Measuring the Impact of Open Source Software Innovation Using Network Analysis on GitHub Hosted Python Packages

Derek Banks
School of Data Science
University of Virginia
Charlottesville, VA, USA
dmb3ey@virginia.edu

Camille Leonard
School of Data Science
University of Virginia
Charlottesville, VA, USA
cvl7qu@virginia.edu

Shilpa Narayan,
School of Data Science
University of Virginia
Charlottesville, VA, USA
smn7ba@virginia.edu

Nicholas Thompson,
School of Data Science
University of Virginia
Charlottesville, VA, USA
nat3fa@virginia.edu

Brandon Kramer,
Edge and Node
New York, NY, USA
brandonleekramer@gmail.com

Gizem Korkmaz,
The Coleridge Initiative
Arlington, VA, USA
gizem.korkmaz@coleridgeinitiative.org

***Abstract*—Open Source Software (OSS) is computer software that has its source code publicly 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. Despite its extensive use, reliable measures of the scope and impact of OSS are scarce. In this paper, we focus on packages developed for Python programming language as it is one of the most widely-used languages mainly due to its flexibility and simple syntax that makes its framework easy to learn and share. We aim to develop a framework to measure the impact of Python packages listed on Package Index (PyPI.org). We use data from GitHub repositories (where these packages are developed) to obtain information about their development activity e.g., lines of code. Our goal is to identify influential actors, e.g., packages, developers, countries by using the impact measures. We use network-based and OSS-based measures such as number of downloads. Network-based statistics include centrality measures such as degree, and eigenvector centrality. Moreover, we calculate the cost of OSS as intangible capital using the COCOMO II model [1] to determine the cost of development and study the relationship between development cost and impact of Python projects. The findings show that the number of downloads for a package are correlated with the centrality statistics, supporting the hypothesis that the most influential are the most downloaded as well. We show which packages are saving on development cost by leveraging dependencies. This framework and measures can be applied more broadly to the OSS ecosystem and contribute to the National Science Foundation (NSF) policy indicators for measurement of innovation.**

***Index-Terms*: python, open source, open-source, github, network**

I. Introduction

Open Source Software (OSS) is software with source code that anyone can inspect, modify, and enhance [2]. This allows for the code to be copied, learned from, altered, and shared, creating an easy-to-access and powerful tool for developers who wish to utilize it. OSS therefore provides its users with additional control, security, and stability, as users are able to examine the code directly, and have additional developers ensure that the code does not contain security or design flaws. This in turn provides an additional sense of training and community to developers, as it allows them to work together to create better software.

Many OSS projects create long-lived tools that are often outputs of public spending, a kind of freely shareable intangible asset which, in many cases, is developed outside the business sector and subsequently used within the business sector. The scale and use of these modifiable software tools highlights an aspect of technology diffusion and flow that is not captured in market measures. Measures of the creation and use of OSS would complement existing science and technology indicators on peer-reviewed publications and patents that are calculated from databases covering scientific articles and patent documents. Many well-developed methodologies and extensions exist, and a research community continues to grow, invigorated by improved computing power and algorithms. We are motivated to better account for both the scale of OSS overall, as well as the contribution of public spending to investments in open source software, a vital component of science activity [3].

In this paper, we focus on conducting network modeling of OSS licensed Python packages from PyPI.org. These are packages that are installed through Python package managers such as pip, and are therefore made available to the public. Utilizing this package ecosystem, we attempted to model the overall impact of a package through a variety of different measures. We primarily measured these impact statistics through the number of times a package had been downloaded,

the cost of the package as determined by the COCOMO II model [1], and the network centrality statistics of the associated package.

We find that the packages with highest centrality measures are also the most downloaded and therefore influential. Additionally, the development cost does not predict the impact or influence a package will have after its completion.

The rest of the paper is organized as follows. The next section provides background information into our choice of the programming language Python for open-source analysis. Section 3 summarizes the previous network analysis work that this paper was built on. Section 4 describes the process of data collection and merging that was required to begin building our dependency networks. Section 5 details the methods we used for network analysis, calculating costs, and our observations of relationships between centrality measures, costs, and reuse of packages. In Section 6, we conclude and suggest future work that can be done to take this analysis forward.

