A Methodology for Analyzing Uptake of Software Technologies Among Developers

***Abstract*—Motivation: The question of what combination of attributes drive the adoption of a particular software technology is critical to developers because it determines what technologies receive wide support, and which technologies may be abandoned, thus rendering developers' investments worthless. Aim and Context: Using discrete choice models, we investigate software technology adoption among developers and provide insights to help them to both improve visibility among alternative technologies as well as to insure that their own product becomes widely adopted. Approach: We leverage social contagion theory to operationalize various choice models. These choice models use developer actions, i.e., the technologies they actually choose, to estimate implicit preferences among these technologies. We employ a large collection of open source version control repositories (containing over 4 billion unique versions) to construct a software dependency chain for R language source code files and investigate the combination of attributes that drive adoption among competing data frame (a core concept for a data science language) implementations in the R language: `tidy` and `data.table`. To characterize the technology choices, we quantify key characteristics of the choice (time until response to a raised issue, overall deployments, number of open defects, knowledge base) and of the decision maker (performance needs, scale, and their social network). Results: We find that quick response to raised issues, larger overall deployments, and fewer open defects lead to higher adoption. Decision makers tend to adopt technology that is closer to them in the dependency network and author collaborations network while meeting their performance needs. Future work: We hope that our methodology encompassing social contagion, elucidation of key measures from large collections of version control data, and choice models provides a general path toward increasing visibility, driving better decisions, and producing more sustainable and widely adopted software.**

***Index Terms*—choice models, social contagion, technology adoption, library migration, software supply chain**

I. Introduction

Open source has revolutionized software development by creating and enabling both a culture and practice of reuse, where developers can leverage a massive number of software languages, frameworks, libraries, and tools (we refer to these as software technologies) to implement their ideas. Public repositories of software technologies allow developers to focus on their innovation [1]–[4], potentially reducing lead times and effort. This approach, however, is not absent of risks. For example, if a particular technology chosen by a developer is later supplanted by another, incompatible technology, the support for the supplanted technology is likely to diminish. Reductions in support for the supplanted technology result in increased effort on the part of the developer to either provide fixes upstream or to create workarounds in their software. Furthermore, the value of the developer's creation to new downstream projects may be diminished in favor of the now more popular alternative technology. As a consequence, both the importance of a developer's product and their reputation would suffer. To remediate these two risks, developers must understand the implications and utility of their software products among downstream adopters and consumers of the technology in relation to alternative, competing technologies. It is natural, therefore, to adopt the position that open source software development should be investigated from a supply chain perspective, which also pertains to distributed decision and supply networks among different stakeholders. We refer to the collection of developers and groups (software projects) producing new versions of the source code as a Software Supply Chain (SSC) [5], [6]. The upstream and downstream links from project to project are represented by the source code dependencies, sharing of the source code, and by the contributions via patches, issues, and exchange of information. While the product adoption in supply chains has been well studied [7]–[10], little is known or understood about how developers choose what components to use in their own software projects.

As a complex dynamical system, every player in the open source ecosystem may have their specific set of preferences and by their actions, can affect the ultimate outcome of wide (or narrow) adoption and/or abandonment of formerly popular technologies. These decisions are not only based on technical merit but fashion plays a role as well. Furthermore, these SSC networks may severely limit developer choices at the particular point in time when they need to make decisions on which components or technologies to use. Hence, in contrast to common conventions, we should not simply model the preferences of individual developers but must also take into account the complexity of the supply networks. We want to address this major gap in knowledge empirically by using a very large data source comprising version control data of millions of software projects. Our methodology involves using this data to construct software supply chain networks, identifying software technology choices, theorizing about factors that characterize the developer and the technologies they chose, and finally fitting and interpreting the choice models for specific technology choices and, thus, characterizing the implicit developers' utility functions they use to make their decision.

Despite the practical and theoretical importance of the question how developers make technology choices, the literature does not offer any theoretical guidance in this process. We, therefore, leverage social contagion theory, which has been effective in clarifying key aspects of organizational adoption

of technology [11], [12]. This theoretical lens can bring theoretical and practical insights for specific technology choices by providing interpretations to manage the risks and benefits associated with each choice. We focus on two questions in particular, **RQ1:** What characteristics of a developer affect their choice between major technology alternatives? Furthermore, **RQ2:** what properties of the technology matter in the developers' choice between alternative technologies?

To answer RQ1 and RQ2, we need to collect data on the actual choices made by developers. To operationalize key theory-based measures, we need an approximation of all public software projects that may choose the technology under study.

To exemplify the proposed methodology, we investigate rapidly growing data-science software ecosystem centered around the `R` language. One of the key technology choices in this area are the data structures used to store data (in the data-science sense). R has two major competing technologies implemented in packages `data.table` and `tidy`. (a more detailed introduction of these two packages is given in Sec. IV)

Our research provides several theoretical and practical innovations. From the theoretical standpoint, the novelty of our contribution first lies in introducing social contagion theory that provides first-principles based methods to construct hypotheses and to determine measures that should affect technology adoption. The second novelty is the context in which we investigate technology choices, i.e., a complete SSC [13], [14], not restricted to a set of projects or ecosystems. Third, we use choice models to understand how macro trends at the scale of the entire SSC emerge from actual decisions the individual developers make to select a specific software technology. More specifically, as a result of contextualizing social contagion theory through SSCs, our approach provides novel measures, such as proximity in a dependency network and authorship network, questions and answers with high quality in Q&A, performance needs, and total deployments, that strongly affect the spread of technology and that were not used in prior work on library migration.

From the practical standpoint, our contribution consists of proposing a method to explain and predict the spread of technologies, to suggest which technologies are more likely to spread in the future, and suggest steps that developers could take to make the technologies they produce more popular. Developers can, therefore, reduce risks by choosing technology that is likely to be widely adopted. The supporters of open source software could use such information to focus on and properly allocate limited resources on projects that either need help or are likely to become a popular infrastructure. In essence, our approach unveils previously unknown critical aspects of technology spread and, through that, makes developers, organizations, and communities more effective.

In Sec. II we introduce the diffusion of innovation, social contagion, and the application of choice models. In Sec. III, we describe the dataset and how we operationalize software supply chain. Choice model theory and our candidate technology are introduced in Sec. III-E and Sec. III-G respectively. In Sec. IV, operationalization of attributes of choice model is illustrated. Sec. V describes and interprets the result of applying the choice model. Related work is discussed in Sec. VII and major limitations are considered in Sec. VI. We summarize our conclusions and contribution in Sec. VIII.

II. Conceptual background

We draw on methodologies from a diverse set disciplines. The phenomena we are investigating is often called adoption [15] or diffusion of innovation [16]. Both theoretical approaches model how products or ideas become popular or get abandoned. We would like to fit such models and, in order to do so, find relevant set of predictors that have theoretical justification. Fichman [17] considered how internal factors such as resources and organization predict innovations in commercial enterprises, and DiMaggio [18] included the factor of environment as well. The adopters of the technology may influence non-adopters over time. Angst *et al.* [11], use the concept of social contagion [19], which consists of observation, information transmission, and learning to study spread of electronic health records. These concepts are familiar to any open source developer. More specifically, in addition to purely social contagion, we also have technical dependencies that act as strong constraints on developer actions. The signaling theory applied for social coding platforms [20], [21] provides some specific guidance as to what may motivate developers to chose one project over another. Many of the actions developers take on GitHub are focused on building or maintaining their reputation, hence they pay a particular attention to measures such as activity, numbers of participants, or "stars"[1].

