A Methodology for Analyzing Uptake of Software Technologies Among Developers

Yuxing Ma, *StudentMember, IEEE,* Audris Mockus, *Fellow, IEEE,* Russel Zaretzki, Randy Bradley and Bogdan Bichescu

Abstract—Motivation: The question of what combination of attributes drives the adoption of a particular software technology is critical to developers. It determines both those technologies that receive wide support from the community and those which may be abandoned, thus rendering developers' investments worthless. Aim and Context: We model software technology adoption by developers and provide insights on specific technology attributes that are associated with better visibility among alternative technologies. Thus, our findings have practical value for developers seeking to increase the adoption rate of their products. Approach: We leverage social contagion theory and statistical modeling to identify, define, and test empirically measures that are likely to affect software adoption. More specifically, we leverage a large collection of open source version control repositories (containing over 4 billion unique versions) to construct a software dependency chain for a specific set of R language source-code files. We formulate logistic regression models, where developers' software library choices are modeled, to investigate the combination of technological attributes that drive adoption among competing data frame (a core concept for a data science languages) implementations in the R language: `tidy` and `data.table`. To describe each technology, we quantify key project attributes that might affect adoption (e.g., response times to raised issues, overall deployments, number of open defects, knowledge base) and also characteristics of developers making the selection (performance needs, scale, and their social network). Results: We find that a quick response to raised issues, a larger number of overall deployments, and a larger number of high-score StackExchange questions are associated with higher adoption. Decision makers tend to adopt the technology that is closer to them in the technical dependency network and in author collaborations networks while meeting their performance needs. To gauge the generalizability of the proposed methodology, we investigate the spread of two popular web JavaScript frameworks `Angular` and `React`, and discuss the results. Future work: We hope that our methodology encompassing social contagion that captures both rational and irrational preferences and the elucidation of key measures from large collections of version control data provides a general path toward increasing visibility, driving better informed decisions, and producing more sustainable and widely adopted software.

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

1 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. Open source allows developers, by building on the existing work of others, to focus on their own innovation [1], [2], [3], [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 diminish in favor of the now more popular alternative technology. As a consequence, both the importance of a developer's product and their reputation may suffer. To remedy these two risks, developers must understand how attributes of their software products may be perceived among potential and actual downstream adopters (consumers of the technology), especially in relation to alternative, competing technologies these adopters may have. 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 updates (patches and 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], [8], [9], [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 or biases, which can affect the ultimate outcome of wide (or narrow) adoption and/or entire abandonment of formerly popular technologies. These decisions are not

- *Y. Ma is with the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville. E-mail: yma28@vols.utk.edu*
- *A. Mockus is with the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville. E-mail: see http://mockus.org*
- *R. Zaretzki, R. Bradley and B. Bichescu are with the Department of Business Analytics and Statistics, University of Tennessee.*

only based on technical merit but the availability and accessibility of relevant information along with the tastes of consumers(adopters). 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 based on what components they are aware of and how much time or inclination they have to investigate the relative merits of the possible choices. This suggest the potentially strong influence of default choice well documented in behavioural economics. 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 and their specific position within them.

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 models for specific technology choices and, thus, characterizing the implicit primary factors (social, behavioural, and rational) they may use to make their decision.

Despite the practical and theoretical importance of the question how developers make technology choices, the extant literature does not offer theoretical guidance on this subject. We, therefore, leverage social contagion theory, which has been effective, among other things, in clarifying key aspects of organizational adoption of technology [11], [12]. Social contagion theory mimics models of the spread of contagious diseases but apply them in the behavioral/-social context instead of the physiological one. The first key concept is exposure or how widespread the infectious agent is in the population. In our case the agent is a specific technology and the population is the entire collection of FLOSS repositories. Exposure is critical in epidemiology because without exposure a disease can not spread. This brings us to

RQ1: Does the exposure to a technology, such as the number of FLOSS repositories in existence, the rate at which new repositories are adopting this technology, or the number of high-score questions on StackExchange affect the decisions of the developers to adopt that technology?

The second key concept is infectiousness: a highly virulent agent is more likely to spread in a population. We deal with technologies (groups of packages), so in our case we would like to establish:

RQ2: Will extremely attractive technology (with few open issues, short response times to issues or pull requests, heavy activity and many authors), be more likely to be adopted?

The final concept is proximity: some infectious agents may not survive the travel through air or physical barriers, thus halting their spread. In our case, the distance from a developer to a technology is not physical, but it may be represented by the technological constraints (lack of compatibility with other technologies the developer already uses), need for certain performance characteristics, or a social distance to collaborators who are working with other developers already exposed to the technology or a related one. Hence:

RQ3: Will proximity of a developer or a project to a technology increase the rate of adoption? More specifically, **RQ3a:** will the proximity of a developer to a related technology used by a developer increase the chances of adoption; **RQ3b:** will the proximity of a developer to collaborators who already use the technology or a related one increase the chances of adoption?; **RQ3c:** will the performance requirements of the project a developer is working on increase the chances of adoption of a technology that has the desired performance attribute?.

To answer RQs, we need to collect data on the actual choices made by developers, operationalize key theory-based measures, and reconstruct the past states (historical states before adoption) of all public software projects that may choose the technology under study. For example, for a project that chooses Technology A in January 2014, we need to establish how many other projects have used A before that date (exposure), what average response time to issues the project had at that time (infectiousness), and what actions the developer making the choice to add the dependence had prior to that point in time, including her social network, technology network, etc.

To exemplify the proposed methodology, we investigate the 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[1] `data.table` and `tidy`. (a more detailed introduction of these two packages is given in Sec. 4). To gauge the generalizability of the proposed methodology, we applied[2] our approach to investigate the spread of two modern popular web JavaScript frameworks Angular and React.

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 regression 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 score 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

1. We use 'package' in the rest of our paper as a synonym for 'technology', since most software technologies are implemented in package format for use and 'package' is more appropriate to use in analysis.