II. Background

The popularity of programming languages frequently changes as new programming languages gain developer attention, while older ones fall out of favor. With over 700 different programming languages [4], there is clearly no straightforward choice when choosing which one to utilize, as it often depends on what someone is trying to accomplish. In this paper, we focus primarily on one of the most popular programming languages: Python.

Python, successor to the ABC language (Wikipedia contributors 2018), was developed by a Dutch programmer at the National Research Institute for Mathematics and Computer Science in the Netherlands (Wikipedia contributors 2018), in the late 1980s [5]. It is one of the most widely used programming languages mainly due its simple syntax which makes code easy to learn and share while also being quite flexible [6]. Developers utilize PyPI (Python Package Index) to contribute their work and distribute their software as packages to other developers.

In this paper, we analyzed Python, and the distribution of OSS across PyPI as it is a continuously growing ecosystem that is gaining steam, with strong relevance towards Data Science, a rapidly growing field of study. According to the TIOBE Index as of March 2022 [7], Python is the number one most used programming language, just recently passing C and Java for the top spot.

Not only is Python the most popular programming language for general use, Python proves itself to be incredibly Open-Source friendly as well. Python is consistently ranked a close second to Javascript, the most open-source programming language [8]. With Python's rapidly growing usage and ease of access to open source packages via PyPI, it provided itself as an easy example to measure the impact of Open Source Software as a whole.

III. Related Work

Open source software development is closely related to collaborative production in academic research [6]. The National Science Foundation (NSF) recently announced their usage of $21 million to fund open source development through a new program: Pathways to Enable Open-Source Ecosystems (PEOSE). Existing NSF-funded research projects already result in publicly accessible, modifiable, and distributable open source software, data platforms, and even open hardware that catalyzes further innovation. Open source savings for scientists are large. The return on investment (ROI) for funders of open source development (100%-1,000% after only a few months) is frankly too high to ignore. The NSF wants to follow the best examples of open source development where the product is widely adopted and forms the foundation of a self-sustaining open source ecosystem (OSE). A distributed community of developers and a broad base of users across academia, industry, and government make up these OSEs [2]. The analysis in this paper is focused on the impact of open source software measured through a package dependency network. This analysis is very similar to measuring the impact of citations. In academia, h-index is a metric which is used very often to measure the scientific research impact [2]. It measures productivity and citation impact of the publications [9].

Empirical studies show that a network analysis approach can be applied to open source software networks to measure influence of a package similar to a node in a network using centrality measures [10]. The dependency network is represented by a directed acyclic graph and the graph centrality measures that we used to measure influence of packages are degree, in-degree, out-degree and eigenvector centrality.

IV. Data

PyPI package data was gathered through Google Cloud BigQuery (GCB) API [11]. PyPI package metadata table (distribution_metadata) and PyPI downloads (file_downloads) table were downloaded using this API. Package download data for this project was limited to downloads occurring the time period 01/01/2020 to 01/01/2021.

The distribution metadata table (referred to as distribution_metadata on GCB), contains information about each package including package name, version, author, maintainer, dependencies. Only packages which had an open source license and a repository on GitHub were used for this analysis. All versions of a package were treated as one package. The home_page field was processed to remove http, https, slashes, colons to keep only the username and repository name. Only the username and repository name signifies a GitHub slug [12]. For example, a package called "vcert", with the home page "https://github.com/venafi/vcert-python" is processed to "venafi/vcert-python". These slugs helped

to identify commits, contributors, countries, and emails for conducting impact analysis.

The file downloads table (referred to as file_downloads on GCB), contains information about each package download for the subject time period. The data of interest was the number of downloads completed during the subject time period for each package, not information on each individual download. We aggregated the data downloaded from the file_downloads table by package name, version, and download country code to produce a download count.

We used publicly available data on development activity of individual GitHub hosted projects and their contributors [13]. This data was collected and aggregated by previous work conducted by Korkmaz et al. [14]. To begin, the researchers used the Ruby Gem Licensee to classify the LICENSE file in each repository. This enabled them to select only packages with OSS approved licenses. Then the GHOST.jl package was used to collect and track GitHub user attributes and scrape all the commit history. This data was filtered to exclude any forked, mirrored, or archived repositories spanning GitHub's creation in 2008 through the end of 2019. Finally, the data was de-duplicated and filtered to exclude known bot accounts and commits merged from previous repositories. The produced commits activity data set totaled 3,260,612 distinct contributors and 7,628,101 distinct repositories. We used this data to calculate cost using the commits information on a package. A detailed explanation of this calculations is given in the methods section.