The basic premise of social contagion theory is that developers may observe the actions and decisions of others, communicate them, and learn to emulate them over time. This premise implies that groups and individuals who are in social and spatial proximity to prior adopters are more susceptible to the influence of prior adopters of technology. This susceptibility (synonymous with potency or infectiousness of influence) is likely to result in an increased likelihood to adopt the same technology [11]. These precursors of spread, if measured and calibrated with the actual level of technology spread, would provide the relative importance of each factor in driving the adoption and provide badly needed understanding to help developers choose technologies wisely and provide hints on how to make their own technology more widely adopted. Fortunately, the mathematical adoption models have been developed and refined over time. These so called choice models [22] are used to describe the behavior of a decision maker given a well defined set of choice alternatives. Choice models have been used successfully in the fields of marketing [23]–[26] and economics [27]–[29] to understand how consumers make choices. Adapting and applying choice models to technology adoption, we focus on a developer, or more precisely, a software project as a decision maker. The actual decision is the first among the alternative technologies that a

[1] placing a star on a GitHub repository allows a developer to keep track of projects they find interesting and to discover similar projects in their news feed.

project selects. Choice models have two types of predictors: properties of the choice (i.e., the technology) and properties of a decision maker (i.e., the project or individual developer).

With the social contagion theory and choice models, we set out to address RQ1 and RQ2 by empirically characterizing the spread of software technology through analysis of a very large collection (VLC) of version control data introduced in [30] and curated by OSCAR project [2]. We refer to it as OSCAR-VLC. According to the curators, OSCAR-VLC approximates the entirety of public version control and includes major forges such as GitHub, BitBucket, GitLab, Bioconductor, SourceForge, now defunct Googlecode, and many others and currently contains over 46M projects. OSCAR-VLC production involves discovering [31] and cloning the projects, extracting Git objects from each repository, and then storing these objects in a scalable key-value database.

OSCAR-VLC is used to construct the SSC [32], [33] by determining dependencies among software projects and developers, then by characterizing these projects according to their technical characteristics and supply chains. The social contagion and signaling theories allow us to select meaningful measures for the decision makers and for their choices. (We present our measures in Sec.IV-A)

III. Constructing software supply chains

Source code changes made in software projects are recorded in a VCS (version control system) used by the software project. Many of the projects are using git as their version control system, sometimes with historic data imported from SVN or other VCS used in the past. Code changes are typically organized into commits that make changes to one or more source code files. Internally, the Git database has three primary types of objects: commits, trees, and blobs [34]. Each object is represented by its sha1 value that can be used to find its content. The content of a blob object is the content of a specific version of a file. The content of a tree object is, essentially, a folder in a file system represented by the list of sha1s for the blobs and the trees (subfolders) contained in it. A commit contains the sha1 for the corresponding tree, a list of parent commit sha1s, an author string, a committer string, a commit timestamp, and the commit message. Fig. 1 illustrates relationships among objects described above.

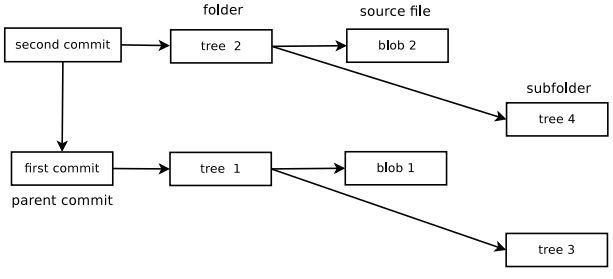

Fig. 1. Git objects graph

[2]bitbucket.org/swsc/overview

We utilize all Git objects (1.1 billion commits and 4 billion of blobs and trees) from OSCAR-VLC to construct the relevant supply chain, social, and adoption measures. For our analysis we create mappings among these objects and their attributes, e.g., filename to associated blobs.

A. Software supply chain

In traditional supply chains [35]–[39], the networks include material, financial and information relationships. Similar concepts can be operationalized in the software domain, with developers or projects representing the nodes and information transfer or static dependencies among projects representing links. Based on the characteristics of software domain, especially the open source community, and the ability to measure various attributes relevant to technology adoption, we consider two different types of network relationships: dependency networks, and authorship networks.

B. Measuring the dependency network

While many types of static dependencies exist, here we focus on explicit specification of the dependency in the source code. For example, 'import' statements in Java or Python, 'use' statements in Perl, 'include' statements in C, or, as is the case for our study, 'library' statements for the R language.

We analyze the entire set of 4 billion blobs in the collection using following steps:

1) Use file to commit map to obtain a list of commits (and files) for all R language files by looking for the filename extension '.[rR]$'
2) Use filename to blob map to obtain the content for all versions of the R-language files obtained in Step 1
3) Analyze the resulting set of blobs to find a statement indicating an install or a use of a package:
   - `install\.packages\(.*"PACKAGE".*\)`
   - `library\(.*[\"']*?PACKAGE[\"']*?.*\)`
   - `require\(.*[\"']*?PACKAGE[\"']*?.*\)`
4) Use blob to commit map to obtain all commits that produced these blobs and then use the commit to determine the date that the blob was created
5) Use commit to project map to gather all projects that installed the relevant set of packages

These steps are illustrated in a flowchart in Fig. 2. In Fig. 2, the rectangular boxes represent inputs and outputs, and ovals represent maps or dictionaries we utilized in this study. f2b stands for filename-To-blob map, b2cnt stands for blob-To-content map, b2cmt stands for blob-To-commit map, and cmt2prj for commit-To-project map. The number on the left side represents the unique number of corresponding objects.

A similar approach can be applied to other languages with suitable modification in the dependency extraction procedures, since different package managers or different languages might require alternative approaches to identify dependencies or the instances of use.

In addition to dependencies, we also need to obtain measures that describe various aspects of social relationships among developers because the theories of adoption, such

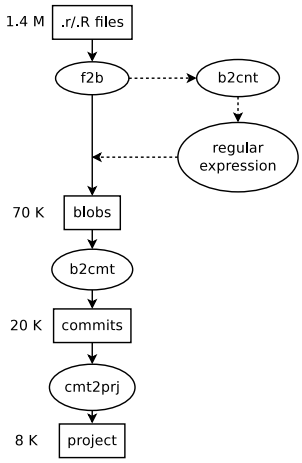

Fig. 2. Project discovery

as social contagion theory we employ, need measures of information flows among individuals as an important factor driving the rate of adoption.

C. Measuring the authorship network

The authorship network can be viewed as the process of developers working with other developers either by implicitly learning skills from other's contribution (source code) or by explicitly communicating through emails or discussion platforms. Here we focus on the former mode of communication since the bulk of direct communication may be private. We consider two types of links among developers. A weak link exists between a pair of developers if they commit in at least one project that is common between them and a strong link exists if they change at least one file in common.

D. StackExchange

StackExchange is a popular question answer website related to programming. When people search for information there they may notice answers that suggest the use of either `tidy` or `data.table` (discussion about choosing these packages is in Sec. III-G) and, consequently, might be inclined to incorporate one of these packages into their own code. The latest (2017-12-08) StackExchange data dump including 57GB of posts was imported into MongoDB, out of which 6k questions (excluding answers) were found to be related to either `data.table` or `tidy` by searching for these two terms in the title or the content of the post. We operationalize two measures: one counts the total number of posts while another measure counts only questions that have a score above 20 to gauge the amount of high-quality content that is likely to be referred to from search engines.