2. Source code and result are provided in https://github.com/ssc-oscar/PackageAdoptionAnslysis

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. 2 we introduce the diffusion of innovation, social contagion, and the application of choice models. In Sec. 3, we describe the dataset and how we operationalize software supply chain. Choice model theory and our candidate technology are introduced in Sec. 3.5 and Sec. 4.1 respectively. In Sec. 4, operationalization of attributes of choice model is illustrated. Sec. 5 describes and interprets the result of applying the choice model. Related work is discussed in Sec. 7 and major limitations are considered in Sec. 6. We summarize our conclusions and contribution in Sec. 8.

2 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"[3].

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]. Notice, that the susceptible to influence of prior adopters represents non-rational behaviour. Rational behaviour would require developer to choose the best technology irrespective of social influences. It may also represent cognitive bias of the default choice. The developer may not know about the alternatives if their social or technical networks do not present them with an encounter with alternatives. This would represent the irrational bias toward default choice. 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 the 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. A variation of multinomial regression models also called choice models [22] can be used to describe the behavior of a decision maker given a set of alternatives. Choice models have been used successfully in the fields of marketing [23], [24], [25], [26] and economics [27], [28], [29] to understand how consumers make choices. Adapting and applying these regression models to technology adoption, we focus on a developer, or more precisely, a software project as a decision maker. The actual decision is operationalized as the first among the alternative technologies that a project in a commit modifying one of the files within a repository. As with the social contagion theory, two types of predictors can be included: properties of the choice (i.e., the technology) and properties of a decision maker (i.e., the project or individual developer).

Equiped with this theoretical and modeling framework, we set out to address RQ1 and RQ2 by empirically characterizing the spread of software technology through analysis of a very large collection of version control data introduced in [30] which is referred to as WOC-DATA in this paper. WOC-DATA is used to construct the SSC [31], [32] 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.

3 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 [33]. 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

3. 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.2993758, IEEE Transactions on Software Engineering

string, a commit timestamp, and the commit message. Fig. 1 illustrates relationships among objects described above.

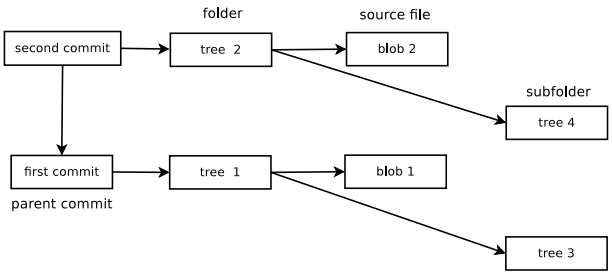

Fig. 1: Git objects graph

We utilize all Git objects (1.1 billion commits and 4 billion of blobs and trees) from WOC-DATA 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.

3.1 Software supply chain

In traditional supply chains [34], [35], [36], [37], [38], 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 the 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.

3.2 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 existing in the database at the time of the analysis 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

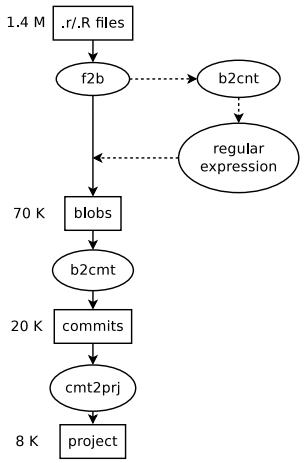

Fig. 2: Project discovery

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

3.3 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.

3.4 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. 4.1) 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-score content that is likely to be referred to from search engines.

3.5 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 last 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 this paper we applied the mixed logit model to study developers' choice over analogous R packages (`data.table` v.s. `tidy`). While many variations of choice models exist, the mixed logit model has the fewest assumptions on the distribution of the choice. Here we are not trying to solve the classical choice model which, for example, assumes a complete knowledge about the alternatives and produces implicit utility function. Instead, we simply look for factors that strongly affect the decisions developers make, whether these factors may be rational or related to cognitive or social biases.

3.6 Issues

It is 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. This factor belongs to a set of rational choices. To measure it we collect the issues reported during the development of `data.table` and `tidy` packages. Since both packages are hosted on GitHub, we use GitHub API to scrape all issues[4] reported for both packages. We collected 2.6k issues for the `data.table` and 1.6k issues for the `tidy`.

4 Case Study

4.1 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, therefore, 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`.

Apart from the `dataframe` package that is a part of core R language, `data.table` and `tidy*` are the two other most popular packages for data manipulation[5]. 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 here as the `tidy` package. 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 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.

In addition to the case study in R domain, we selected another case study focusing on the Javascript ecosystem. JavaScript[6] is the most popular programming language and has been commonly used in developing web applications, where one or more web frameworks were involved. Among all successful web frameworks, `Angular` and `React` are two of the most successful and widely supported by big technology companies (Google[7], Facebook[8]). Together, the massive amounts of adoption data, similar functionalities and corporate support from leading companies, make `Angular` and `React` a suitable comparison group for the investigation of technology adoption. The selection of the second case study was based on concerns about the generalizability of the findings from the R domain to other ecosystems. We, therefore, attempted to find a case that is radically different in a number of dimensions. We contrast the domain, scale, and governance principles in the second case study. Specifically, the mundane data management tasks supported by `data.table/tidy` and the need to create an engaging user interface in `Angular/React` contrast not only the domain of application, but also the types of functionality that is being implemented. Regarding scale, the R ecosystem is used by a relatively small number (a few thousand) of data analysts, while the `Angular/React` ecosystems are used by tens of thousands of web developers. Both `tidy` and `data.table` follow the community development model. `Angular` and `React`, in contrast, are primarily funded by corporate sponsors.

4. Notice that GitHub API treats pull requests as issues (https://developer.github.com/v3/issues/), and we dropped the pull requests from all collected issues.

5. Based on WOC-DATA in https://bitbucket.org/swsc/overview/src/master/deps/README.md

6. https://stackify.com/popular-programming-languages-2018/

7. https://en.wikipedia.org/wiki/Angular˙(web˙framework)

8. https://en.wikipedia.org/wiki/React˙(web˙framework)

4.2 Data collection

We leveraged both WOC-DATA and WOC [30] infrastructure for data collection and filtering. According to [30], WOC-DATA approximates the entirety of public version control and includes major forges such as GitHub, BitBucket, GitLab, Bioconductor, SourceForge, the now defunct Googlecode, and many others and contained over 46M projects at the time of analysis. WOC-DATA production involves discovering [39] and cloning the projects, extracting Git objects from each repository, and then storing these objects in a scalable key-value database.