V. Methods

In this section, we describe the networks we created using the data gathered in the previous section. Our goal was to create these networks in order to measure the impact of packages using network-based (e.g., centrality) and OSS-based measures (e.g., number of downloads of packages). Additionally, we created a framework to measure the impact (cost) of OSS as intangible capital. We leveraged the COCOMO II model [1] to measure the cost of development as a resource cost of each package to determine which package had the highest development cost.

The initial number of rows of data we started with was 554,604. This included 75,000 unique packages. There were multiple rows for a package based on its dependencies. Packages without dependencies were removed. The data was then combined to obtain the unique packages to account for issues like multiple versions of packages. These packages were merged with commits data available for each package. The commits data was used to calculate development costs for each package and their dependencies to measure the impact. The results shown are only for the final merged data set, where the number of nodes in the network totaled 68,992.

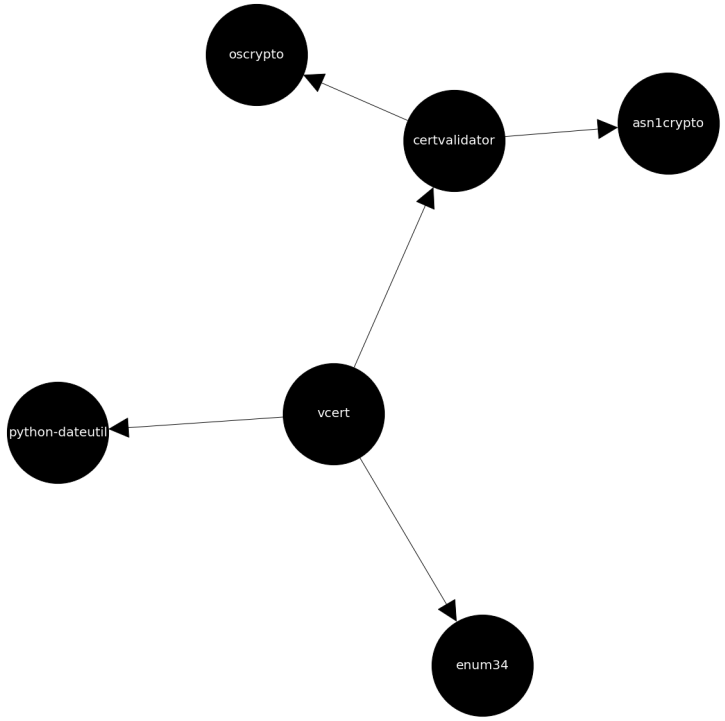
We were interested in analysis of centrality measures and the generated cost statistics to evaluate whether the most costly packages were the most influential, or if there was a correlation between these two statistics.

A. Network

The dependency network was created from a edge list with two columns: package and dependency name. The edge list contained 75,000 unique packages and their corresponding dependencies with a total row count of 409,385.

The edge list was used to create a directed acyclic graph [15] with packages represented by the nodes and edges representing dependency relationships. The number of nodes in the network including packages and their dependencies totaled 89,127. An arrow represents that a node was dependent on the node it was pointing to, as shown in Figure 1.

Fig. 1. Sample Dependency network



B. Centrality

A dependency network was created using the NetworkX package [16]. Only packages which had dependencies were included in this analysis. The following centrality measures were calculated for the dependency network: degree centrality, in-degree centrality, out-degree centrality, and eigenvector centrality. These measures are defined in Table I.

Degree centrality identified packages that exhibit a greater impact by indicating which packages were the most utilized and depended upon. Nodes with high in-degree centrality were the most influential in the application of Open Source Software development as their creation enabled the creation

of additional packages that leveraged the code previously created. Depending on what the in-degree centrality was of high out-degree centrality packages, one can describe whether the package was both leveraging and contributing to OSS (high in and out degree) or is leveraging OSS but has not significantly contributed to OSS (low in degree and high out degree). Meaning the work they have contributed to OSS was not being used for further development of new packages. This, however, does not exclude them from future influence as OSS continues to grow.