E. The Choice Model

The choice set (set of alternatives) needs to exhibit three characteristics to be able to fit a discrete choice model. First, the alternatives need to be *mutually exclusive* from the perspective of decision maker, i.e., choosing one alternative means not choosing any other alternative. Second, the choice set must be *exhaustive* meaning all alternatives need to be included. Third, the number of alternatives must be finite. The first two conditions can be easily met in our case: Our choice set consists of two packages - `data.table` and `tidy`; Decision makers are restricted into the group of projects in our collection where either of those two packages is installed. To ensure the choices are mutually exclusive we model the choice of the first technology selected.

In general, discrete choice models are derived from random utility models (RUMs) [28] under the assumption of utility-maximizing behavior by the decision maker. The derivation of RUMs can be summarized as follows. A decision maker n faces a choice among J alternatives. The decision maker would gain a certain amount of utility or profit from choosing each alternative. The utility that the decision maker n obtains from choosing alternative j is labeled $U_{nj}, \quad j = 1, ..., J$. This utility is known to decision maker but not to researchers. Decision maker n choosing alternative i is denoted as $D_n = i$. The assumption is that decision maker chooses the alternative with greatest utility, i.e.,

$$D_n = i \quad \Longleftrightarrow U_{ni} > U_{nj} \quad \forall j \neq i \tag{1}$$

Now let's consider the researcher. The researcher does not observe the utility of the decision maker. The researcher observes some attributes of decision maker, labeled s_n, and some attributes of alternatives faced by decision maker, labeled $x_{nj}, \forall j$. These observed attributes can be aggregated into a function to estimate utility of the decision maker. The function is denoted $V_{nj} = V(x_{nj}, s_n) \forall j$ and is named *representative utility*. Since not all the attributes can be observed by researchers, $V_{nj} \neq U_{nj}$. Utility can be decomposed into $U_{nj} = V_{nj} + \varepsilon_{nj}$, where ε_{nj} represents influence of unobserved factors on decision maker's utility. So the probability of decision maker n to choose alternative i is:

$$\begin{aligned} P_{ni} &= Prob(U_{ni} > U_{nj} \quad \forall i \neq j) \\ &= Prob(V_{ni} + \varepsilon_{ni} > V_{nj} + \varepsilon_{nj} \quad \forall i \neq j) \\ &= Prob(\varepsilon_{nj} - \varepsilon_{ni} < V_{ni} - V_{nj} \quad \forall i \neq j) \\ &= \int_{\varepsilon} I(\varepsilon_{nj} - \varepsilon_{ni} < V_{ni} - V_{nj} \quad \forall j \neq i) f(\varepsilon_n) d\varepsilon_n \end{aligned} \tag{2}$$

In equation (2), $f(\varepsilon_n)$ is the joint density of the random vector ${\varepsilon_n}' = \langle \varepsilon_{n1}, ..., \varepsilon_{nJ} \rangle$, i.e., the density of the unobserved portion of utility. $I(\cdot)$ equals 1 when the expression in parentheses is true and 0 otherwise.

Different discrete choice models are obtained from different specifications of density $f(\varepsilon_n)$. In particular, the mixed logit model allows the unobserved factor to follow any distribution [27], [28]. In our case we do not know what the unobserved factors could be nor what distributions they follow. In this paper we, therefore, applied the mixed logit model to study developers' choice over analogous R packages (`data.table` v.s. `tidy`).

F. Issues

It's reasonable to believe that the number of issues and how an issue is solved during the development of a software package may affect a developer's choice. We, therefore, collect

the issues reported during the development of `data.table` and `tidy` packages. Since both are hosted on GitHub, we use GitHub API to scrape every issue[3] reported for both packages. We collected 2.6k issues for the `data.table` package and 1.6k issues for the `tidy` package.

G. Selecting candidates for study of adoption

We chose software technologies from the data science ecosystem of projects using the R language because several of the co-authors are knowledgeable and have decades of development experience in R, and we do not need to seek external experts to provide interpretations of the findings. As with most language-based ecosystems, the core language provides only basic functionality with most of the external packages being maintained in CRAN and Bioconductor distributions. Each package can be thought as presenting a technology choice. Since the technologies of storing and managing data are crucial in data science, we selected two widely used such technologies: `data.table` and `tidy`.

The extraction of the supply chain data for these two packages started from 1.4M R files in the entire collection, with 70K blobs that contained the statement of installing either package. Fewer than 20K commits produced these blobs and were done in 24K (including forks) projects which installed either package.

We further refined the list of projects because a large fraction involved forks of other projects. One of the most typical ways to make contribution to the development of a project on GitHub is by creating a fork for a project, making changes to this clone and then sending a pull request to original project. As a result, a popular project may have hundreds of forked projects that share a large portion of source code and commit history. To a certain extent, these forks are not equivalent to the original projects and we want them to be removed from consideration. In order to detect and delete these forks, we classify projects based on common commits, i.e., a pair of projects stay in one cluster if they have at least one commit in common. After applying such classification method on 24K projects, we obtain 8K clusters. Each cluster represents a single observation in our study and, the date the first blob containing the use of technology was created is used as the date that technology was adopted for this cluster.

IV. Case study

Apart from the `dataframe` package that is a part of core R language, `data.table` and `tidy*` are two other most popular packages for data manipulation. More specifically, `tidy*` represents a list of packages that share an underlying design philosophy, grammar, and data structures that are built for data science in R. Hadley Wickham, the Chief Scientist at RStudio and the main developer of `tidy*`, developed a family of packages called `tidyverse` to facilitate[] the usage of `tidy*` packages by assembling them into one meta package. We extract a set of packages from `tidy*` that share similar functionalities with `data.table` and refer to all of them as `tidy` package in our paper. This includes `tidyr`, `tibble` and `readr` packages.

`data.table` was written by Matt Dowle in 2008 and is known for its speed and the ability to handle large data sets. It's an extension of base R's data.frame with syntax and feature enhancements for ease of use, convenience and programming speed. It's built to be a comprehensive, efficient, self-contained package, to be fast in data manipulation, and it has a very succinct DSL (domain-specific language). Conversely, `tidy` focuses on the beauty of function composition and data layer abstraction which enable users to pull data from different databases using the same syntax.

A. Operationalizing Attributes for Choice Models

In this section, we define and justify the variables that quantify the key attributes pertaining to the set of software choices available to developers, as well as the key attributes characterizing the developers making the choices.

