a measure of being connected to other nodes that are not connected to each other

(as a bridge) and eigencentrality of a node takes into account the centrality of its neighbors. When we compare these values to the cost of these packages, we do not observe a consistent pattern between cost and value based on the centrality measures.

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	1.08M
	Rcpp	838	0	0.50	0	0	960K
	dplyr	626	10	0.76	6.5K	0.06	875K
Python	requests	735	0	0.92	0	0	643K
	setuptools	182	0	0.71	0	0	555K
	scipy	131	0	0.96	0	0	2.58M
	Django	103	0	0.82	0	0	2.04M
Julia	Compat	596	0	0.60	0	0	235K
	Distributions	147	7	0.76	1.7K	0.07	530K
	StatsBase	136	4	0.56	856	0.02	306K
CDNJS	mocha	468	0	0.62	0	0	694K
	gulp	438	2	0.93	801	0.02	236K
	chai	258	0	0.86	0	0	633K

Table 5. Cost Estimates and Network Centrality (Impact) Measures.

Cost and Impact

The dependency relationships between packages with a OSS language allow us to explore the relationship between the estimated cost and impact. The absence of a strong relationship 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.

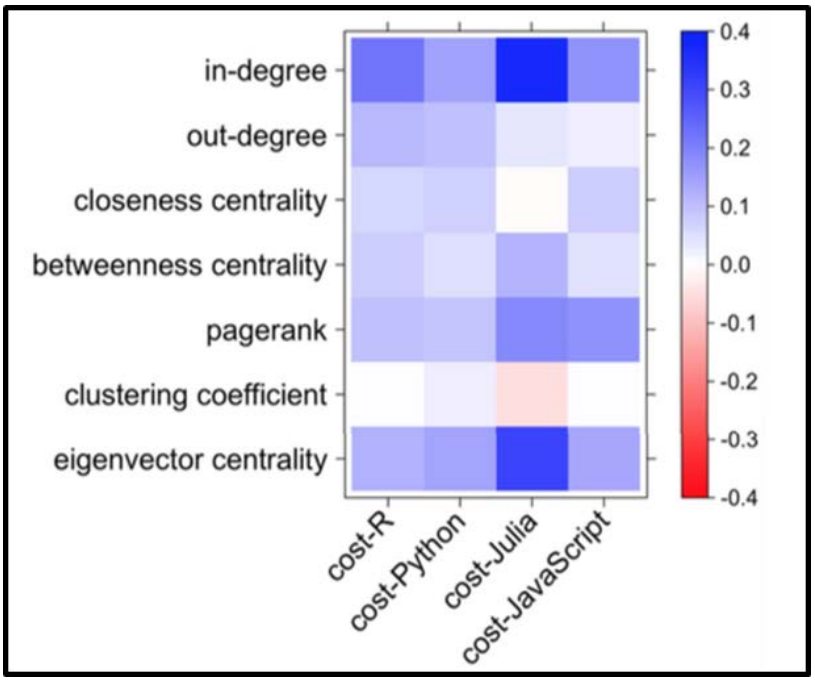

Figure 5. Heatmap of Pearson Correlation between log(cost) and network characteristics of packages.

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.[7] We observe that there is a positive correlation (given in blue) between most of 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.

[7] 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.

Apply Cost Approach to Code.gov

Using the cost approach described above for OSS packages, we estimated the cost this implies for the OSS projects shared through Github on Code.gov. Since many of the projects contributed by Government organizations have been developed by contractors as custom software, this resource cost is not an estimation of what the Government actually paid for these software projects. Rather, it gives an order of magnitude cost estimate consistent with own-account software that allows comparison with the OSS language packages described in the previous section.

The quantity inputs are the same –kilo-lines of code and we have not yet integrated other variables into the cost estimation. Using the OSS language parameters, working in well-known programming language with a small, experienced team of contributors, we estimated a resource cost for these 2.5 billion lines of code at about $1.1 billion dollars, calculating all contributions at the rate of 2017 costs. The code contributed by the DOE accounts for about $480 million of that total, and the GSA accounted for about $330 million. This is a partial estimate of all the contributed projects, because our calculation is only for those projects on Github.