WOC is an open source data mining infrastructure [9] [30], which provides not only APIs for extraction of development data on various levels for open source projects, but also offers intermediate collections and results which are extremely useful for studying domain knowledge. In particular, as illustrated in Fig. 2 and Sec. 3.2, we used the collection of all R file names and maps of file-to-blob, blob-to-content, blob-to-commit and commit-to-project to discover our targeted projects. Meanwhile, for each project, we found the first commit in which `data.table` or `tidy` was imported by sorting the commit time. By querying the content for this first commit, we learned the author of the commit, the commit message, etc. We set this first commit time as the end point of our analysis for each project, as it is at this time that the developer's choice between `data.table` or `tidy` becomes clear. For every targeted project, we evaluate explanatory behavior, activities, and relationships as computed before this end point, so that the analysis considers all factors as they were at the time the choice was made.

We further refined the list of projects because a large fraction involved forks of other projects. One of the most typical ways to make contributions to the development of a project on GitHub is by creating a fork of the project, making changes to this clone, and then sending a pull request to the original project. As a result, a popular project may have hundreds of forks that share a large portion of the source code and commit history. These forks are not equivalent to the original projects from which these forks were created and, therefore, were removed from consideration. To detect and delete these forks, we classify projects based on common commits, i.e., a pair of projects are linked if they have at least one commit in common. Based on these links, a transitive closure produces disjoint clusters. Each cluster represents a single observation in our study. The date when the first blob containing the focal technology was created is used as the technology adoption date for this cluster.

The extraction of the supply chain data for these two packages started from 1.4M R files in the entire WOC-DATA collection, with 70K blobs (versions of these files) that contained information pointing to the installation of either package. As a result, fewer than 20K commits were found that produced these 70K blobs. Then by using the commit-to-project map in WOC, we identified around 24K projects (7K for `tidy` and 17K for `data.table`) that installed either `data.table` or `tidy` between June, 2009 and January, 2018. After removing forks, we were left with a total of 8,303 projects (2,660 for `tidy` and 5,643 for `data.table`). Furthermore, we removed 4,961 `data.table` adoptions occurring prior to June 16, 2014, the date when `tidy` was first introduced. Thus, while `data.table` predates the introduction of `tidy`, our analysis focuses on the period of time when both choices are available.

We applied a series of similar procedures (as described above for `data.table` and `tidy`) on `Angular` and `React`, and found the list of projects, which at some point in the past have adopted either `Angular` or `React`. The only difference in the `Angular` and `React` case is that Javascript projects (deployed via NPM) usually record dependency information in a specific file named 'package.json', and we went through all of the versions of the 'package.json' file in WOC to identify projects that adopted either `Angular` or `React`. In summary, we identified 292494 projects (100894 for `Angular`, 191600 for `React`) that adopted either Angular or React.

4.3 Operationalizing Attributes for Regression 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 characteristics of the developers making the choices. To this end, we propose 11 variables that seek to capture the key factors that may have influenced developers' choice of `data.table` or `tidy`, `Angular` or `React`. To streamline the presentation of the operationalizations of the variables in both studies, we only describe the operationalization for the R domain and note differences, if any. All of these attributes apply to the Javascript domain as well. These variables are listed in Table 1 and described in more detail below.

 # commits and authors (Cmts & Aths) are the number of commits and authors, respectively, and aim to capture the size of a project, as project size which may affect package adoption. Larger projects, for example, may prefer less controversial, more conservative package choices. This is a quality of the choice, so it would most closely fit under the "infectiousness" category according to the social contagion theory. We chose not to use lines of code (LOC) as a measure of size, since it has a less stable distribution than the number of commits, while, at the same time, being highly correlated with it. Operationally, for a particular adopter, we collected all commits prior to the end point (i.e., the first adoption of one of the two targeted packages) by applying the project-to-commit map followed by a time point filtering. We extracted the authors of these commits and counted the number of unique authors.

 # projects of deployments (CumNum) is the overall number of project deployments of `tidy` or `data.table`. A larger **CumNum** should increase the chance that a developer would be aware of, and get exposed to, a particular package and may influence the developer's package adoption decision. This measure falls within the "exposure" category of contagion, because it quantifies the chances that a developer may become a user of the technology. This is characterized as a factor that is not rational, as the project is hypothesized to be biased towards technology that they are more likely to encounter, not necessarily technology that would be optimal for that project. To assess **CumNum** for a given package, we counted the number of projects that adopted `data.table` and `tidy`, respectively, before a

9. https://github.com/ssc-oscar/Analytics

TABLE 1: Independent variables

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

decision was made by the developers of the package under evaluation.

 # open issues over all issues (Unrslvd) can be an indicator of package quality. A higher fraction of unresolved issues may indicate that the package has a significant number of problems, which, like a bad review, may undermine people's confidence in it. This is a quality of the choice, so we hypothesize that it relates to the "infectiousness" aspect of the social contagion paradigm. To measure this quantity, we leveraged GitHub API to collect all issues for `data.table` and `tidy` packages and filtered issues raised before the decision end point for each adopting project. We count the number of unresolved issues (issues that are still open) and normalize it over all issues raised before end point, because in general a project tends to have more issues and unresolved ones as its age grows, and we believe the averaged rate of unresolved issues is more reflective of a package's maintenance and quality.

Association of c code with a project (C) is used as a proxy for the requirement for high performance. Typically, computations that are too slow for the interpreted R language are implemented in C to improve performance. This is a requirement of the decision maker that would most closely fit under the "infectiousness" category because it likely indicates a strong preference for higher performance embodied by the `data.table` choice. This is a good example of a factor that may represent a rational choice for some decision makers. To measure this aspect, we applied commit-to-file map on every commit prior to end point for each project and filtered files with suffix '.[cC]'.