We propose 11 variables to measure various aspects that may have influenced a developer's choice. These variables are listed in Table I.

a) Cmts & Aths: the number of commits and authors represent the size of a project which may affect the choice of package. Larger projects, for example, may prefer less controversial/more conservative choices. This is a quality of the choice, so it would most closely fit under the "infectiousness" category in social contagion theory. We chose not to use lines of code (LOC) as size measure since it has less stable distribution than the number of commits. Also, we wanted to reduce correlations among predictors and LOC was highly correlated with the number of commits.

b) CumNum: the overall number of deployments increase the chances that a developer would see an example of the usage of a package and may choose to use it in their code. This measure is most closely related to the "exposure" category, because it quantifies the chances that a developer may get in contact with the technology.

c) Unrslvd: a higher fraction of unresolved issues indicates that the package has a significant number of unresolved problems which may undermine people's confidence in it. This is a quality of the choice, so it would most closely fit under "infectiousness" category in the social contagion theory.

d) C: is used as a proxy of whether or not a project has a requirement for high performance. Typically, only computations that are too slow for the interpreted R language are implemented in C to reach higher performance. This is a quality of the decision maker that would most closely fit under the "infectiousness" category because it indicates the preference for more performance embodied by `data.table` choice.

e) StckExch: is a proxy for the popularity of each packages. It counts the number of high quality (score > 20) questions related to each choice. Developers often search for answers to issues they face and may stumble on one of these packages presented as a solution to a problem they are facing,

[3]A pull request is also treated as an issue in this paper. https://developer.github.com/v3/issues/

TABLE I
INDEPENDENT VARIABLES

Independent variables	Annotation	Category	Property type
CumNum	the total number of projects that deployed the package	exposure	choice related
RplGp	the time gap until the first reply to an issue	infectiousness	choice related
Unrslvd	the number of open issues over the number of all issues	infectiousness	choice related
StckExch	the number of questions with score above 20 related to either package	exposure	choice related
C	boolean, indicating whether a project contains C file	infectiousness	decision maker
Cmts	the number of commits	infectiousness	decision maker
Aths	the number of authors/developers	infectiousness	decision maker
Prx2TD	the proximity to tidy through dependency network	proximity	decision maker
Prx2DT	the proximity to data.table through dependency network	proximity	decision maker
AthPrx2TD	the proximity to tidy through authorship network	proximity	decision maker
AthPrx2DT	the proximity to data.table through authorship network	proximity	decision maker

TABLE II
NETWORK CHARACTERISTICS

Characteristics	data.table	tidy
No. of downstreams	813	2203
No. of downstreams layers	5	5
No. of packages in common	636	
overlap ratio	0.78	0.28

thus increasing chances that they may use that technology. In social contagion theory this would represent the category of "exposure". We avoid counting the total number of questions because most of the questions tend to be of low quality and the search engines tend to avoid including links to them, thus they do not increase "exposure."

f) Prx2DT/Prx2TD: can be explained from the perspective of software supply chain networks. Based on the characteristics of software domain, especially the open source software community, dependency networks can be viewed as technologies (library/package) spreading from upstream (original package) to downstream (packages where the original package was installed) and from downstream to its downstream.

We consider all downstream packages of `data.table` as in the `data.table` cluster and those of `tidy` as in the `tidy` cluster. The hypothesis is that if a project installed one package that is in the `data.table` cluster, then the project is more likely to install `data.table` than `tidy`. The rationale of such hypothesis is that if developers installed a package because of certain preferences for some of its functionalities or features which are inherited from its upstream package (`data.table`), or the way such package works which sometimes was influenced by or derived from its upstream package (`data.table`), it's more likely that these developers will choose the upstream package (`data.table`) over other alternatives whenever there is a need.

Based on the dependencies of R CRAN packages, the clusters of `data.table` and `tidy` can be easily constructed. More specifically, we used METCRAN[4] API and scraped meta data for more than 11K R CRAN packages for which dependency information is available. After that, we constructed these networks and show the basic information in Table II.

[4]https://www.r-pkg.org/about

Each downstream package in the `data.table/tidy` dependency networks needs to be weighted before calculating proximity to both `data.table` and `tidy` for each observation. We suggest that the algorithm used to determine the weights be based on several key principles:

- for each downstream package, only the relative weight to root package (`data.table/tidy`) matters
- for each downstream package, the sum of its weights to both root packages is a constant
- the closer to a root package, the higher the weight that a downstream package gets relative to that root package

We assume that each package has a weight of 1 in total. Let's denote the packages set in `data.table` downstream network as S_d, that in `tidy` as S_t, the weight of package a to `data.table` as W_{ad} and that to tidy as W_{at}, the depth of package a in the `data.table` network as D_{ad} and that in the `tidy` network as D_{at}, then based on principles mentioned above, the weights of package a are determined as follows:

- $W_{ad} = 1, W_{at} = 0$ if $a \in S_d$ & $a \notin S_t$
- $W_{ad} = 0, W_{at} = 1$ if $a \in S_t$ & $a \notin S_d$
- otherwise, $W_{ad} = D_{at}/(D_{ad} + D_{at})$, $W_{at} = D_{ad}/(D_{ad} + D_{at})$

The next step is to extract the list of packages installed in each observation/project, after which we can aggregate the weights of these packages to compute the proximity of each project.

As we have mentioned in Sec. III, various maps among Git objects have been created. By utilizing maps of project-To-commit, commit-To-blob, blob-To-content in sequence and selecting the install statements in blob content via regular expressions similar to ones mentioned in Sec. III-B, we get the list of packages installed in each project. From this set, we obtain projects that are either in `data.table` or in `tidy` clusters.

For a project p, denote the list of packages obtained in last step as L_p and denote a package in that list as a. Then the proximity of a project p to `data.table`, denoted as P_{pd}, and to `tidy` as P_{pt}, can be determined via:

$$\begin{cases} P_{pd} = \Sigma_a^{L_p} W_{ad} \\ P_{pt} = \Sigma_a^{L_p} W_{at} \end{cases} \tag{3}$$

g) AthPrx2DT/AthPrx2TD: can be explained from the perspective of social contagion. Social contagion refers to

the propensity for a certain behavior to be copied by others. Consider the fact that developers in GitHub are linked through common projects they are devoted to, where information and ideas are shared and transmitted from one to others, an underlying social network emerges. Organizational actions are deeply influenced by those of other referent entities within a given social system, according to DiMaggio [18]: non-adopters are influenced by adopters over time, and they influence the actions of other non-adopters after their own adoption [11] if thinking of our case as package adoption. In short, the adoption of `data.table/tidy` is a temporal process of social contagion.

We attempt to look for developers that are exposed to contagious packages — `data.table/tidy`. These developers include not only the authors of each package who are directly exposed inherently, but also developers who cooperate with directly-exposed authors in other projects. Directly-exposed authors, the authors of `data.table/tidy`, are obtained by applying project-To-author map to `data.table/tidy` package separately and indirectly-exposed authors are obtained by combining map author-To-project and map project-To-author serially and then applying it on each directly-exposed author.

We name authors exposed to `data.table` as `data.table` author cluster and those to `tidy` as `tidy` author cluster. Projects/observations may have authors who are in either of these two clusters and these authors may have an impact on choosing alternative technologies (`data.table` vs. `tidy`). In order to estimate the impact of every author in each cluster, we use:

- $W_{bd} = 1, W_{bt} = 0$ if $b \in C_d$ & $b \notin C_t$
- $W_{bd} = 0, W_{bt} = 1$ if $b \in C_t$ & $b \notin C_d$
- otherwise, $W_{bd} = D_{bt}/(D_{bd} + D_{bt})$, $W_{bt} = D_{bd}/(D_{bd} + D_{bt})$

where b represents an author in a project; C_d/C_t stands for author cluster of `data.table/tidy`; D_{bd}/D_{bt} refers to the distances from author b to `data.table/tidy`, i.e., author b's depths in author cluster of `data.table/tidy`, 1 for directly-exposed author and 2 for indirectly-exposed author; W_{bd}/W_{bt} is the proximity of author b to `data.table/tidy`, indicating author b's impact on choosing `data.table/tidy`. Note that these measures are similar to the ones used in calculating *Prx2DT/Prx2TD* and are based on similar principles.