Putting these values in context, BEA reports that private investment in software in 2017 was $352.9 billion dollars, composed of $147.6 billion in prepackaged software, $141.1 billion for custom software and $64.3 billion dollars for own-account software. Government investment measures for 2017 are not yet available from BEA, however, for 2016 Government investment in software was $43.1 billion dollars for Federal and $15.8 billion dollars for State and Local, the category that includes investments by public universities (BEA, 2017.)

Further Work

Further work has three parts:

1) Use harvested information to link OSS contributors with institutional affiliations into meaningful economic sectors. 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 have these questions to address:
    a. Flow Measures: How much is created each year?
    b. Categories: What types can be identified?
    c. Sectors and collaborators: Who creates it?
    d. Users: Beyond the developers of OSS, who benefits from its development?

2) Scale up and extend our collection of OSS project and package data to additional repositories, including adding to the Code.gov estimates in this paper to include data on projects that are not linked to Github. Our prototype estimates used one simple set of parameters in the conversion of lines of code to person-months. Through additional data collection, 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 compared with our blunt assumptions.

3) 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.

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. This is a bottom-up approach that we apply to a handful of OSS languages as well as to 60% of OSS software shared by Federal agencies on Code.gov projects. The ultimate goal is to measure the value and impact of OSS in the economy. These data provide characteristics about OSS packages and their contributors that are used in two different ways.

- With lines of code to estimate effort, we use a modification of a national economic accounts method to estimate an order-of- magnitude resource cost.
- We estimate a resource cost of producing R, Python, Julia, and JavaScript exceeds $3 billion dollars, based on 2017 costs.
- For OSS shared by the US Federal Government on Code.gov, we estimate a lower bound resource cost of over $1 billion dollars.

We conclude with network analysis to provide a nonmarket measure of impact, reuses of code. Using this we identify software packages that are high impact.

.

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

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.

Bureau of Economic Analysis. 2017. Table 7.5B. Investment in Government Fixed Assets, Last Revised August 23, 2017. [https://www.bea.gov/].

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.

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.” Publication forthcoming, November 1, 2018.

(Kelly, McCulla, and Wasshausen, 2018. “Improved Estimates of the National Income and Product Accounts Results of the 2018 Comprehensive Update,” Survey of Current Business, September, 98:9.

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

Lakhani, K., Wolf, B., Bates, J., and DiBona, C. 2002. Boston Consulting Group. Survey of Free Software/Open Source Developers.

[http://cmaptools.cicei.com:8002/rid=1203015502582_177114975_1313/BCG-HACKERSURVEY.pdf]

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.

OECD/Eurostat (2018), Oslo Manual 2018: *Guidelines for Collecting, Reporting and Using Data on Innovation*,4th Edition, The Measurement of Scientific, Technological and Innovation Activities, OECD. Publishing, Paris/Eurostat, Luxembourg. https://doi.org/10.1787/9789264304604-en

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

Appendix Table for Cost Estimation

Per month Resource Cost for Software Developers						
Cost Component	Data Source	Cost in 2017 Dollars			Cost Factor Applied to Wages	Cost Factor Applied to Compensation
		per person year		per person month		
Labor Compensation Cost	Total compensation for professional and technical services industries; for 2017, BLS [https://www.bls.gov/news.release/eci.t05.htm]	136,909				
Wage and Salary Rate	Mean Annual Wage for Software Developers and Applications, 2017, BLS [https://www.bls.gov/oes/2017/may/oes_nat.htm]		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 [https://www.bls.gov/news.release/eci.t05.htm]		30,199		28%	
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)	[https://www.bea.gov/industry/input-output-accounts-data]		22,405			16%
Estimated resource cost						
without gross operating surplus		217,157.69				
with gross operating surplus		239,562.62				
without gross operating surplus				18,096.47		
with gross operating surplus				19,963.55		