 # related questions on StackExchange with high score (StckExch) is a proxy for the popularity of each package. It counts the number of highly ranked (score > 20) questions related to each package. Developers often search for answers to issues they face and may stumble upon one of these packages presented as a solution to a problem they are facing, thus increasing the chances that they may adopt that technology. From a social contagion perspective this would increase "exposure". We avoid counting the total number of questions because most of the questions tend to be of low score[10] and the search engines may avoid including links to them, thus they do not increase "exposure." According to personal experience of all authors, search engines tend to avoid including links to questions with low score if questions of higher ranks are available. This factor may be interpreted from a rational perspective (leveraging experience of others when lacking other information), but more appropriately, it is a great example of social bias since the developer did not engage in due diligence, instead relying on social cues to make a technical choice. **StckExch** was obtained by selecting all relevant posts in the StackExchange dump (2017-12-08), which have `data.table` and `tidy` in the post's title and body (including code snippet if any). Manual inspection found that posts without R-language tag '<r>' tag in the 'Tags' field were not relevant and we excluded them. Furthermore, we only selected posts with score above 20. Again, we counted posts created prior to the end point for each adopter project.

Project proximity to `data.table` and `tidy` in dependency network (Prx2DT/Prx2TD) measure dependency networks and can be understood from the perspective of software supply chain networks. Based on the characteristics of the 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, in turn, to further downstream packages.

We consider all downstream packages of `data.table` and `tidy`, e.g. those in the `data.table` and `tidy` clusters respectively. We hypothesize that if a project installed a package within the `data.table` cluster, then the project is more likely to install `data.table` than `tidy`. The rationale of such a hypothesis is that if developers installed a package because of 1)preferences for some of its functionalities or features inherited from an upstream package or 2) the way such a package works, which is sometimes influenced by or derived from an upstream package, then it is more likely that these developers will gravitate toward the upstream package over other alternatives.

Based on the dependencies of R CRAN packages, the clusters of `data.table` and `tidy` are easily constructed. More specifically, we used the METCRAN[11] API and scraped meta data for more than 11K R CRAN packages for which dependency information is available. Table 2 summarizes basic information on the networks that were constructed and more detailed information on the methodology follows.

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

- for each downstream package, only the relative

10. In total, only 131/1666 data.table related questions and 162/1785 tidy related questions have score larger than 20

11. https://www.r-pkg.org/about

TABLE 2: Network characteristics

Characteristics	data.table	tidy
# downstream packages	813	2203
# downstream layers	5	5
# of packages in common	636	
overlap ratio	0.78	0.28

weight to the 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 the `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. 3, various maps among Git objects have been created. By utilizing maps of project-To-commit, commit-To-blob, and blob-To-content in sequence and selecting the install statements in blob content via regular expressions similar to those mentioned in Sec. 3.2, 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 computed:

$$\begin{cases} P_{pd} = \Sigma_a^{L_p} W_{ad} \\ P_{pt} = \Sigma_a^{L_p} W_{at} \end{cases} \quad (1)$$

To summarize the process described above, we first measured the weight of each downstream package in either `data.table` or `tidy` by leveraging the R package dependency networks and the formulas above. Secondly, by following a similar flow in Fig. 2, we extracted all R packages that were adopted in the commits prior to end point where one of the focal packages was first adopted. Finally, we calculated the proximity to `data.table` and `tidy` by summing up the weights of all downstream packages for each project. Notice that a project's downstream packages that were not in `data.table` or `tidy` downstream set were dropped.

Project proximity to `data.table` and `tidy` in authorship network (AthPrx2DT/AthPrx2TD) represents the proximity of a developer to a focal project as measured through their author network. It 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 behavior 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. Authors of other projects that are directly exposed to authors of `data.table/tidy`, are identified by applying a project-To-author map to both `data.table/tidy` packages separately and indirectly-exposed authors are obtained by combining the map of author-To-project and the map of project-To-author serially and then applying it on each directly-exposed author.

We classify authors exposed to `data.table` into the `data.table` author cluster and those exposed to `tidy` into the `tidy` author cluster. Projects/observations may have authors who are in either of these two clusters and these authors may influence the choice of data frame technology, i.e., (`data.table` vs. `tidy`). In order to estimate the impact of every author in each cluster, we use the following weights,

- $W_{bd} = 1, W_{bt} = 0$ 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 the 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:

$$\begin{cases} PA_{pd} = \dfrac{\Sigma_b^{A_p} W_{bd}}{N_p} \\ PA_{pt} = \dfrac{\Sigma_b^{A_p} W_{bt}}{N_p} \end{cases} \quad (2)$$

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., *AthPrx2DT/AthPrx2TD*, is the overall influence of exposed authors on a project p. Notice that *AthPrx2DT/AthPrx2TD* 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 *AthPrx2DT/AthPrx2TD*. By normalization we remove this bias induced by the difference in the number of authors for different projects. This factor falls clearly within a realm of a social bias. It may also be partially explained as cognitive bias if the developer is not aware of alternative choices.

To summarize the computation of proximity through authorship network, we started by measuring the weight of each author who was either a co-author of `data.table/tidy` or had cooperated with at least one of the authors of `data.table/tidy`, which was detailed above. Then we summed up the weight of every author of a project and normalized it over the total number of authors in this project. Again, here we applied end point filter on every step in calculation.

Time gap between the raise of an issue and the first reply (RplGp) measures how fast developers or maintainers of a package respond once an issue has been raised. The timeliness of this response reflects the efficiency of package maintenance and can be attributed to the 'infectiousness' category of social contagion theory and could clearly be of interest for those deciding on which package to adopt.

The calculation of reply gap is worth discussing. We are interested in understanding how long it takes for an issue to get its first reply after being reported. For each individual in the study, we focus on the time period just before the key commit that includes the choice of focal package (`data.table/tidy`). However, several additional obstacles that needed to be addressed in order to measure the reply gap :

1) It is rare that an issue was raised simultaneously with the key commit (inside which either the`data.table/tidy` package is installed).
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 representative is not reasonable
3) For some issues, it took a significant amount of time to get a reply and in some cases no reply was ever made to an issue, thus, averaging reply-time to previous issues is problematic due to long right-censored cases.