After estimating each exposed author's influence, the overall exposed authors' influence in project p can be measured as follows:

$$\begin{cases} PA_{pd} = \dfrac{\Sigma_b^{A_p} W_{bd}}{N_p} \\ PA_{pt} = \dfrac{\Sigma_b^{A_p} W_{bt}}{N_p} \end{cases} \tag{4}$$

where A_p is the set of authors of project p who are in either of `data.table/tidy` author cluster; W_{bd}/W_{bt} is the proximity of author b to `data.table/tidy` calculated in previous step; N_p is the number of authors in project p; PA_{pd}/PA_{pt}, i.e., *PrxAth2DT/PrxAth2TD*, is the overall influence of exposed authors on a project p. Notice that *PrxAth2DT/PrxAth2TD* is calculated through aggregating the influence of each exposed author and being normalized over the total number of authors in that project. The rationale for normalization is that a project tends to have more exposed authors if it contains more authors, resulting in a higher value for *PrxAth2DT/PrxAth2TD*. By normalization we remove this bias induced by the difference in the number of authors for different projects.

h) RplGp: measures how fast the developers or maintainers of a package respond to a raised issue. The timeliness of this response reflects the efficiency of package maintenance and can be attributed to 'infectiousness' category of social contagion theory.

The calculation of replygap is worth discussing. We want to know how long it takes for an issue to get its first reply when the issue is raised right before the installation of either package (`data.table/tidy`). However, there are a few obstacles needing to be addressed first.

1) it's rare that an issue was raised right at the time point when the commit (inside which either `data.table/tidy` package is installed) was made
2) the timeliness of replying to an issue may vary drastically during the development of a package, hence taking the closest issue's reply-time as a substitution is not reasonable
3) for some issues, it took a significant amount of time to get a reply and in some cases no reply have been made to an issue, thus, averaging reply-time to previous issues is not a good substitution either.

This is a case where survival models fit. In this scenario, an issue can be compared to a patient and the first reply to death. We aim at finding out the time for an issue to get its first reply, i.e., the survival time of a patient. For each issue of either package, we record its raising time (timestamp at which an issue is raised), calculate its survival time, train a survival model for each package (`data.table/tidy`) using R package 'survival' [40] and then make predictions on each project (observation) at the timestamp when either `data.table/tidy` package was installed. In the end, we get the *Unrslvd* of each project for both packages.

Notice that for each project (observation), every variable is calculated at the time point the project installed that package (`data.table/tidy`) and that time point is defined as the first commit that contains the statement of installing that package. In addition, for each observation, every predictor with choice property (Table I) needs to be calculated for both packages, e.g., *Unrslvd* needs to be calculated for both `data.table` and `tidy`, which ends up as *Unrslvd.datatable* and *Unrslvd.tidy*.

V. Results

We found 24K projects (7K for `tidy`, 17K for `data.table`) that installed either `data.table` or `tidy` between June, 2009 and January, 2018. After removing

forks, we were left with 8K projects (3K for `tidy`, 5K for `data.table`). These projects serve as observations in our choice model. Table III includes basic statistics for independent variables we used in the model. We use the R package 'mlogit'[5] [41] to train the choice model using 11 predictor variables (described above) and a dependent variable representing the choice.

Very high correlations among predictors (above 0.9) occurred between *Prx2DT* and *Prx2TD*. High correlations may lead to unstable and hard-to-interpret models and need to be addressed. Since we do not have any *a priori* theory-derived reasoning for removing one or the other variable, we removed *Prx2DT*. The modeling results do not change if we remove the other variable instead. Table IV shows the resulting choice model. Below we interpret these results by discussing each predictor variable separately.

TABLE III
BASIC STATISTIC FOR INDEPENDENT VARIABLES

Variable	median	mean	std.dev
Cmts	2	43.92	621.64
Aths	1	2.03	7.93
C (boolean)	0	9.07e-03	9.48e-02
Prx2DT	0	0.15	0.92
Prx2TD	0	0.59	2.70
AthPrx2DT	0	6.79e-2	0.17
AthPrx2TD	0	0.11	0.24
CumNum.datatable	2.67e+03	2.64e+03	1.84e+03
CumNum.tidy	221	7.84e+02	9.05e+02
RplGp.datatable	1.40	1.38	0.25
RplGp.tidy	1.93	1.97	0.30
Unrslvd.datatable	0.27	0.28	5.96e-2
Unrslvd.tidy	0.17	0.20	9.66e-2
StckExch.datatable	128	124.48	7.78
StchExch.tidy	157	151.10	11.07

TABLE IV
THE FITTED COEFFICIENTS. $R^2 = 0.15$ $n = 8k$

Variable	Estimate	Std. Error	p-val
tidy:(intercept)	-7.02	0.25	2.20e-16
CumNum	1.63e-04	1.42e-05	2.20e-16
Unrslvd	-2.52	0.46	1.72e-08
RplGp	-0.42	7.76e-02	8.46e-08
StckExch	0.25	0.01	2.20e-16
tidy:Cmts	-4.63e-04	2.34e-04	4.76e-2
tidy:Aths	2.14e-03	7.40e-03	0.77
tidy:C	-0.66	0.29	2.03e-3
tidy:Prx2TD	0.19	2.91e-02	6.37e-11
tidy:AthPrx2TD	1.32	0.14	2.20e-16
tidy:AthPrx2DT	-2.67e-02	0.19	0.89

StckExch: the coefficient is 0.2, indicating that as the number of high quality questions on StackExchange increases, the likelihood that a project would choose to use the technology goes up if all other factors are equal. For illustration, if the number of high quality questions increases by one standard deviation of 8 questions from a median value of 128 for `data.table`, the chances of choosing `data.table` go up from 0.55 to 0.90 for a project that has all other predictors at their median values.

This result aligns well with the social contagion theory that posits that increased adoption is a product of increased exposure. If, however, we include an additional predictor counting the total number of questions (of high and low quality), adoption will not increase. It appears to be counter-intuitive as more exposure should increase adoption. However, when developers want to solve an issue related to the functionality of the R `data.frame`, they often may not search on StackExchange, but use a general search engine and follow links to StackExchange. The total number of posts, therefore, may be not visible to developers, only the set of posts that the search engine deems to be of sufficiently high quality. The number of posts (questions), may, therefore, not be a good proxy of exposure.

Unrslvd: the coefficient is -2.5, indicating that as the fraction of unresolved issues increases, the likelihood that the package is adopted decreases. If the fraction of unresolved issues increases by one standard deviation of 6% from a median value of 27% for `data.table`, the chances of choosing `data.table` go down by four percent from 0.55 to 0.51. The rationale here appears to be clear. Before installing a package, a developer may be concerned if bugs are not being fixed. A package with many unaddressable issues may be perceived as high risk, since if the developer may encounter same or new issues in that package, these issues would be less likely to be resolved.