The eigenvector centrality value indicates the measure of influence of a package has on the network. This metric is based on the centrality score of a package's neighbors. In the context of our project, a package with a high eigenvector centrality is connected to other packages that are highly connected as well. The more highly connected a package is in the network the greater influence it has over the network. In other words, nodes with a high eigenvector centrality should have many neighbours that are also highly connected to other nodes [17].

TABLE I
CENTRALITY MEASURE DEFINITIONS

Centrality Measure	Definition
Degree Centrality	The fraction of nodes in the network a package is connected to. The degree centrality values are normalized by dividing by the maximum possible degree in a simple graph n-1 where n is the number of nodes in G.
In-degree Centrality	The fraction of nodes a package's incoming edges are connected to.
Out-degree Centrality	The fraction of nodes a package's outgoing edges are connected to.
Eigenvector Centrality	The measure of a node's influence in a network. Relative scores are assigned to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. A high eigenvector score means that a node is connected to many nodes who themselves have high scores.

C. Package Cost

For our analysis, we quantified economic impact by package development cost considered as intangible capital. Intangible capital is a non-physical factor expected to generate future benefits to the entities that control their use [18]. We used the COCOMO II model [1] to calculate the costs. The formula used is given below. The cost of a package is determined based on the kilo lines of code (KLOC) added to a repository. The effort multipliers from COCOMO II are parameters that we 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.

$$\text{Effort} = 2.4(\text{KLOC})^{1.05}$$
$$\text{Nominal development time} = 2.5(\text{Effort})^{0.38}$$
$$\text{Development cost} =$$
$$\text{(Monthly Resource Cost) (Nominal development time)}$$

The resulting nominal development time in person months is then multiplied by estimated monthly resource cost based on the wages of computer software developers. The methodology is consistent with measurement of software in the National Accounts.

D. Dependency Cost

In order to compute various statistics about a package, we utilized the NetworkX Package breadth first search function. This function, when given a package in the network, returned all of the dependencies for that package, as well as all of those packages' dependencies, branching until it collected all of the dependencies that the package either directly or indirectly leverages.

These statistics and metrics quantified how a package leveraged other packages across the network. One such statistic was the cost of all dependencies a package uses. We once again leveraged the COCOMO II model to calculate the cost of each dependency, summing them for each package to come to our final dependency cost statistic.

E. Downloads

As described in the data section the downloads are calculated for the year 2020-2021 by package, version and country. However, for this analysis the downloads were aggregated at a package level.

VI. RESULTS

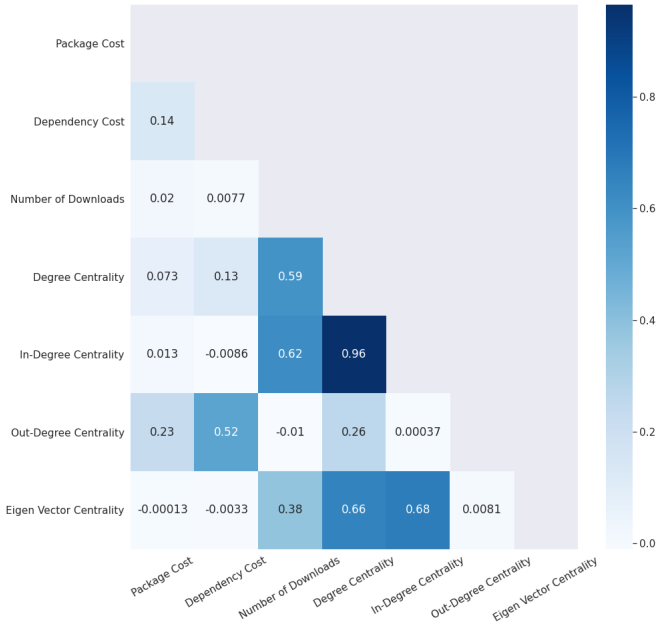
1) Correlation: Empirical studies have found correlation between the centrality measures in co-authorship networks and research performance of authors and institutions [19]. [19] used g-index, an extension of widely used h-index [9], as a performance measure. The studies found that only normalized degree centrality, efficiency, and average ties strength had significant influence on impact. Our analysis was consistent with these studies as we found that degree and eigenvector centrality measures had significant correlation with the number of downloads. For our analysis, we considered Karl Pearson Coefficient [20] as the correlation coefficient. A correlation score of 0.2 and above was considered a significant correlation given the power law distribution of our network.