This is a case where statistical models for survival(time-to-event) are appropriate. In this scenario, an issue can be viewed like a patient under study with the first reply analogous to conclusion of the medical issue or death of the patient. We aim to model the time until reply to the reported issue, i.e., the survival time of the issue, with shorter lifetimes indicating a more interactive development team. Irrespective of package, for each issue, we record the time that it was submitted (timestamp recorded when the issue is raised) and use survival analysis to model the distribution of the issue lifetimes for each package (`data.table/tidy`) using the R package 'survival' [40]. Predictions for the reply time for each project (observation) can be made based on data collected before the key commit. The *RplGp* for a project is simply the median issue lifetime for an issue generated before a key commit. This factor appears to be clearly related to rational choice factors as the delays in response may cause real problems.

In practice, we extracted all issues of `data.table/tidy` from GitHub and measured difference between the time an issue is first raised and the first response time. As described above, we trained a survival model to estimate the distribution of the delay until first response delay. The model was trained using all issues that had been raised before the current package key commit. Those issues that had not been responded to yet were right censored in the model fitting. The reply gap represents the median value of response times.

In summary, we note that for each project that eventually adopts one of the two focal packages (`data.table/tidy`), all of the variables described in this section are calculated dynamically using only data that occurs before the key commit. In addition, for each observation, every predictor with choice property (Table 1) needs to be calculated for both packages, e.g., *Unrslvd* needs to be calculated for both `data.table` and `tidy`. These will end up being denoted as *Unrslvd.datatable* and *Unrslvd.tidy*.

5 Results

5.1 Result of data.table VS. tidy

Table 3 summarizes basic statistics for independent variables analyzed in the model. We use the R package 'mlogit'[12] [41] to fit the model using the 11 predictor variables defined above with the response being an indicator of the package chosen.

Very high correlations among predictors (above 0.9) occurred between *Prx2DT* and *Prx2TD*. High correlations may lead to unstable and difficult 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 remain stable if this approach is reversed. Table 4 presents the resulting model fit.

Below we summarize findings for each predictor variable separately.

 # related questions on StackExchange with high score (StckExch): the coefficient is 0.2, indicates that the number of high score questions on StackExchange is associated with the likelihood that a project would adopt the respective technology. The association is positive, holding other factors equal. For illustration, if the number of high score questions increases by 6 questions (1 std. dev.) from a median value of 130 for `data.table`, the estimated probability of choosing `data.table` increases from 0.58 to 0.87, while holding all other predictors at their median values.

12. https://cran.r-project.org/web/packages/mlogit/vignettes/mlogit.pdf

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.2993758, IEEE Transactions on Software Engineering

TABLE 3: Summary Statistics for Independent variables (data.table VS. tidy)

Variable	median	mean	std.dev
Cmts	3	46.83	645.68
Aths	1	2.13	8.23
C (boolean)	0	9.79e-03	9.85e-02
Prx2DT	0	0.15	0.95
Prx2TD	0	0.62	2.79
AthPrx2DT	0	6.99e-2	0.17
AthPrx2TD	0	0.11	0.24
CumNum.datatable	2.72e+03	2.66e+03	1.87e+03
CumNum.tidy	305	8.44e+02	9.12e+02
RplGp.datatable	2.09	2.16	0.33
RplGp.tidy	3.03	2.95	0.53
Unrslvd.datatable	0.29	0.28	3.23e-02
Unrslvd.tidy	0.20	0.16	7.7e-02
StckExch.datatable	130	125.76	6.53
StchExch.tidy	158	152.57	10.14

TABLE 4: The Fitted Coefficients. (data.table VS. tidy) $McFadden$ [22] $R^2 = 0.14$ $n = 7k$

Variable	Estimate	Std. Error	p-val
tidy:(intercept)	-6.07	0.28	2.20e-16
CumNum	1.56e-04	1.44e-05	2.20e-16
Unrslvd	2.45	0.78	1.59e-03
RplGp	-0.38	4.88e-02	5.11e-15
StckExch	0.27	1.26e-02	2.20e-16
tidy:Cmts	-3.95e-04	2.22e-04	7.54e-02
tidy:Aths	-3.02e-04	7.12e-03	0.97
tidy:C	-0.67	0.28	1.82e-02
tidy:Prx2TD	0.17	2.87e-02	6.79e-10
tidy:AthPrx2TD	1.27	0.14	2.20e-16
tidy:AthPrx2DT	-7.06e-02	0.19	0.72

This result aligns well with the social contagion theory that posits that increased adoption is a consequence of increased exposure. Surprisingly, including an additional predictor that counts the total number of questions (of high and low score), shows no statistical significance. 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 score. The number of posts (questions), may, therefore, not be a good proxy of exposure. As such, the total number of posts of low-score questions, in fact, appear to discourage developers from using a package.

Finding 1: *We found that exposure measured via the total number of questions on StackExchange had no impact on adoption, while the number of high score questions has a strong and positive correlation with increased adoption.*

 # open issues over all issues (Unrslvd): the coefficient is 2.5, indicating that the higher the ratio of unresolved issues a package has, the more likely it would be adopted. While it appears to be counter-intuitive from the perspective that unresolved issues may indicate a lack of attention from maintainers. However, the causal relationship may go the other way: the increased interest from users when a package becomes popular among developers, may lead to more contributions in the form of issues, and may exceed packages developers' processing capacity, which results in a larger ratio of open(unresolved) issues. In short, the ratio of unresolved issues over all issues might indicate high rates of adoption especially in the early stages of the projects when the total number of issues is low.

Finding 2: *We found that infectiousness of a package as measured via the fraction of unresolved issues, is associated with a higher adoption rate for that package.*

Project proximity to `tidy` in authorship network (AthPrx2TD): the coefficient is 1.3, indicating that the closer a project is to authors of the package `tidy` vis-a-vis the author network, the more likely they are to choose `tidy` over `data.table`. If the proximity to `tidy` in the 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 estimated probability of choosing `tidy` increases seven percent from 0.42 to 0.49. This finding supports the basic premise of the social contagion hypothesis that developers' choices are affected by the environment they are in.

Finding 3: *Proximity as measured by the fraction of authors who are either developers of the package to be adopted or who work with at least one developer of that package, increase the chances of adoption.*

This may be a consequence of authors who have direct expeience or are familiar through word-of-mouth. However, **Project proximity to `data.table` in authorship network (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 the `data.table` community. Furthermore, the exposure in the `tidy` community may come from a much larger set of packages in the `tidyverse`, while `data.table` does not have an equivalent brand that involves a wider variety of tools beyond data handling.