AthPrx2TD: the coefficient is 1.3, indicating that the closer (through author network) a project is to authors of the package `tidy`, the more likely they are to choose `tidy` over `data.table`. If the proximity to `tidy` in author network increases by one standard deviation of 0.24 from a median value of 0 (e.g., a project that has four authors and one of them cooperates with `tidy`'s developers, but not with any of `data.table`'s developers), the chances of choosing `tidy` go up by nine percent from 0.45 to 0.54. This finding supports the basic premise of the social contagion hypothesis that developers' choice is affected by the environment they are in. If a project has a large fraction of authors who are developers of a package or who work with at least one developer of that package, then this project tends to adopt that package over other alternatives. This may be caused by, for example, many of authors who already have experience in using that package or who have heard of that package. However, **AthPrx2DT** is not statistically significant. One reason may be that `data.table` is a more widely deployed package and the deployments may play a larger role than the social connections. Also, each community of users and developers may be different. For example, the `tidy` community may have more social interactions than `data.table` community. Furthermore, the exposure in the `tidy` community may come from a much larger set of packages in `tidyverse`, while `data.table` does not have an equivalent brand that involves a wider variety of tools besides the data handling.

C: the coefficient is -0.6, indicating that a project containing

[5]https://cran.r-project.org/web/packages/mlogit/vignettes/mlogit.pdf

at least one C file is less likely to choose `tidy`. The chances of choosing `data.table` go up by 15 percent from 0.55 to 0.70. The finding is consistent with our hypothesis that if an R project has a need for performance, it implements functionality natively in C language.

RplGp: the coefficient is around -0.4, indicating that the more quickly a package's issue gets a response, the more likely that this package will be chosen. If the number of days until first response to an issue increases by one standard deviation of 0.25 from a median value of 1.4 for `data.table`, the chances of choosing `data.table` go down by three percent from 0.55 to 0.52 for a project with all median values. The time until first response is not as readily visible to developers as most other measures that we used, so developers may not be able to observe it when making a choice. However, it appears to be a reasonable proxy for project's reactions to external requests that could be easily gleaned by reading through some of issues on the issue tracker. A well maintained package is more likely to respond to new issues quickly and thoroughly, leaving a good impression and, thus, increasing the likelihood of being adopted. This has implications for designing project dashboards intended to make key project attributes more visible.

Prx2TD: the coefficient is 0.2, indicating that the closer (through dependency network) a project is to package `tidy`, the more likely it is to choose `tidy` over `data.table`. If proximity to tidy in dependency network increases by one standard deviation of 2.7 from a median value of 0 (e.g., a project installs/uses three packages that are in first layer downstream from `tidy`), the chances of choosing `tidy` go up by 12 percent from 0.45 to 0.57. It supports our hypothesis that the supply chain influences projects' choices. A project tends to install a package if it already has installed some packages that depend on it, i.e., if a project uses downstream packages of a package, it is more likely to use the upstream package than other alternatives. Being familiar with downstream packages may reduce the overheads or learning curve of using an upstream package, which generates the advantage of adopting upstream package over other choices.

CumNum: the coefficient is 1.6e-4, indicating that a larger number of deployments of a package in the past will make it more likely to be adopted. If the number of deployments increases by one standard deviation of 1840 projects from a median value of 2670 projects for `data.table`, the chances of choosing `data.table` go up by seven percent from 0.55 to 0.62 for a project that has all other values at the median. A larger number of overall deployments, on one hand, increases the chance for a package to be known by adopters. On the other hand, from the perspective of adopters, more deployments usually insinuate a stable and mature product (though it is not clear if the number of deployments is visible to a developer), and enhances adopters' confidence in this package. Either of these reasons leads to adoption of widely deployed package as predicted by the social contagion theory.

Cmts: the coefficient is not statistically significant at 0.005 level recommended for reproducibility [42].

We also find that the number of authors of adopting project does not affect the choices between these two technologies. Social contagion theory does not suggest that this predictor should have an effect, but it could be that project activity (which has a substantial correlation with the number of authors), may already account for the differences in propensity to chose `tidy` over `data.table` and cause the variation in the number of authors.

Choice models are explanatory, but we can also use them to do prediction. The 10-fold cross-validation done by randomly splitting projects into 10 parts and fitting the model with predictors listed in IV on nine parts and predicting on the remaining part yielded a reasonable AUC of 76%. Average accuracy was 70% with balanced Type I and II errors (obtained by choosing predicted probability cutoff of 0.45).

VI. Limitations

Empirical studies have to be interpreted carefully due to a number of inherent limitations. Here we highlight some of the potential issues and how we try to address them.

To obtain an unbiased picture of technology spread representing the entirety of the projects, we consider a very large collection of projects. While large, our sample can not be complete as many projects do not publish their code and our collection may have missed even some public projects. We, at least, have updated the collection prior to analysis to ensure that we have the latest commits for projects that involve R language source code files. The sample we have should limit the findings to projects that share their version control data on one of the many forges, such as GitHub, BitBucket, GitLab, Bioconductor, SourceForge, etc. However, it may not be representative of the entire universe of projects, especially projects that do not share their version control data.

We have selected only projects with extension $[rR]$, but some older projects may use extension $[sS]$ indicating the historic name for R language, or some other source code without any (known) extension.

Regular expressions can capture most of the install statement in .r/R file, however, in some cases the install statement may be missed due to a dynamic way the installation is specified in a function:

```
ipak <- function(pkg){
    new.pkg <- pkg[!(pkg %in% installed.
        packages()[, "Package"])]
    if (length(new.pkg)) install.packages(new.
        pkg, dependencies = TRUE)
    sapply(pkg, require, character.only = TRUE
        )
}
# usage
packages <- c("ggplot2", "plyr", "reshape2", "
    RColorBrewer", "scales", "grid")
ipak(packages)
```

Moreover, regular expressions can over capture an install statement in some cases, e.g., install statements that are commented out are captured by regular expressions, files that are not used in project may also contain installment statements

that are captured by regular expression. Since R language requires comment character '#' on each line, this may not be as severe an issue because we ensured that the matched install is not preceded by the comment character.

These errors would affect the dependency networks we construct and under-count/over-count the number of projects using the choices we are looking for. Developer identities may not be spelled the same way and that may affect the author network [43]. We have tried to address these and other issues encountered when dealing with operational data from software repositories and big data [44]–[46].

Its important to note that the particular operationalizations of the concepts from social contagion theory are but one possibility to measure these concepts. Each measure may capture other concepts besides the one it is intended for. The correlations among predictors may lead to unstable models that are hard to interpret. We address that by carefully considering various interpretations of the measures, conducting exploratory analysis of the obtained measures, selecting a subset that does not pose threats to model stability and investigating compliance with model assumptions including inspecting for outliers, non-homogeneous variance, and performing general model diagnostics. We also model the first choice, but it is also reasonable to model the full set of choices made. In the latter case, we need to include the third option, i.e., projects choosing both packages: `tidy` and `data.table`. We fitted a variety of alternatives to ensure that the reported results are not affected.We only present the results for two alternatives due to space considerations, but we have applied choice model on several other R packages.

VII. Related Work

The closest related work involves studies of use and migration of software libraries. A number of metrics and approaches were proposed to mine and explore usage and migration trends. A software library encapsulates certain functionality that is then used by applications (or other libraries). The application may benefit from extra functionality or performance in the new libraries that may be created later, but switching to a new library (library migration) involves some recoding of the application [47]–[51]. Most prior work, therefore, focused on costs and benefits of library migration [52]–[59]. Similarly to that work we ask why developers chose a new library. In contrast to prior work, we construct new predictors of adoption (e.g., technical and author dependency networks, breadth of deployment, quality of support measured through StackExchange, issue number and response times). that are based on sound theoretical foundation and we use choice models to understand how macro trends at the scale of the entire SSC emerge from actual decisions the individual developers make to select a specific software technology.