- *Degree and in-degree centrality:* In a directed graph, in-degree centrality represents packages that are dependent

on another package. In our dependency network, we found that there were a higher proportion of packages that had significantly high in-degree centrality compared to out-degree centrality. Based on the way degree centrality was calculated, packages typically had a higher in-degree than out-degree and therefore the in-degree makes up a larger portion of the degree centrality. Therefore degree and in-degree centrality were highly correlated in our network. We picked only degree centrality to compare with other correlated metrics instead of in-degree as they were very similar in this case.

- *Degree and eigenvector centrality:* Degree centrality captures the number of connections in general and eigenvector centrality captured the number of connections and the weight based on the number of neighbor's connections. They were highly correlated because some of the packages which had high degree centrality may also be well-connected with neighbors which are highly connected, such as package 'six' and 'pytest-cov', but not perfectly correlated. This is because, though some have high degree centrality, their neighbors may not be well connected to other packages such as package 'flask'.
- *Number of downloads and eigenvector centrality:* When a package is downloaded from PyPI, it could be a core component of a suite of a software package that is being installed. This demonstrates that the package may not have high dependencies but is influential based on neighbor's connections, such as with the package 'typing-extensions'. Therefore, we are able to see high correlation between number of downloads and eigenvector centrality measures.
- *Number of downloads and in-degree centrality:* We observed high correlation between downloads and in-degree centrality. When a package is downloaded, it is forced to download the dependencies. Therefore when a package has high in-degree centrality it might also be downloaded the most, as many packages depend on it, such as 'pytest-cov'. However, if the dependencies sometimes already exists on the environment, it may not be downloaded.
- *Out-degree centrality and dependency cost:* Out-degree centrality is high when a package development is dependent on many other existing packages. Therefore the cost calculated on dependencies will be high when a package has high out-degree centrality. It might not be a perfect correlation since the cost of all dependencies may not be available.
- *Centrality measures and cost:* We observed that there is no correlation between package/ dependency cost with any of the centrality measures. This implies that a package with higher value is necessarily not an influential package.

Fig. 2. Correlation Matrix of Network Impact Measures



2) Influential packages ranked by centrality: Centrality measures and number of downloads were compared for the top ranked packages. Multiple packages ranked highly across metrics indicating that the packages were influential in multiple ways. Specifically, the top seven packages based on degree centrality were compared. The results are shown in Table II. Our results indicate the package "six" is the most highly ranked across all centrality measures and number of downloads. Therefore, package "six" is the most highly-connected node in the graph (degree centrality), most depended upon (in-degree centrality), is highly-connected with highly-connected neighbors (eigenvector centrality), and most downloaded package for the time period of study.

TABLE II
MOST INFLUENTIAL PACKAGES RANKED BY CENTRALITY AND DOWNLOADS

Package	Degree Centrality	In-Degree Centrality	Eigenvector Centrality	Downloads Ranked
six	1	1	1	1
pyyaml	2	2	33	5
pytest-cov	3	3	3	9,810
coverage	4	4	8	5,678
flask	5	5	154	8,477
twine	6	6	2	9,757
toml	21	20	4	28,586
releasecmd	936	273	5	46,570

The package "six" provides a Python 2 and 3 compatibility library, intended to smooth over differences between the Python versions. It has the goal of writing Python code that is compatible with both Python versions.

The package "PyYAML" is a YAML parser for Python. YAML is a data serialization format designed for human readability and interaction with scripting languages.

Pytest-cov is used for software testing and provides coverage reports, subprocess and Xdist support.

The package "coverage" is a tool for measuring code coverage of Python programs. It monitors a Python program and provides which parts of the code have been executed and identifies code that could have been executed but was not.

The package "flask" provides a web framework facilitates easy development of web applications.

Twine is a utility for publishing Python packages on PyPI and allows for both source code and binary package distribution.

The package "toml" is used for processing TOML (Tom's Obvious Minimal Language).

Releasecmd is a subcommand for setuptools. It creates a git tag and push and enables upload of packages to PyPI.