Association of c code with a project (C): the coefficient is -0.6, indicating that a project containing at least one C file is less likely to choose `tidy`. The estimated chances of choosing `data.table` increase by 15 percent from 0.58 to 0.73. The finding is consistent with our hypothesis that if an R project has a need for performance, as evidenced by the use of functionality being developed natively in the C language, then it is more likely to choose the higher performance of `data.table`.

Finding 4: *Proximity, as measured by the project's need for performance, is associated with adoption of packages that emphasize high performance*

Time gap between the raise of an issue and the first reply (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 0.21 days (1 std. dev.) from a median value of 1.4 for `data.table`, the estimated chances of a project choosing `data.table` decrease by two percent from 0.58 to 0.56 assuming all other variables remain at their 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.

> ***Finding 5:*** *Infectiousness of a package as measured by speed of response to issues is associated with a higher adoption rate for that package.*

Project proximity to `tidy` in dependency network (Prx2TD): the coefficient is 0.2, indicating that the closer (through a dependency network) a project is to the package `tidy`, the more likely its authors are to choose `tidy` over `data.table`. If proximity to `tidy` in dependency network increases by one standard deviation of 2.8 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 0.42 to 0.54. It supports our hypothesis that the supply chain influences projects' choices. A project tends to install a specific package if it has already installed other packages that also depend on it, i.e., if a project uses downstream dependencies of a package, it is more likely to use the package itself rather than other alternatives. Being familiar with downstream packages may reduce the overhead or learning curve required for an upstream package, leading to an advantage over other choices.

> ***Finding 6:*** *Proximity to a package as measured via technical dependency networks is associated with a higher adoption rate.*

 # projects of deployments (CumNum): the coefficient is 1.4e-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, 1870 projects, from a median value of 2660 projects for `data.table`, the estimated chances of choosing `data.table` go up by seven percent from 0.58 to 0.65 for a project holding 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 justifies adoption of the widely deployed package as predicted by the social contagion theory.

> ***Finding 7:*** *Exposure to a package that is widely deployed is associated with a higher adoption rate.*

We also find that the number of authors in the adopting project does not affect the choice of 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` making the variation in the number of authors statistically insignificant.

> ***Finding 8****: We did not find statistically significant association between infectiousness as measured via the number of commits and adoption propensity.*

We achieved a $McFadden$[13] R^2 of 0.14, which is a good fit according to McFadden [22]. (Notice that the R package 'mlogit' use $McFadden$ R^2 instead of R^2 to estimate fitness of the model because logit models do not generate the sums-of-squares needed for standard R^2 calculation.)

Regression 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 Table 4 on nine parts and predicting on the remaining part yielded a reasonable AUC of 73%. Average accuracy was 70% with balanced Type I and II errors (obtained by choosing predicted probability cutoff of 0.49).

Finally, it is worth noting that out of six predictors that were statistically significant, only **CumNum, RplGp,** and **C** were clearly grouped into predictors that would support rational choice. The remaining three predictors primarily reflect a mixture of social and cognitive biases associated with social preference or default choice when alternatives are not known. If we include the effort needed to obtain the necessary information into the utility function, these social and cognitive biases can, of course, be explained rationally as well.

Based on the findings mentioned above, we derived a list of recommendations for package developers and users: To increase the popularity of a package, package developers should maintain quick response to users' questions and concerns such as raised issues on development platform; package development team should involve active developers, especially authors of existing packages, to have a broad author network; package developers should choose popular packages among alternatives (when there is a need) to use. To choose a popular (right) package among alternatives, users should choose the package with more high score questions on Q&A website such as StackExchange; users should choose the package with better maintenance such as response speed to raised issues and questions on development platform; users should choose the package that is more widely used. Counter-intuitively, packages with a

13. https://stats.stackexchange.com/questions/82105/mcfaddens-pseudo-r2-interpretation

large number of outstanding issues should also be preferred as they might command a more active or larger contributor community.

5.2 Result of Angular VS. React

The basic statistics are shown in Table 5. We fitted the same model on this dataset and show the result in Table 6. (Notice that we use abbreviation 'AG' for `Angular` and 'RT' for `React`)

TABLE 5: Summary Statistics for Independent variables (Angular VS. React)

Variable	median	mean	std.dev
Cmts	1	13.23	315.07
Aths	1	1.21	3.12
C (boolean)	0	8.81e-04	2.96e-02
Prx2AG	0	7.54e-05	1.22e-02
Prx2RT	0	2.19e-04	1.65e-02
AthPrx2angular	0	0.14	0.25
AthPrx2react	0	0.14	0.25
CumNum.angular	1.0e+05	7.66e+04	3.35e+04
CumNum.react	5.05e+04	6.85e+04	6.27e+04
RplGp.angular	0.47	0.49	0.12
RplGp.react	0.44	0.44	4.83e-02
Unrslvd.angular	5.77e-02	7.42e-02	3.37e-02
Unrslvd.react	7.83e-02	9.19e-02	5.04e-02
StckExch.angular	3272	3.03e+03	4.65e+02
StchExch.react	1213	1.05e+03	2.91e+02

TABLE 6: The Fitted Coefficients. (Angular VS. React) $McFadden$ [22] $R^2 = 0.45$ $n = 292k$

Variable	Estimate	Std. Error	p-val
react:(intercept)	-12.68	0.10	2.20e-16
CumNum	2.02	1.08e-02	2.20e-16
Unrslvd	-27.28	0.39	2.20e-16
RplGp	-0.50	5.92e-02	2.20e-16
StckExch	-7.27e-03	4.95e-05	2.20e-16
react:Cmts	2.05e-04	2.73e-05	6.48e-14
react:Aths	-3.94e-02	3.24e-03	2.20e-16
react:C	-0.24	0.19	0.21
react:Prx2RT	-1.67	0.32	3.04e-07
react:AthPrx2RT	1.37	3.03e-02	2.20e-16
react:AthPrx2AG	0.18	2.83e-02	7.32e-11

Below our findings are briefly summarized.