Approaches to detect library usage include issue report analysis [52]. As in prior work we detect usage by searching for library statements in source files of projects [53].

De la Mora *et al.* [54] introduce an interface to help developers choose among the libraries by displaying their popularity, release frequency, and recency. While building on this research, we add novel network, deployment, and quality measures that would inform developer choice. More importantly, we radically improve the ability of developers to make informed decisions by providing a statistical model that explains which of these measures matter and how they affect the choice.

Prior studies that examined technology choices have used a variety of approaches ranging from surveying developer preferences [60], to mining version control and issue tracking repositories [52]–[54]. Similarly, we mine version control data, but at a larger scale of all projects with public version control data that include R language files. This allows us to construct complete software supply chains that depict end-to-end technical and social dependencies.

VIII. Conclusions

The methodology of applying software supply chain concepts, social contagion theory, and choice models to investigate software technology selection appears to have provided a number of potentially useful insights in the present case study of two data manipulation technologies within R language. More specifically, the methodology can show which factors are influential in decision-makers' choices between software technologies and demonstrate the need to account not only for the properties of the choice but also of the chooser and of the importance of the supply chain dependencies and information flows. This study introduces the concept of software supply chains and demonstrates how a software supply chain for the entire open source ecosystem can be constructed from public version control systems. Additionally, by taking a social contagion perspective and employing the econometric technique of choice models, we explicate a parsimonious model that is capable of modeling software technology choices. The findings of this study have wide reaching implications for the software engineering community as well as those who study traditional supply chains. For example, the ability to model and understand which aspects of a network of software supply chain or physical supply chain partners and affiliates influence uptake and spread of a given artifact (e.g., package or product) might help contributors adjust their contributions in a way to maximize their reach, while also extending the viability and propagation of a core package or product. This notion is consistent with our findings that a number of characteristics of a developer and properties of technology are found to be important in the choice between major alternatives. More specifically, packages with large number of overall adopters, higher responsiveness to new issue, and more stack exchange questions with high quality are more likely to be chosen. In contrast, packages with more unresolved issues/bugs tend to become less popular. Furthermore, from the perspective of decision-makers (project), technical features and proximity to a technology in both the dependency network and author collaboration network increase the probability of adoption.

Source code and data for this study will be made public to facilitate reproducibility and wider adoption of the proposed methodology.

REFERENCES

[1] E. Von Hippel, "Innovation by user communities: Learning from open-source software," *MIT Sloan management review*, vol. 42, no. 4, p. 82, 2001.

[2] G. von Krogh, S. Spaeth, and K. R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," *Research Policy*, vol. 32, no. 7, pp. 1217 – 1241, 2003, open Source Software Development. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0048733303000507

[3] B. Kogut and A. Metiu, "Opensource software development and distributed innovation," *Oxford Review of Economic Policy*, vol. 17, no. 2, pp. 248–264, 2001. [Online]. Available: http://dx.doi.org/10.1093/oxrep/17.2.248

[4] J. West and S. Gallagher, "Patterns of open innovation in open source software," *Open Innovation: researching a new paradigm*, vol. 235, no. 11, 2006.

[5] J. Holdsworth, *Software Process Design*. McGraw-Hill, Inc., 1995.

[6] B. Farbey and A. Finkelstein, "Exploiting software supply chain business architecture: a research agenda," 1999.

[7] S. H. Huang, M. Uppal, and J. Shi, "A product driven approach to manufacturing supply chain selection," *Supply Chain Management: An International Journal*, vol. 7, no. 4, pp. 189–199, 2002. [Online]. Available: https://doi.org/10.1108/13598540210438935

[8] S. Kalish, "A new product adoption model with price, advertising, and uncertainty," *Management Science*, vol. 31, no. 12, pp. 1569–1585, 1985. [Online]. Available: http://www.jstor.org/stable/2631795

[9] D. M. Russell and A. M. Hoag, "People and information technology in the supply chain: Social and organizational influences on adoption," *International Journal of Physical Distribution & Logistics Management*, vol. 34, no. 2, pp. 102–122, 2004.

[10] M. Christopher and H. Lee, "Mitigating supply chain risk through improved confidence," *International Journal of Physical Distribution & Logistics Management*, vol. 34, no. 5, pp. 388–396, 2004. [Online]. Available: https://doi.org/10.1108/09600030410545436

[11] C. M. Angst, R. Agarwal, V. Sambamurthy, and K. Kelley, "Social contagion and information technology diffusion: the adoption of electronic medical records in us hospitals," *Management Science*, vol. 56, no. 8, pp. 1219–1241, 2010.

[12] M. Samadi, A. Nikolaev, and R. Nagi, "A subjective evidence model for influence maximization in social networks," *Omega*, vol. 59, pp. 263 – 278, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305048315001425

[13] A. A. Chhajed and S. H. Xu, "Software focused supply chains: Challenges and issues," in *Industrial Informatics, 2005. INDIN'05. 2005 3rd IEEE International Conference on*. IEEE, 2005, pp. 172–175.

[14] R. J. Ellison and C. Woody, "Supply-chain risk management: Incorporating security into software development," in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*. IEEE, 2010, pp. 1–10.

[15] F. M. Bass, "A new product growth for model consumer durables," *Manage. Sci.*, vol. 50, no. 12 Supplement, pp. 1825–1832, Dec. 2004. [Online]. Available: http://dx.doi.org/10.1287/mnsc.1040.0264

[16] E. M. Rogers, "Innovation in organizations," *Diffusion of innovations*, vol. 4, pp. 371–404, 1995.

[17] R. G. Fichman, "Going beyond the dominant paradigm for information technology innovation research: Emerging concepts and methods," *Journal of the association for information systems*, vol. 5, no. 8, p. 11, 2004.

[18] P. J. DiMaggio and W. W. Powell, "The iron cage revisited: Institutional isomorphism and collective rationality in organizational fields," *American Sociological Review*, vol. 48, no. 2, pp. 147–160, 1983. [Online]. Available: http://www.jstor.org/stable/2095101

[19] R. S. Burt, "Social contagion and innovation: Cohesion versus structural equivalence," *American Journal of Sociology*, vol. 92, no. 6, pp. 1287–1335, 1987. [Online]. Available: https://doi.org/10.1086/228667

[20] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 1277–1286. [Online]. Available: http://doi.acm.org/10.1145/2145204.2145396

[21] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 356–366. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568315

[22] D. McFadden *et al.*, "Conditional logit analysis of qualitative choice behavior," 1973.

[23] W. A. Kamakura and G. J. Russell, "A probabilistic choice model for market segmentation and elasticity structure," *Journal of Marketing Research*, vol. 26, no. 4, pp. 379–390, 1989. [Online]. Available: http://www.jstor.org/stable/3172759

[24] J. A. Hausman, G. K. Leonard, and D. McFadden, "A utility-consistent, combined discrete choice and count data model assessing recreational use losses due to natural resource damage," *Journal of Public Economics*, vol. 56, no. 1, pp. 1 – 30, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0047272793014157

[25] K. Talluri and G. van Ryzin, "Revenue management under a general discrete choice model of consumer behavior," *Manage. Sci.*, vol. 50, no. 1, pp. 15–33, Jan. 2004. [Online]. Available: http://dx.doi.org/10.1287/mnsc.1030.0147

[26] T. J. Gilbride and G. M. Allenby, "A choice model with conjunctive, disjunctive, and compensatory screening rules," *Marketing Science*, vol. 23, no. 3, pp. 391–406, 2004. [Online]. Available: https://doi.org/10.1287/mksc.1030.0032

[27] D. McFadden and K. Train, "Mixed mnl models for discrete response," *Journal of Applied Econometrics*, vol. 15, no. 5, pp. 447–470.