VII. Future Work

The creation and use of OSS highlight an aspect of technology diffusion and flow that is not captured in science and technology (S&T) indicators. We want to extend this work to measure the impact of diffusion of OSS innovation. One approach is to quantify OSS diffusion between countries. We are gathering the country information of the package's top contributor and assigning that country to a package. Attributing packages to the countries that developed them could provide the means to show centrality measures between countries. This could additionally provide information about which countries contribute the most to the open source software world, or which countries leverage the most packages from other contributors. This analysis could uncover potential groupings or clusters of countries that have greater collaboration amongst themselves than others. These diffusion measures can be applied to organizations, sectors, and other open source languages such as R.

VIII. Conclusion

Our analysis can be scaled to measure impact of any open source ecosystem or other aspects of open science, such as shared data available in public access repositories. The costs we have calculated for the dependency tree show that the influence and impact of a package and can be used to model the diffusion of innovation between countries. This diffusion occurs as developing packages leverage packages created by other countries to develop their own packages. The diffusion of innovation in the organizations where the packages were developed is leveraged by other organizations for development of other packages. Our exploratory research will provide benefit to many actors in the research ecosystem, including researchers, policy-makers, and funders. In the future, industry will likely begin to build impact measurements into their software platforms. For example, GitHub is making similar styles of analysis easier by providing a dependency graph for the packages being hosted there [21].

IX. Acknowledgments

This material is based on work supported by the National Science Foundation (Award Number: 2135757). We extend our sincere gratitude to Dr. Gizem Korkmaz and Dr. Brandon Kramer for their consistent support and insights. We thank the University of Virginia School of Data Science for facilitating the success of this project and Dr. Abbas Kazemipour for his invaluable guidance and for being our mentor. We would also like to thank Neil Kattampallil for his support with database access and queries.

References

[1] Software cost estimation with Cocomo II. Prentice Hall, 2009.
[2] K. Fogel, Producing open source software: how to run a successful free software project, 1st ed. O'Reilly, 2005.
[3] C. A. Robbins, G. Korkmaz, J. B. S. Calderón, D. Chen, C. Kelling, S. Shipp, and S. Keller, "Open source software as intangible capital: measuring the cost and impact of free digital tools," in Paper from 6th IMF Statistical Forum on Measuring Economic Welfare in the Digital Age: What and How, 2018, pp. 19–20.
[4] DevSkiller - Powerful tool to test developers skills, Sep 2020.
[5] G. V. Rossum, "The history of python: A brief timeline of python," The History of Python, Jan 2009.
[6] G. Korkmaz, C. Kelling, C. Robbins, and S. Keller, "Modeling the impact of python and r packages using dependency and contributor networks," Social Network Analysis and Mining, vol. 10, no. 1, pp. 1–12, 2020.
[7] "Tiobe index for march 2022."
[8] A. D. Rayome, "The 10 programming languages developers use most in open source projects," TechRepublic, Oct 2018.
[9] Wikipedia, Feb 2022, page Version ID: 1069803333.
[10] M. Joblin, S. Apel, and W. Mauerer, "Evolutionary trends of developer coordination: A network approach," Empirical Softw. Engg., vol. 22, no. 4, p. 2050–2094, aug 2017.
[11] "Google big query," Google Cloud Blog.
[12] "Slug," Semrush Blog.
[13] B. Kramer, "Network analysis of open-source software python packages on github," 2022, type: dataset.
[14] B. Kramer, G. Korkmaz, B. Calderon, and C. Robbins, "International collaboration in open source software: A longitudinal network analysis of github."
[15] S. Andale, "Acyclic graph & directed acyclic graph: Definition, examples," Statistics How To, Apr 2016.
[16] "Networkx documentation."
[17] K. Kubara, "Feature extraction for graphs," Medium, Sep 2020.
[18] "OECD and Eurostat", "Oslo Manual 2018", "2018".
[19] A. Abbasi, J. Altmann, and L. Hossain, "Identifying the effects of co-authorship networks on the performance of scholars: A correlation and regression analysis of performance measures and social network analysis measures," Journal of Informetrics, vol. 5, no. 4, p. 594–607, Oct 2011.
[20] Statistics How To.
[21] GitHub Docs.