CumNum (total number of projects using the package), **RplGp** (time it takes to respond to user issues) are both significant and have coefficients pointing in the same direction as the `tidy` and `data.table` study.

Unrslvd (# open issues over all issue) is significant but the coefficient is negative, which indicates that unlike in the case of `tidy` and `data.table`, developers prefer to adopt a technology with the lower fraction of unresolved issues. The reason such difference is exhibited may be due to the fact that both packages have strong commercial backing, so the developers may take this as a signal that a company is not as supportive of the product. In R ecosystem, `data.table` is a completely volunteer-supported operation, hence developers may be more forgiving to the limitations of the maintainers.

StckExch (the number of related questions on StackExchangewith high score) is significant but, unlike in R ecosystem, has a negative coefficient. Developers in Javascript domain may believe that a technology with more posts on StackExchange may be harder to use or more complex. Developers in different domains may have different opinions regarding the 'exposure' on StackExchange and further research on these differences would be needed to better understand this.

From Table 6, we can also find that a project with larger numbers of commits and fewer contributors are more likely to adopt `React` than `Angular`. One explanation could be that productive developers may prefer `React` over `Angular`; thus, a more productive project (commits count over contributor count) will be more likely to adopt `React`. These differences may also reflect the different nature of projects adopting each framework that we have not been able to capture using existing variables.

Among the four 'proximity' measures, only the proximity to `React` through author network is statistically significant. Given the complexity of the dependencies in JavaScript domain, the large size and the independent nature of the packages, the dependency network may not be as important when making a choice.

5.3 Generalizability between the Javascript and R domains

The two case studies were intended to test the generalizability of social contagion models in two distinct contexts that varied in terms of domain functionality, user base, scale, and primary governance styles. By comparing Table 6 with Table 4, we can conclude that despite these radical differences in context there are substantial similarities.

CumNum (total number of projects using the package), and **RplGp** (time it takes to respond to user issues) have the same and expected impact on adoption: more exposure from wider past adoptions and higher infectiousness from better response rates were associated with increased rates of adoption in both case studies.

However, **Unrslvd** (the ratio of unresolved to all issues at the time of adoption) had an expected negative effect in the `data.table/tidy` study, but had a counter-intuitive positive effect in the `Angular/React` study. Similarly, **StckExch** (the number of related questions on StackExchange with high score) switched from having a positive impact to having a negative impact on the chances of adoption.

Our conjecture is that the operationalization of project quality (infectiousness) as expressed by the fraction of unresolved issues and the exposure to StackExchange high-score questions may have limitations. Specifically, `Angular/React` have experienced a spectacular growth over the considered period. We presume that this growth was fueled by a heavy commercial involvement, which, in turn, motivated the adoption and created an inflow of issues that could not be resolved on a timely basis.

Similarly, the rapid adoption was not matched by the corresponding increase in the number of high-score questions on StackExchange, resulting in the negative coefficient.

Furthermore, it may be the case that the developers in the `Angular/React` (Javascript) domain may believe that a technology with more posts on StackExchange may be harder to use or more complex, unlike the developers in the data science domain where users expect a fair amount of

esoteric and quirky functionality; thus, good questions on StackExchange are helpful and necessary even when faced with relatively common tasks. This suggests that further investigations are needed on how developers in different domains may react to 'exposure' on StackExchange.

These differences suggest that the social contagion models may need to be adjusted for cases where the adoption is primarily driven by the commercial involvement that may manifest itself in directly funding developers, providing better training, or even by dangling good job opportunities. Specifically, large companies, may create direct signals driving adoption that do not propagate via social and technical networks and may lead to counter-intuitive values of the model coefficients.

The remaining variables represent the characteristics of the adopter, and cannot be directly compared between the studies. Interpreting them, however, is instructive for potential applications related to understanding the adopter base.

Specifically, while larger projects (measured by the number of commits **Cmts**) prefer `data.table`, presumably due to its stability and ability to handle larger data sets, in the User Interface domain, larger projects tend to use `React`, presumably because it is, according to opinions of knowledgeable developers we have consulted, designed to support large projects and applications. We had a separate predictor **C** indicating the need for performance in the R ecosystem, and, as expected, it had no effect in the Javascript domain where C language is not used to improve performance via native methods as is common for R domain.

From the perspective of the technical network **Prx2TD/RT**, the proximity to `tidy` increases the chances of adoption, presumably due to the rich functionality of the *tidyverse* framework that can be exploited for tasks other than simply doing data management. `data.table`, in contrast, does not have as wide an ecosystem of its own, thus proximity in the technical network does not appear to bring any distinct advantages. The case of `React` vs. `Angular` is unusual as the proximity of projects to each of these frameworks is highly correlated, $\rho = 0.9$. That is, if the proximity in the technical network is high for one of these frameworks it is also high for another. This may reflect the possibility that each of these frameworks are fairly comprehensive for the intended functionality and, therefore, the proximity is based on other JavaScript technologies that may, in fact, be more closely aligned with the `Angular` ecosystem. It is, therefore, important to note that the proximity in the technical network may not always fully reflect the actual interdependencies between the technologies and future work is needed to explore how general this relationship may be. Another possible scenario is that this negative relationship is simply an artifact indicating that `React`'s adoption may have been more strongly influenced by direct marketing signals (such as job opportunities) and the technical networks did not co-evolve at the same rate as the adoption.

From the perspective of social networks **AthPrx2TD/RT**, the proximity to `tidy` increases the chances of adoption, presumably due to the charismatic leadership of the founder and lead developer who also contributes to many other data science projects. The founder of `data.table`, in contrast, does not have as commanding a presence and social following in the community. The proximity to React within a social network increases the chances of adoption while the proximity to `Angular` decreases the adoption of `Angular`. There is some anecdotal evidence that React gets more positive testimonials from its users, see, e.g, "Is React killing Angular?" from Quora[14]. In such a case, knowing someone who actually uses `Angular` and can testify to the potential complexities and effort needed to use it fluently, may discourage adopters from trying it. If the `React` users are much more positive in their testimonial, this would explain the power of the social contagion model to discriminate between such differences.

Further considerations on the generalizability and other limitations of our approach are discussed in the next section.