[28] S. T. Berry, "Estimating discrete-choice models of product differentiation," *The RAND Journal of Economics*, vol. 25, no. 2, pp. 242–262, 1994. [Online]. Available: http://www.jstor.org/stable/2555829

[29] K. Small and H. Rosen, "Applied welfare economics with discrete choice models," *Econometrica*, vol. 49, no. 1, pp. 105–30, 1981. [Online]. Available: https://EconPapers.repec.org/RePEc:ecm:emetrp:v:49:y:1981:i:1:p:105-30

[30] A. Mockus, "Amassing and indexing a large sample of version control systems: Towards the census of public source code history," in *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, ser. MSR '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 11–20. [Online]. Available: http://dx.doi.org/10.1109/MSR.2009.5069476

[31] Y. Ma, T. Dey, J. M. Smith, N. Wilder, and A. Mockus, "Crowdsourcing the discovery of software repositories in an educational environment," *PeerJ Preprints*, vol. 4, p. e2551v1, 2016.

[32] J. Greenfield and K. Short, "Software factories: assembling applications with patterns, models, frameworks and tools," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 2003, pp. 16–27.

[33] E. Levy, "Poisoning the software supply chain," *Security & Privacy, IEEE*, vol. 1, no. 3, pp. 70–73, 2003.

[34] S. Chacon and B. Straub, *Pro Git*, 2nd ed. Berkely, CA, USA: Apress, 2014.

[35] M. L. Christopher, *Logistics and Supply Chain Management*. London: Pitman Publishing, 1992.

[36] S. Chopra and P. Meindl, "Supply chain management. strategy, planning & operation," in *Das Summa Summarum des Management*. Springer, 2007, pp. 265–275.

[37] J. T. Mentzer, W. DeWitt, J. S. Keebler, S. Min, N. W. Nix, C. D. Smith, and Z. G. Zacharia, "Defining supply chain management," *Journal of Business Logistics*, vol. 22, no. 2, pp. 1–25. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2158-1592.2001.tb00001.x

[38] H. L. Lee, V. Padmanabhan, and S. Whang, "Information distortion in a supply chain: The bullwhip effect," *Management Science*, vol. 43, no. 4, pp. 546–558, 1997. [Online]. Available: https://doi.org/10.1287/mnsc.43.4.546

[39] J. Buurman, *Supply chain logistics management*. McGraw-Hill, 2002.

[40] T. T, *survival: A Package for Survival Analysis in S*, 2015, r package version 2.38. [Online]. Available: https://CRAN.R-project.org/package=survival

[41] Y. Croissant, *mlogit: multinomial logit model*, 2013, r package version 0.2-4. [Online]. Available: https://CRAN.R-project.org/package=mlogit

[42] D. J. Benjamin, J. O. Berger, M. Johannesson, B. A. Nosek, E.-J. Wagenmakers, R. Berk, K. A. Bollen, B. Brembs, L. Brown, C. Camerer *et al.*, "Redefine statistical significance," *Nature Human Behaviour*, vol. 2, no. 1, p. 6, 2018.

[43] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ser. MSR '06. New York, NY, USA: ACM, 2006, pp. 137–143. [Online]. Available: http://doi.acm.org/10.1145/1137983.1138016

[44] A. Mockus, "Engineering big data solutions," in *Proceedings of the on Future of Software Engineering*, ser. FOSE 2014. New York, NY, USA: ACM, 2014, pp. 85–99. [Online]. Available: http://doi.acm.org/10.1145/2593882.2593889

[45] M. Audris, "Software support tools and experimental work," in *Empirical Software Engineering Issues. Critical Assessment and Future Directions*. Springer, 2007, pp. 91–99.

[46] I. Gorton, A. B. Bener, and A. Mockus, "Software engineering for big data systems," *IEEE Softw.*, vol. 33, no. 2, pp. 32–35, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1109/MS.2016.47

[47] A. Hora and M. T. Valente, "Apiwave: Keeping track of api popularity and migration," in *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ser. ICSME '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 321–323. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2015.7332478

[48] Y. M. Mileva, V. Dallmeier, and A. Zeller, "Mining api popularity," in *Proceedings of the 5th International Academic and Industrial Conference on Testing - Practice and Research Techniques*, ser. TAIC PART'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 173–180. [Online]. Available: http://dl.acm.org/citation.cfm?id=1885930.1885952

[49] B. E. Cossette and R. J. Walker, "Seeking the ground truth: A retroactive study on the evolution and migration of software libraries," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 55:1–55:11. [Online]. Available: http://doi.acm.org/10.1145/2393596.2393661

[50] R. Lämmel, E. Pek, and J. Starek, "Large-scale, ast-based api-usage analysis of open-source java projects," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC '11. New York, NY, USA: ACM, 2011, pp. 1317–1324. [Online]. Available: http://doi.acm.org/10.1145/1982185.1982471

[51] H. A. Nguyen, T. T. Nguyen, G. Wilson, Jr., A. T. Nguyen, M. Kim, and T. N. Nguyen, "A graph-based approach to api usage adaptation," *SIGPLAN Not.*, vol. 45, no. 10, pp. 302–321, Oct. 2010. [Online]. Available: http://doi.acm.org/10.1145/1932682.1869486

[52] S. Kabinna, C.-P. Bezemer, W. Shang, and A. E. Hassan, "Logging library migrations: A case study for the apache software foundation projects," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 154–164. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901769

[53] C. Teyton, J.-R. Falleri, M. Palyart, and X. Blanc, "A study of library migrations in java," *Journal of Software: Evolution and Process*, vol. 26, no. 11, pp. 1030–1052. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1660

[54] F. L. de la Mora and S. Nadi, "Which library should i use?: A metric-based comparison of software libraries," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '18. New York, NY, USA: ACM, 2018, pp. 37–40. [Online]. Available: http://doi.acm.org/10.1145/3183399.3183418

[55] C. Teyton, J.-R. Falleri, and X. Blanc, "Mining library migration graphs," in *19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012*, 2012, pp. 289–298.

[56] T. T. Bartolomei, K. Czarnecki, R. Lämmel, and T. van der Storm, "Study of an api migration for two xml apis," in *Software Language Engineering*, M. van den Brand, D. Gašević, and J. Gray, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 42–61.

[57] T. Tonelli, Krzysztof, and Ralf, "Swing to swt and back: Patterns for api migration by wrapping," in *2010 IEEE International Conference on Software Maintenance*, Sept 2010, pp. 1–10.

[58] B. Dagenais and M. P. Robillard, "Semdiff: Analysis and recommendation support for api evolution," in *2009 IEEE 31st International Conference on Software Engineering*, May 2009, pp. 599–602.

[59] Y. M. Mileva, V. Dallmeier, M. Burger, and A. Zeller, "Mining trends of library usage," in *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, ser. IWPSE-Evol '09. New York, NY, USA: ACM, 2009, pp. 57–62. [Online]. Available: http://doi.acm.org/10.1145/1595808.1595821

[60] E. E. Anderson, "Choice models for the evaluation and selection of software packages," *Journal of Management Information Systems*, vol. 6, no. 4, pp. 123–138, 1990.