6 Limitations

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

6.1 Limitations to Internal Validity

To obtain an unbiased, representative characterization of technology spread, we examined a very large collection of projects. While large, our sample however is not complete, as many projects do not publish their code and our data collection process may have missed even some of the public projects. The sample we have limits the findings of this study to projects that share their version control data on one of the many forges, such as GitHub, BitBucket, GitLab, Bioconductor, SourceForge, etc. However, our project repository may not be representative of the entire universe of projects, especially projects that do not publish 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 that we used to identify instances of package usage or installation can capture most of the install statement in the .r/R file, however, in some cases the install statement may be missed due to a dynamic specification in the installation such as in the case below,

```
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)
```

Also, multiple packages may be wrapped into a variable before calling the install function:

14. https://www.quora.com/Is-React-killing-Angular

```
load.lib<-c("EIAdata", "gdata", ...,"stringr",
    "XLConnect",
"xlsReadWrite","zipcode")
install.packages(lib,dependences=TRUE)
```

Moreover, regular expressions may occasionally falsely capture an install statement, e.g., install statements that are commented out may, in rare cases, be captured by regular expressions. Files that are contained in a project but not used may also contain installment statements that are captured by regular expressions. To alleviate this potential issue, we used the R language requirement to have a comment character '#' on each line and ensured that the matched install is never preceded by the comment character.

Another threat to validity is that the import of a package may not always indicate that the package was actually used. In some cases, a project may only contain package import statements without any calls to package API. For example, a developer may have dropped all actual API calls in the code without removing the corresponding package import statement. Alternatively, a developer may only add an import statement as a place holder for future usage. One approach to mitigate such potential inaccuracies is to identify if any package APIs were called in the project. However, this approach may not cover all possible cases. One such scenario can be that multiple packages may contain one or several common APIs (e.g., the predict API in R exists in multiple packages) and it is generally not possible to automatically identify with precision which package an API belongs to.

We used keyword filtering on both tag field and on the text of the post itself to find the relevant posts from StackExchange dump. We emphasize that a manual inspection was needed as a followup step to ensure the relevance of the posts. While we did a manual inspection of our data, the approach we used may require a lot of manual effort to check the relevance if extremely popular technologies are investigated.

These potential limitations may affect the dependency networks we construct and result in an imprecise count of the number of projects using our two focal packages. Moreover, developer identities may not be consistent across our data sources, which may affect the author network [42]. We have tried to address these and other issues encountered when dealing with operational data from software repositories and big data in accordance with guidelines provided in the literature [43], [44], [45].

It is important to note that the particular operationalizations of the concepts from social contagion theory represent only one possible approach. Measures are not entirely orthogonal, i.e., each measure may capture the aspects of other dimensions beyond the one it is intended to measure. The correlations among predictors may lead to unstable models that are hard to interpret. We address this limitation by carefully considering various interpretations of the measures, conducting exploratory analyses of the obtained measures, selecting a subset that does not pose threats to model stability, and investigating compliance with model assumptions including inspection of 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 would need to include the third option, i.e., projects choosing both packages: `tidy` and `data.table`. We fitted a variety of alternatives models to ensure that the reported results are not affected by these variations in the approach. We only present here the results for two alternatives due to space considerations, but we have applied our choice model to several other R packages as well.

6.2 Limitations to External Validity

We demonstrate how to use social contagion modeling with version control data to evaluate developer behaviour when choosing software packages. The particular results we obtained for R and the two focal packages may not, therefore, generalize beyond this specific context. We evaluated the generalizability of the results in the JavaScript domain in section 6 and found some variations to the finding in R ecosystem in JavaScript ecosystem. The framework we provided, however, allows future researches to investigate the nuances of developer behaviour in much greater detail and apply it to other contexts.

7 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 [46], [47], [48], [49], [50]. Most prior work, therefore, focused on costs and benefits of library migration [51], [52], [53], [54], [55], [56], [57], [58]. 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, exposure of techniques on StackExchange, quality of support measured through issue number and response times) that are based on sound theoretical foundations 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 [51]. As in prior work we detect usage by searching for library statements in source files of projects [52]. De la Mora *et al.* [53] 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 [59] and reasons [60] behind, to mining

version control and issue tracking repositories [51], [52], [53]. 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.

8 Conclusions

Integrating software supply chain concepts and models to operationalize key variables from social contagion theory to investigate software technology adoption 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 was able to identify factors that were 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. It also validates the measures deemed to be the drivers of technology adoption by the social contagion theory.

This study introduces the concept of two types of software supply chains (based on technical dependencies and on the relationships among developers induced by projects they have worked on) and demonstrates how software supply chains for the entire open source ecosystem can be reconstructed as they have existed at any point in the past from public version control systems. Additionally, by taking a social contagion perspective and employing the logistic regression 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., technology 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 technology 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, technologies with large number of overall adopters, higher responsiveness to new issues, and more high-score stack exchange questions are more likely to be chosen. Furthermore, from the perspective of a project's decision-makers, their technical features and proximity to a technology in both the technical dependency network and author collaboration network increase the probability of adoption. On a more speculative side, we find that half of the significant predictors do not appear to be related to a traditional rational choice, but are likely a reflection of social and cognitive biases or, in plain language, shortcuts people take. Developers, at least in the context of technical decisions regarding which technology to use, do not appear to be immune from these biases.

Source code and data for this study is publicly available [15] to facilitate reproducibility and wider adoption of the proposed methodology.

Acknowledgment

This work was supported by the National Science Foundation NSF Award IIS-1633437.

15. https://drive.google.com/drive/folders/1YjC31l5NrD5XzI5ZyxtRLF29OM2owb1X?usp=sharing

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] Y. Ma, C. Bogart, S. Amreen, R. Zaretzki, and A. Mockus, "World of code: An infrastructure for mining the universe of open source vcs data."

[31] 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.

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

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

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

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

[36] 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

[37] 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

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

[39] 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.

[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] 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

[43] 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

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

[45] 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

[46] 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

[47] 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

[48] 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

[49] 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

[50] 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

[51] 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

[52] 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

[53] 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

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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

[59] 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.

[60] S. Xiao, J. Witschey, and E. Murphy-Hill, "Social influences on secure development tool adoption: why security tools spread," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 2014, pp. 1095–1106.